Prozessorganisation in eingebetteten, ubiquitären Rechnersystemen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Christian Decker

aus Zeulenroda

Tag der mündlichen Prüfung:	13.07.2009
Erster Gutachter:	Prof. Dr. Wilfried Juling
Zweiter Gutachter:	Prof. Dr. Michael Beigl

Danksagung

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter in der Forschungsgruppe am Telecooperation Office (TecO) des Instituts für Telematik an der Universität Karlsruhe (TH). Das Umfeld und meine Tätigkeit prägten mich nachhaltig.

Als Student begann ich an Forschungsprojekten im Bereich Ubiquitous Computing der TecO Gruppe mitzuarbeiten. Insbesondere die unternehmensnahe und praktische Umsetzung der Forschungsarbeiten begeisterten mich; forderte sie doch einen kontinuierlichen akademischen wie auch praxisorientierten Wissenserwerb für den erfolgreichen Brückenschlag zwischen Hochschule und Unternehmen. Mit dieser Begeisterung brachte ich mich ein, um neue Ideen und Ansätze durchzusetzen und zu realisieren. Umgekehrt ließ ich mich von der gleichen Faszination anderer mitreißen. Dies war und ist mein Antrieb, mich fortwährend fachlich wie persönlich weiterzuentwickeln.

Das TecO bot dafür einen idealen Rahmen. Gestaltungswille, Verantwortung für die Ausgestaltung der eigenen Arbeit und Übernahme von Verantwortung im Team wurden stets gefördert. Der internationale Austausch mit anderen Experten und der kritische Diskurs bringen dabei besonders innovative Arbeiten zur Lösung komplexer Fragestellungen hervor. Ein solches Arbeitsumfeld empfinde ich als außergewöhnlich bereichernd.

Zusätzlich nahm ich ab März 2006 die Herausforderung an, die Mitarbeiter der Forschungsgruppe am TecO anzuleiten. Damit verbunden waren Definition und Umsetzung wissenschaftlicher Ziele der Gruppe, Akquise, Koordination und Ausbau internationaler Projektkooperationen mit anderen Forschungsgruppen und Industrieunternehmen, sowie ein Engagement in der universitären Lehre. Die Kombination und erfolgreiche Balance von wissenschaftlicher Arbeit und organisatorischen, bisweilen unternehmerischen, Tätigkeiten beurteile ich als wichtigste persönliche Erfahrung.

An dieser Stelle möchte ich den Personen, die mich auf dem Weg dieser vielschichtigen Ausbildung begleitet haben, meinen Dank ausdrücken. Bei meinem Referenten und Koreferenten, Herrn Prof. Dr. W. Juling und Herrn Prof. Dr. M. Beigl, bedanke ich mich für die wissenschaftlich-technische wie auch organisatorische Unterstützung der Forschungen, die zu dieser Arbeit führten. Im Ergebnis entstanden erfolgreiche internationale Fachpublikationen. Beiden danke ich insbesondere für das entgegenbrachte Vertrauen, als ich neben der wissenschaftlichen Tätigkeit, auch organisatorische Verantwortung für die Forschungsunternehmungen der TecO Gruppe übernahm. Ein besonderer Dank gilt Herrn Prof. em. Dr. Dr.-Ing. E.h. Dr. h.c. mult. G. Krüger, der die Forschungsaktivitäten und das Entstehen dieser Arbeit engagiert verfolgt hat und sich persönlich in hohem Maße für die Forschungsgruppe einsetzte.

Meinen Kollegen Albert Krohn, Tobias Zimmer, Till Riedel und Martin Berchtold bin ich zu außerordentlichem Dank verpflichtet. Die Diskussionen mit ihnen und ihre Einwürfe waren inspirierend und haben mir geholfen, meine eigenen Ideen besser zu formulieren und wichtige Aspekte aufzudecken. Die positive Atmosphäre und der Optimismus während unserer Zusammenarbeit halfen über stressige Arbeitsphasen hinweg.

Die Studierenden am TecO rekrutieren sich aus unterschiedlichen Fachbereichen. Sie leisten durch ihr Wissen, Geschick und enorme Einsatzbereitschaft erfolgsentscheidende Beiträge im wissenschaftlichen Alltag. Besonders hervorzuheben sind die überdurchschnittlichen Leistungen von Emilian Peev, Andrea Sayer und Steffen Melischko. In Seminararbeiten, Studien- und Diplomarbeiten und in den Tätigkeiten als wissenschaftliche Hilfskräfte haben sie die experimentellen Untersuchungen für diese Arbeit ermöglicht. Es freut mich, dass wir nicht nur bei einzelnen Fragestellungen sondern kontinuierlich zusammenarbeiten konnten.

Schließlich möchte ich mich auch bei meinem ganz persönlichen Umfeld bedanken, bei meinen Eltern und meiner Partnerin, die mich stets unterstützt und mein Promotionsvorhaben befürwortet haben.

Zusammenfassung

Das Ziel dieser Arbeit ist es, die periodische, aperiodische, echtzeitfähige und verteilte Prozessausführung ubiquitärer Rechnersysteme unter unbekannten und sich zeitlich verändernden Einsatzbedingungen systematisch zu organisieren, so dass eine verbesserte Informationsverarbeitung ubiquitärer Appliances erreicht wird.

Es können zwei Arten von Prozessverhalten während der Ausführung unterschieden werden: (1). aufeinander wirkende Prozesse, und (2). miteinander interagierende Prozesse. Kooperation ist der Mechanismus zur Organisation von aufeinander wirkenden Prozessen, Kollaboration ist der Mechanismus zur Organisation von miteinander interagierenden Prozessen. In dieser Arbeit wird die Einsicht gewonnen, dass sich diese Organisationsmechanismen einheitlich als lineares, zeitinvariantes Regelkreissystem modellieren lassen. Damit gelingt eine mathematisch geschlossene Beschreibung, die die Prozessausführung unter unbekannten Einsatzbedingungen kontrollierbar hält. Die Prozesse werden dabei als *Black-Box* abstrahiert. Der zentrale Beitrag dieser Arbeit ist die *Theorie der kooperativen und kollaborativen Prozessorganisation* für eingebettete, ubiquitäre Rechnersysteme.

Laufzeitinformationen über Ausführungszeit, Energieverbrauch, Pufferfüllstände und andere Ressourcenbelegungen werden in einen Regler zurückgeführt. Dieser passt das Laufzeitverhalten der Prozesse durch Stellgrößen wie beispielsweise Ausführungsperioden, Operationszyklen und Pufferauslastungen an, um eine zeitnahe Informationsverarbeitung des Gesamtsystems zu erreichen. Dieser Ansatz wurde in dem Laufzeitsystem Particle OS für ubiquitäre Rechnersysteme umgesetzt. Es übernimmt die Regelung für alle Prozesse, sowohl lokal zwischen den Prozessen eines Rechnersystems wie auch verteilt über mehrere Systeme hinweg. Beide Eigenschaften ermöglichen es, dass kooperative und kollaborative Prozessausführungen lokal und verteilt in ubiquitären Systemumgebungen realisiert werden können.

Für jede Prozessklasse werden aus der Theorie der kooperativen und kollaborativen Prozessorganisation neue Organisationsmechanismen hergeleitet. Im Ergebnis werden Verbesserungen des Nachrichtendurchsatzes um Faktor 8 bis 10 und eine um 27% bis 32% kleinere Latenz der Kommunikationsbeziehungen eines kollaborativ organisierten Verbundes von Rechnersystemen nachgewiesen. Die periodischen, aperiodischen und echtzeitfähigen Prozesse sind in der Lage, auf stark speicher-, energie- und rechenzeitbeschränkten Rechnersystemen, ein unbekanntes und veränderliches Datenaufkommen zuverlässig und zeitnah zu verarbeiten. Schließlich erreichen eingebettete, ubiquitäre Rechnersysteme *bestmöglich* eine vorgegebene Betriebsdauer.

Das konsistente Bild aus Theorie, Simulation und experimenteller Evaluation in dieser Arbeit lässt ein leistungsfähiges Rahmenwerk der Datenverarbeitung in ubiquitären Rechnerumgebungen entstehen.

Inhaltsverzeichnis

1. Einleitung

In seinen grundlegenden Artikeln [1][2] über Ubiquitous Computing entwirft Mark Weiser ein neues Paradigma von Rechnersystemen und deren Informationsverarbeitung. Die Benutzer sind von einer Vielzahl von Rechnersystemen umgeben und nutzen deren Dienste. Dabei interagieren unterschiedliche Geräte zusammen, tauschen Informationen aus und verarbeiten sie jeweils spezifisch. Beispiele prototypischer Realisierungen solcher ubiquitären Informationsumgebungen finden sich in den Szenarien von ParcTab [3], AwareOffice [4], AwareHome [5], CoBIs [6] und eSeal [7].

Ubiquitäre Rechnersysteme, wie sie in dieser Arbeit betrachtet werden, sind Miniatursensorsysteme, die in Objekte und die Umgebung eingebettet sind und wie in Abbildung 1.1 dargestellt vielzählig den Nutzer umgeben. Die zentrale Recheneinheit ist ein Mikrocontroller, der von Peripheriekomponenten wie Sensoren, Speicher und einer drahtlosen Kommunikationsschnittstelle umgeben ist. Prozesse implementieren

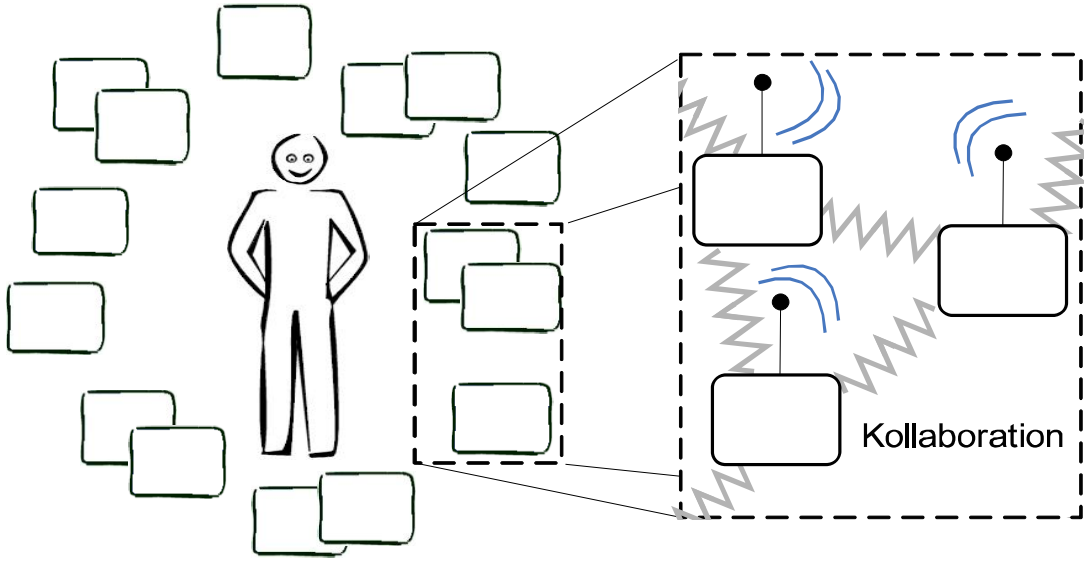


Abbildung 1.1: Kollaboration durch die miteinander verbundene Informationsverarbeitung eingebetteter, ubiquitärer Rechnersysteme ermöglicht gemeinschaftliche Dienste. Abbildung zu Teilen entlehnt aus [8].

die Funktionalität der Rechnersysteme als ausführbare Programmteile auf Mikrocontrollern. Sie realisieren die Informationsverarbeitung in ubiquitären Rechnerverbünden. Dabei sind Mechanismen der Zusammenarbeit und des Zusammenwirkens der Prozesse notwendig. Diese Arbeit formuliert die prozessorientierte Informationsverarbeitung eingebetteter ubiquitärer Rechnersysteme in einem einheitlichen Rahmenwerk von Kooperations- und Kollaborationsmechanismen.

In der Abbildung 1.1 ist neben der drahtlosen Kommunikationsverbindung zwischen den Rechnersystemen noch eine weitere Verbindung gekennzeichnet. Kollaboration koppelt die Ausführungen der Prozesse auf den unterschiedlichen Rechnersystemen miteinander. Die Rechnersysteme arbeiten in einem Verbund, der gemeinschaftlich Dienste erbringt.

An dem praktischen Beispiel von Collaborative Business Items (CoBIs) [6] soll dies illustriert werden. An den Behältern mit Chemikalien aus Abbildung 1.2 sind miniaturisierte Rechnersysteme angebracht. Der Verbund der eingebetteten Systeme ist in der Lage, verschiedene Gefahrensituationen durch gemeinschaftliche Zusammenarbeit zu erkennen. So können die korrekten Lagerungsbedingungen oder feuer- und explosionsgefährliche Materialkombinationen unmittelbar durch die ubiquitären Rechnersysteme erkannt werden. Dazu werden unterschiedliche Prozesse zur Über-

Abbildung 1.2: Kollaborative ubiquitäre Rechnersysteme zur Erkennung von Gefahrensituationen. Links: Einfacher Versuchsaufbau, Rechts: Implementierung in chemischer Industrieanlage (BP, Hull in England).

wachung der Umweltbedingungen, zur Kommunikation mit anderen ausgebrachten Rechnersystemen in der Umgebung und zur Erkennung und Behandlung von verschiedenen Alarmsituationen ausgeführt. Zusätzlich existieren noch eine Reihe von herstellerspezifischen Prozessen zur Aufzeichnung von Ereignissen für ein späteres Audit der Behälter auf ihre Einsatztauglichkeit. Kooperation zwischen den Prozessen eines einzelnen Rechnersystems stellt sicher, dass kein Prozess dauerhaft ausgelassen wird. Dies garantiert die Abarbeitung der verschiedenen Aufgaben unter einem Minimum an Administration für solche Szenarien. Durch den Austausch und die Verwertung von Nachrichten der beteiligten Rechnersysteme untereinander kollaborieren die Prozesse zur Erkennung der Gefahrensituationen miteinander. Mit CoBIs konnte gezeigt werden, dass Warnungen zur Einhaltung von Arbeitsschutzbedingungen wie auch das Kommunikationsaufkommen effizienter gestaltet werden können [6].

1.1 Problemstellung und Motivation

In seinem Artikel „The Coming Age of Calm Technology“ [9] skizziert Weiser das Bild von einer Vielzahl - er spricht von einigen hundert - verschiedenartiger und in die Umgebung eingebetteter Rechnersysteme, mit denen ein Benutzer ungehindert interagiert. Herausragendes Merkmal ist, dass die funktionserbringenden Rechnersysteme in solchen Umgebungen *nicht vereinzelt wahrgenommen* werden, sondern quasi für den Nutzer verschwinden. An ihre Stelle tritt eine rechnerinstrumentierte Umgebung.

Eine besondere Herausforderung besteht im Umgang mit unbekannten Einsatzbedingungen. Die Anzahl umgebender Systeme, Laufzeiteigenschaften und das Datenaufkommen von informationsverarbeitenden Prozessen sind zur Entwurfszeit nicht oder nur ungenau bekannt und unterliegen während des Betriebes ständiger Veränderung. Diensterbringung und Anwendungsfunktionalitäten sind gefährdet, da das Verhalten der Rechnersysteme als unverständlich, nicht systematisch und nicht mehr nachvollziehbar erscheint. Die ubiquitäre Rechnerumgebung zerfällt in der Nutzerwahrnehmung wieder in Einzelsysteme. Die Forderung nach einem geringen Wartungs- und Administrationsaufwand verschärft diese Problematik zusätzlich.

Das zentrale Problem ist die fehlende Kontrollierbarkeit des Verhaltens der Informationsverarbeitung von ubiquitären Rechnersystemen in unbekannten und veränderlichen Umgebungen. Dem soll mit einem *ausführbaren Muster des Zusammenwirkens und der Zusammenarbeit* der Rechnersysteme begegnet werden. Damit ergeben sich neue Herausforderungen an die Prozessorganisation ubiquitärer Rechnersysteme. Die vorliegende Arbeit untersucht vier Prozessklassen:

Periodische Prozesse: Periodische Prozesse sind typisch für repetitive Aufgaben, zum Beispiel das Auslesen von Sensorwerten. Eine zeitnahe Informationsverarbeitung entlang der Verwertungskette der Daten ermöglicht die Einbeziehung und Reaktion auf Änderungen in der Umgebung des ubiquitären Rechnersystems.

Aperiodische Prozesse: Für unregelmäßig auftretende Ereignisse, zum Beispiel Unterbrechungen, müssen Speicher und Rechenressourcen bereitgestellt werden. Auslassungen der Ereignisbehandlung wie auch Beeinträchtigungen anderer Prozesse, zum Beispiel Verzögerungen periodischer Abläufe, sind zu minimieren.

Echtzeitprozesse: Echtzeitprozesse mit Zeitschranken können eingesetzt werden, um ein zeitlich gleichmäßiges Auslesen von Sensoren zu gewährleisten. Das Zusammenwirken mit datenorientierten Prozessen ohne Echtzeitverarbeitung muss koordiniert werden, da anderenfalls Echtzeit-Daten zu spät oder gar nicht verarbeitet werden.

Verteilte Prozesse: Die Zusammenarbeit mehrerer lose gekoppelter Rechnersysteme über ein Netzwerk führt oft zu einer Dominierung durch Systeme mit hoher Kommunikationsrate. Die Erreichbarkeit von Kommunikationspartnern sowie ein ausreichendes Energiebudget für die Aufrechterhaltung des Netzwerkes müssen gewährleistet werden. Bisherige Mechanismen synchronisieren beispielsweise die Netzwerkteilnehmer, berücksichtigen aber keine weiteren auf den Rechnersystemen ablaufenden Prozesse.

Der Einsatz von ubiquitären Rechnersystemen bei minimaler Administration in unterschiedlichen Szenarien verlangt Garantien bei der Abarbeitung von Aufgaben in den Prozessklassen. Die begrenzten Rechnerressourcen erfordern zudem eine möglichst optimale Arbeitsweise, um Ressourcen wie Speicher, Prozessorzeit und Sensorik bestmöglich zu nutzen. Eingebettete ubiquitäre Rechnersysteme verfügen nur über begrenzte Energiereserven. Daher kommt dem Energiemanagement eine besondere Bedeutung zu. Prozesse und Abschaltzyklen müssen zur Laufzeit koordiniert werden, um eine vorgegebene Betriebsdauer bestmöglichst zu erreichen. Methoden des Online- und Offlineprofiling des Energieverbrauchs sind wegen der beschränkten Speicherressourcen nicht anwendbar.

Verschiedenartige Rechnersysteme sind heute in der Lage, miteinander zu kommunizieren, aber für die Zusammenarbeit und das Zusammenwirken in einem Verbund existiert keine ausgearbeitete Lösung. In dieser Arbeit werden die Anforderungen in den vier Prozessklassen durch einen neuartigen, einheitlichen Ansatz adressiert: *Prozesse werden durch regelungsbasierte Kooperations- und Kollaborationsmechanismen effizient organisiert.*

1.2 These und Ziel

Die vorliegende Arbeit stellt einen neuartigen Ansatz der Prozessorganisation für eingebettete, ubiquitäre Rechnersysteme vor. Die folgende These destilliert die gewonnnen Einsichten dieses Ansatzes. Sie werden in der Arbeit bestätigt.

These: Kooperation und Kollaboration sind Mechanismen, die in einem verallgemeinerten Konzept auf alle Prozesse eingebetteter, ubiquitärer Rechnersysteme angewendet werden können. Kooperation und Kollaboration verbessern die Prozessausführung in unbekannten und veränderlichen Einsatzumgebungen. Die Kombination von Kollaboration und Kooperation in einer Kaskade maximiert den Nutzen für ubiquitäre Rechnersysteme.

Ziel ist, die Prozessausführung für die Erbringung von Diensten und Anwendungsfunktionalitäten unter unbekannten Ausführungsbedingungen nachvollziehbar und beherrschbar zu halten. Kooperative und kollaborative Organisationsmechanismen erfassen, bewerten und regeln die Prozessausführung, um den Ablauf der Informationsverarbeitung in ubiquitären Rechnerumgebungen zu verbessern.

1.3 Lösungsansatz: Rückkopplung und Regelung

Prozesse eingebetteter, ubiquitärer Rechnersysteme wechselwirken stark miteinander. Bei der Prozessausführung tritt dabei eine Rückkopplung auf. Das Rückkopplungsprinzip sagt aus, dass die aktuellen Prozessoperationen auf die Ausführung zukünftiger Prozesse zurückwirken. Beispielsweise verbraucht eine aufwendige Datenakquise durch einen Sensor einen Teil des Energievorrates, der für darauffolgende Operationen nicht mehr zur Verfügung steht.

Das Rückkopplungsprinzip stellt online, d.h. zur Ausführungszeit der Prozesse, Informationen zur Verfügung, die eine gezielte Einflussnahme ermöglichen. Das Instrument der Einflussnahme ist ein Regler, der die Anpassungen der Ausführungsparameter bestimmt. Im Ergebnis kann ein *definiertes Muster der Prozessausführung*

erreicht werden. In der jeweiligen noch näher zu spezifizierenden Ausprägung wird diese Art der geregelten Organisation von Prozessen als Kooperation oder Kollaboration bezeichnet.

Rückkopplung und Regelung werden durch ein gemeinsames Laufzeitsystem für alle Prozesse eingebetteter, ubiquitärer Rechnersysteme realisiert. Ein vereinfachtes Modell des Ansatzes ist in der Abbildung 1.3 angegeben. Aktuelle Informationen

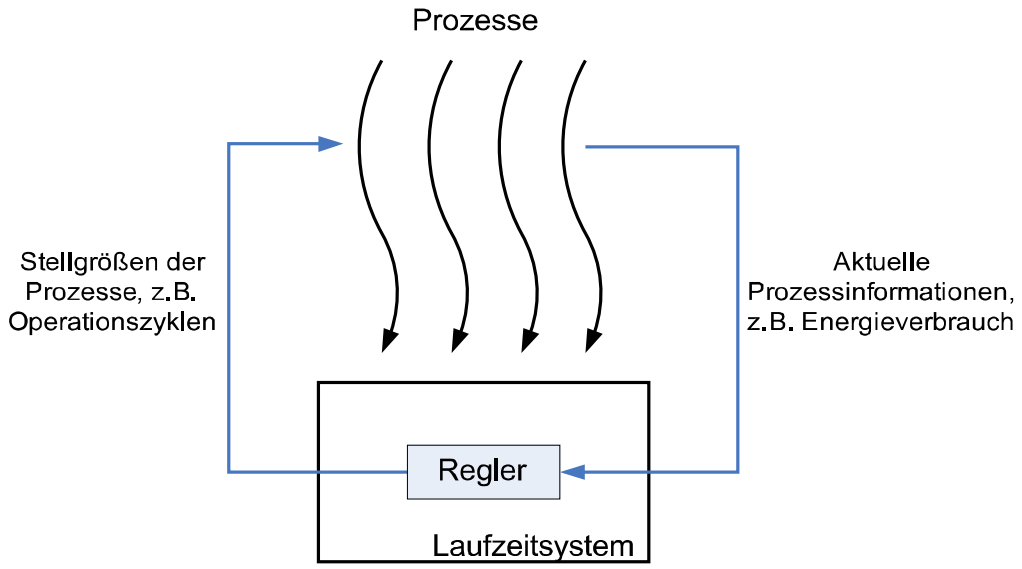


Abbildung 1.3: Lösungsansatz für die Prozesskooperation und -kollaboration.

über Ausführungsparameter wie Ausführungszeit, Energieverbrauch, Pufferfüllstände und andere Ressourcenbelegungen werden ermittelt und dienen als Eingabe in einen Regler. Dieser passt das Laufzeitverhalten der Prozesse durch Stellgrößen wie beispielsweise Ausführungsperioden, Operationszyklen und Pufferauslastungen an, um ein bestimmtes Verhalten des Gesamtsystems zu erreichen.

Die Besonderheit dieses Ansatzes liegt darin, dass ein *gemeinsames* Laufzeitsystem die Regelung für alle Prozesse übernimmt. Die Regelung erfolgt sowohl lokal zwischen den Prozessen eines Rechnersystems wie auch verteilt über Prozesse mehrerer Systeme hinweg. Beide Eigenschaften ermöglichen es, dass Kooperation und Kollaboration für alle Prozessklassen lokal und verteilt in ubiquitären Systemumgebungen realisiert werden können.

Die analytische Beschreibung des Rückkopplungsprinzips und der Regelung wird in der Regelungstheorie diskutiert. Die Eigenschaften kooperativer und kollaborativer ubiquitärer Rechnersysteme werden durch regelungstheoretische Verfahren hergeleitet. Diese Arbeit entwickelt die *Theorie der kooperativen und kollaborativen Prozessorganisation* eingebetteter, ubiquitärer Rechnersysteme und zeigt deren Anwendbarkeit und Nutzen in fünf Bereichen der Prozessorganisation: für die vier Prozessklassen und das Energiemanagement.

1.4 Vademekum

Die Arbeit wird entlang des in Abbildung 1.4 gezeigten Gerüsts entwickelt. Das Vademekum führt den Leser durch die Arbeit. Kapitel 2 bildet das Verständnis und die

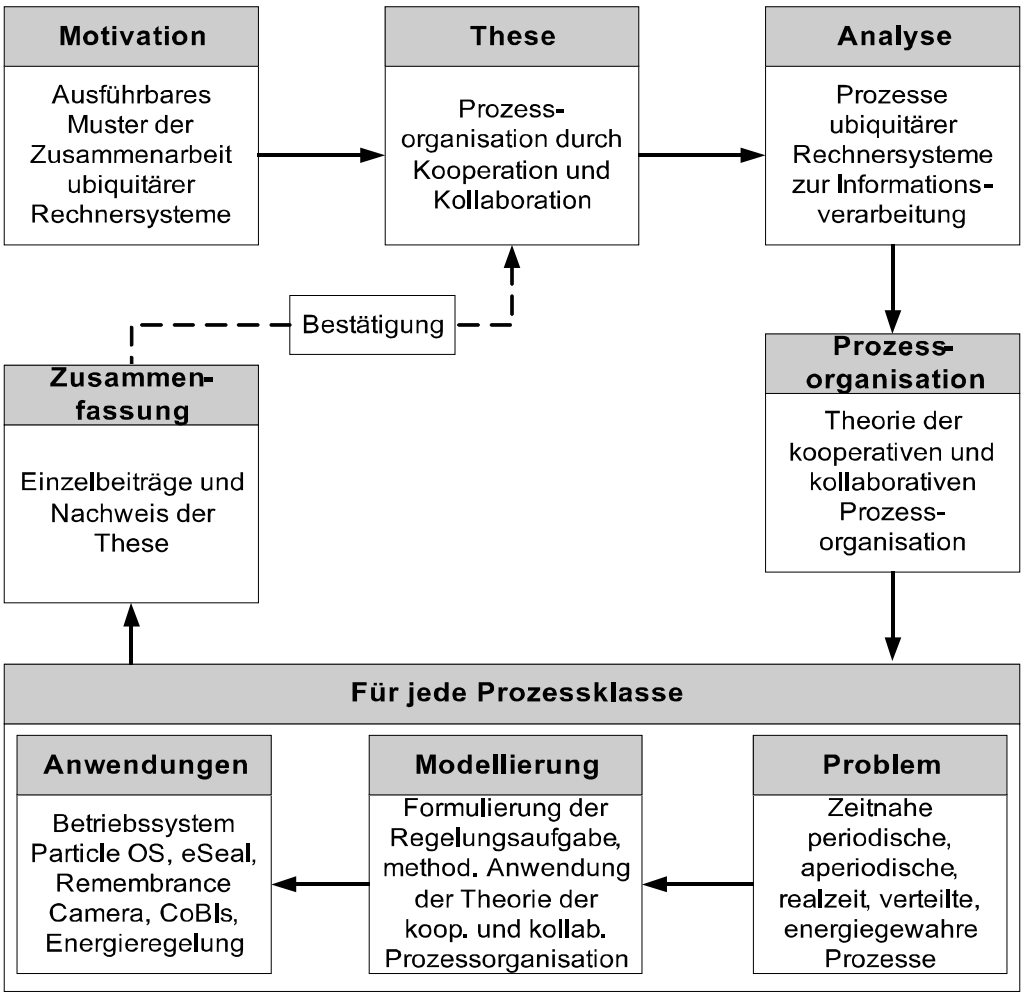


Abbildung 1.4: Struktureller Leitfaden der Arbeit.

Motivation für eine *zeitnahe Informationsverarbeitung* in ubiquitären Umgebungen mit eingebetteten Rechnersystemen, identifiziert die Prozessklassen und skizziert detailliert den Lösungsansatz. Damit sind die Aufgaben für die weiteren Teile der Arbeit festgelegt.

Kapitel 3 bildet die theoretischen Grundlagen der kooperativen und kollaborativen Prozessorganisation. Es wird die allgemeine *Theorie kooperativer und kollaborativer Prozessorganisation* für ubiquitäre Rechnersysteme aus den mathematischen Eigenschaften der Regelkreise entwickelt. Für die Übertragung auf die unterschiedlichen Prozessklassen wird ein Vorgehen spezifiziert. In den folgenden Kapiteln ist zu zeigen, dass sich dieser Ansatz erfolgreich in implementierten Anwendungen verwirklichen lässt.

Die Kapitel 4, 5, 6 und 7 wenden Kooperation und Kollaboration auf die einzelnen Bereiche der Prozessorganisationen an. In den Betrachtungen zu periodischen, aperiodischen, Realzeit-, verteilten und energiegewahren Prozessen findet der Leser immer wieder die *gleiche Herangehensweise* vor. Diese besteht aus folgenden Teilschritten:

- Analyse des wissenschaftlichen Umfeldes für die jeweilige Prozessklasse
- Problemformulierung mit Rücksicht auf ubiquitäre Rechnersysteme aus der Analyse Kapitel 2
- Formulierung der Regelungsaufgabe mittels der Transfermethode aus Kapitel 3
- Mathematische Modellierung des kooperativen und kollaborativen Verhaltens und Angabe des Regelkreises
- Vorhersagen über kooperative respektive kollaborative Eigenschaften der Prozessorganisation
- Anwendungsbeispiele respektive Simulationsstudien, die die Vorhersagen bestätigen

Schließlich fasst Kapitel 8 alle Einzelbeträge zusammen und weist die Aussagen der These im Einzelnen nach. Die Arbeit schließt mit einem Ausblick auf zukünftige Entwicklungsmöglichkeiten.

1.5 Beiträge

Der zentrale Beitrag dieser Arbeit ist die Formulierung der

Theorie kooperativer und kollaborativer Prozessorganisation Mit rückgekoppelten Regelkreisen gelingt eine mathematisch geschlossene Beschreibung, die das Prozessverhalten unter unbekannten Einsatzbedingungen so organisiert, dass eine zeitnahe Informationsverarbeitung erfolgt. Die Prozesse werden dabei als *Black-Box* abstrahiert.

Die Theorie ist für eingebettete, ubiquitäre Rechnersysteme geeignet. Die vorliegende Arbeit liefert die folgenden weiteren Beiträge als Anwendung kooperativer und kollaborativer Organisationsmechanismen:

Neuartiges Verfahren zur Ablaufplanung periodischer Prozesse Mit Datenechtzeit wird ein neues Konzept zur Bewertung der Prozessausführung ubiquitärer Rechnersysteme eingeführt. Der Fair Quality Scheduler (FQS) realisiert damit in einer Kaskadenregelung eine kooperative und kollaborative Ablaufplanung zur Sicherung zeitnaher Prozessausführung für die Informationsverarbeitung. Das Betriebssystem Particle OS implementiert die Laufzeitumgebung für Prozesse und deren Regelung.

Fairer aperiodischer Server Es wird eine kooperative Ereignisverarbeitung vorgeschlagen, die eine faire, stochastische Garantie für beschränkte Buffer gibt, die Prozessornutzung begrenzt und Überlastsituationen vermeidet. Eine Online-Parameterschätzung erfasst die Ausführung aperiodischer Prozesse auf ubiquitären Systemen. Die Laufzeiteigenschaften können mittels eines gesteuerten M/D/k/k Warteschlangenmodells abgeleitet werden.

Koordination zwischen Echtzeit- und datengetriebenen Prozessen Buffer Feedback Scheduler (BFS) ist ein kollaboratives Organisationsverfahren, welches Echtzeit- und datengetriebene Nicht-Echtzeit-Prozesse gemeinschaftlich koordiniert. Der Datenfluss an den Schnittstellen der verschiedenen Prozesse wird optimiert.

Protokoll zur Zusammenarbeit verteilter Prozesse Feedback controlled unsynchronized periodic MAC (FCUP-MAC) koordiniert die Kommunikation ubiquitärer Rechnersysteme in einem Verbund mit kollaborativ reguliertem Nachrichtenaustausch. Mittels Kooperation werden lokale Energiebudgets egalisiert, um Netzwerkverbindungen maximal lange aufrecht zu erhalten.

Verfahren für ein zielorientiertes Energiemanagement Batteriebetriebene ubiquitäre Rechnersysteme implementieren mit PControl eine kooperative, Duty Cycle basierte Regelung entlang einer idealen Energienutzungsfunktion zum Erreichen einer vorgegebenen Betriebsdauer unter gewählter Ansprechbarkeitsgarantie.

2. Analyse

In ubiquitären Rechnerumgebungen interagieren eine Vielzahl von eingebetteten, miniaturisierten Sensorrechnersystemen miteinander und mit den Nutzern. Herausragendes Merkmal ist, dass die funktionserbringenden Rechnersysteme in solchen Umgebungen nicht vereinzelt wahrgenommen werden, sondern quasi für den Nutzer verschwinden. An ihre Stelle tritt eine rechnerinstrumentierte Umgebung. Darin werden Softwarekomponenten zur sensorischen Umgebungserfassung, Datenverarbeitung und Kommunikation zu Appliances kombiniert, um Dienste und Anwendungsfunktionalitäten zu erbringen.

Dieses komplexe Zusammenspiel unterliegt im praktischen Einsatz unbekannten und veränderlichen Einflussgrößen. Dazu zählen zum Beispiel die Anzahl umgebender Kommunikationssysteme, Laufzeiteigenschaften und das zu verarbeitende Datenaufkommen. Diese Ungewissheit gefährdet den entwurfsgerechten Ablauf der Appliances. Die rechnerinstrumentierte Umgebung zerfällt in der Wahrnehmung wieder in Einzelsysteme. Ziel dieses Kapitels ist, die Informationsverarbeitung ubiquitärer Rechnersysteme in solchen Umgebungen zu verstehen, um das Verhalten von Appliances zu kontrollieren.

Zu Beginn werden an ausgewählten Szenarien wichtige Prinzipien ubiquitärer Rechnersysteme dargestellt. Die beteiligten Akteure werden detailliert charakterisiert und ihre technische Umsetzung diskutiert. Ein erweiterter Appliance-Begriff führt drei grundlegende Softwarestrukturkomponenten ein und bestimmt durch die kombinierte Verarbeitung von Datenflüssen auf und zwischen den Rechnersystemen ein für die Arbeit verbindliches Verständnis der Informationsverarbeitung zur Erbringung von Anwendungsfunktionalitäten und Diensten. Als zentrale Problemstellung der Arbeit geht hervor, dass für die Kontrolle des Applianceverhaltens die vollständige Erfassung des komplexen Zusammenspiels der Komponenten nicht möglich ist. Der Lösungsansatz skizziert ein neuartiges Verfahren, welches (1.) mit Prozessen das zeitliche Ausführungsverhalten klassifiziert, (2.) das Prinzip der zeitnahen Informationsverarbeitung für den geordneten und nachvollziehbaren Betrieb ubiquitärer Rechnerumgebungen einführt, und (3.) Kooperation und Kollaboration als rückgekoppelte Organisationsmechanismen zur Realisierung zeitnaher Prozessausführung bestimmt. Damit sind die Aufgaben für den weiteren Verlauf der Arbeit festgelegt.

2.1 Szenarien

Ubiquitäre Anwendungsszenarien sind in der wissenschaftlichen Literatur sehr gut dokumentiert. Aus der Betrachtung ausgewählter Beispiele in Tabelle 2.1 werden die Grundprinzipien von Rechnersystemen abgeleitet. Im Anschluss werden charakteristische Zusammenhänge zwischen den Beteiligten in den Szenarien aufgezeigt.

2.1.1 Grundprinzipien

Die Szenarien der Tabelle 2.1 decken unterschiedliche Anwendungsbereiche ab und spiegeln sehr gut die folgenden Grundprinzipien ubiquitärer Rechnersysteme wider.

Embodied Virtuality [1] Ubiquitäre Anwendungen werden von rechnerinstrumentierten Umgebungen erbracht, d.h. Rechnersystemen, die *in die Umgebung und in alltägliche Objekte eingebettet* sind. Die Systeme sind dabei, vergleichbar mit einem Werkzeug, auf eine Funktion respektive Funktionsgruppe spezialisiert. Diese funktionelle Dekomposition unterscheidet ubiquitäre Rechnersysteme von Universalsystemen wie Desktop-PCs. Die Beispiele aus Tabelle 2.1 zeigen, dass oftmals viele Rechnersysteme in ubiquitären Umgebungen eingebettet sind und vielfältige Anwendungsfunktionalitäten und Dienste erbracht werden. Es ist eine hoch heterogene Applikationslandschaft.

Calm Technology [2] beschreibt eine Arbeitsweise von Rechnersystemen unauffällig im Hintergrund, in der peripheren Wahrnehmung des Nutzers. Der Wechsel in den Vordergrund, d.h. in die bewusste Wahrnehmung des Nutzers, erfolgt dann, wenn eine Anwendungsfunktion, die vom Rechnersystem erbracht wird, auch genutzt wird. Umgeben von vielen Geräten in einer ubiquitären Rechnerumgebung, kann der Nutzer mit dem Calm Technology Prinzip die *Rechnersysteme praktikabel nutzen.*

Unremarkable Computing [11] fordert, dass ubiquitäre Rechnersysteme unsichtbar hinsichtlich der Funktion sein sollen. Dieses „Invisibility-in-use“ ist vergleichbar mit dem Ausüben täglicher Routinen. Anwendungen sollen so entworfen werden, dass sie die beabsichtigten Aufgaben unterstützen. Unremarkable Computing ist kein Prinzip der Schnittstellengestaltung oder der Vermittlung von Funktionen durch Objekte. Vielmehr sollen Rechnersysteme *nicht mehr als Einzelsysteme* wahrnehmbar sein, sondern nur noch die Anwendungsfunktionalität oder Dienste.

2.1.2 Beteiligte in ubiquitären Anwendungsszenarien

Aus den vorangegangenen Betrachtungen wird die in Abbildung 2.1 dargestellte Hierarchie aus Umgebung, Appliance und Plattform entwickelt. In ubiquitären Anwendungsszenarien erbringen Alltagsumgebungen elektronisch gestützte Funktionalitäten. Diese Anwendungen und Dienste werden unter dem Begriff Appliances zusammengeführt. Ubiquitäre Umgebungen enthalten oftmals mehrere Appliances. Die Ausführungsplattform von Appliances sind Rechnersysteme, die so in die Umgebung eingebettet sind, dass sie für den Nutzer nicht mehr als Einzelsysteme von der Umgebung zu unterscheiden sind. Somit erscheinen auch die Anwendungsfunktionalitäten und Dienste als von der Umgebung erbracht. Die Hierarchie aus Abbildung 2.1 soll in den folgenden Abschnitten näher charakterisiert werden.

Szenario	Umgebung	Anwendungen	Rechnersysteme
ParcTab [3]	Büroumgebung	lokationsbasierte Benachrichtigungs-, Kalender-, E-Mail-funktionen, Gerätekontrolle	persönliche, tragbare Rechnersysteme, die mit Rechnern der Umgebung kommunizieren.
AwareOffice [4]	Büroumgebung	automatisiertes Meetingraummanagement und Meetingannotation, Gerätekontrolle	in mobile und feste Bürogegenstände eingebettete Miniatursensorsysteme. Kommunikation untereinander und mit Rechnern der Infrastruktur.
AwareHome [5]	häusliche Umgebung	unauffällige Beaufsichtigung alternder Personen im Privathaushalt, Präsenzdienste zur Erhaltung familärer Beziehungen über große räumliche Entfernungen	eingebettete Miniatursensor-, Kamera-, Ortungs- und Displaysysteme.
Remembrance Camera [10]	persönliche Alltagsumgebung wie Büro und häusliche Umgebung	Gedächtnisunterstützung durch kontextsensitive Erfassung wichtiger Ereignisse im Tagesablauf	persönliche, tragbare Sensorkamera
CoBIs [6]	industrielle Produktionsumgebung	Prüfen der Einhaltung verschiedener Lagerungsregelungen von Chemikalien (Umgebungsbedingungen, Ort, Lagermenge, Lagerungskombinationen)	eingebettete Miniatursensorsysteme, die infrastrukturlos miteinander die Anwendungsfunktionen erbringen.
eSeal [7]	Warentransport	Detektion verschiedener Integritätsverletzungen von Waren und Warenverbünden	eingebettete Miniatursensorsysteme, die infrastrukturlos miteinander die Anwendungsfunktionen erbringen.

Tabelle 2.1: Untersuchte Beispiele ubiquitärer Anwendungsszenarien

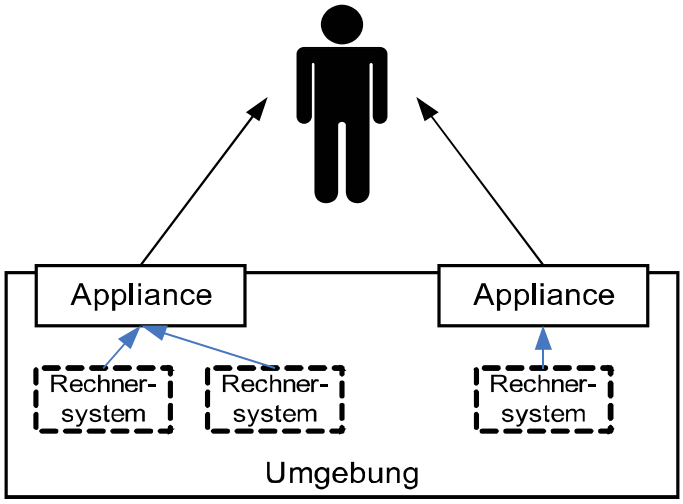


Abbildung 2.1: Hierarchieebenen ubiquitärer Anwendungsszenarien. Der Nutzer bezieht Anwendungsfunktionalitäten und Dienste der Appliances aus der für ihn nicht weiter differenzierbaren Umgebung. Insbesondere können die funktionserbringenden Rechnersysteme aus dieser Perspektive nicht vereinzelt werden.

2.2 Charakterisierung ubiquitärer Rechnerumgebungen

Aus der Szenariendarstellung im vorangegangenen Abschnitt wird eine Charakterisierung ubiquitärer Rechnerumgebungen hergeleitet. Topologische Strukturen, Software und Hardware bilden die Schwerpunkte in den Betrachtungen der folgenden Abschnitte.

2.2.1 Aufbau und Einbettung

Gemeinsames Merkmal der Szenarien aus Abschnitt 2.1 ist die Nutzung eines Verbundes von in die Umgebung eingebetteten Rechnersystemen. Es existiert keine exklusive Beziehung zwischen Nutzer und Gerät, sondern der Nutzer interagiert mit einer rechnerinstrumentierten Umgebung. Das ist aber kein Schnittstellenproblem. Vielmehr müssen die Rechnersysteme die vielen möglichen Zusammenhänge in solchen Umgebungen berücksichtigen. Ubiquitäre Rechnersysteme sind auf zwei Arten eingebettet, die in Tabelle 2.2 erläutert werden. Insbesondere die kognitive Einbettung erfordert, dass ubiquitäre Rechnersysteme nicht mehr als diskrete Zusatzgeräte wahrgenommen werden. Die Verschmelzung der Abläufe auf dem Rechnersystem mit den Abläufen in der Umgebung erfordert, dass ubiquitäre Rechnersysteme die Umgebung erfassen und deuten können. Wichtige Elemente dabei sind Sensoren, die physikalische Umgebungsbedingungen erfassen, und eine anschließende Datenverarbeitung. Umgebung und Nutzerinteraktionen werden erkannt und im Anwendungsablauf berücksichtigt. Die *sensorische Erfassung und Deutung der Umgebung* ist die Schlüsselfunktionalität für die Einbettung ubiquitärer Rechnersysteme.

Darüber hinaus sind die Rechnersysteme auch miteinander verbunden. Diese Kommunikationsbeziehungen sind nicht fest organisiert, sondern entstehen bedarfsorientiert. Die Mobilität der Nutzer einschließlich mitgeführter Einbettungsgegenstände

Kognitive Einbettung	Physikalische Einbettung
• Einbindung in den Alltag des Nutzers ist so stark, dass Rechnersysteme nicht mehr als Geräte wahrgenommen werden. • Wahrnehmung über Funktion/Aufgabe • Einbettung endet, wenn eigenzweckgebundene Aufmerksamkeit benötigt wird, zum Beispiel Reparatur, Wartung	• Integration in alltägliche Gegenstände • Rechnersysteme sind Sekundärartefakte und behindern nicht die reguläre Funktion und Nutzung des Gegenstandes • neue, sogenannte „smarte“ Gegenstände entstehen

Tabelle 2.2: Ubiquitäre Rechnersysteme sind auf zwei Arten eingebettet

lässt die Anzahl kommunizierender Rechnersysteme variieren. In ubiquitären Rechnerumgebungen herrschen *Ad-hoc-Netzwerke* vor. Weitere strukturierende Maßnahmen, zum Beispiel ausgezeichnete Systeme mit zentralen Koordinationsrollen, sind im allgemeinen Fall nicht vorgesehen.

Ubiquitäre Rechnersysteme sind offene Systeme. Sie sind mit der Umwelt und anderen eingebetteten Rechnersystemen derart verbunden, dass sie sensorische Informationen aufnehmen, verarbeiten und mittels Kommunikation und weiterer Aktuatorik wieder an die Umwelt abgeben. Jedes Rechnersystem operiert in einem *Kontext*, der die lokale Umgebung des Rechnersystems repräsentiert. Die lokale Umgebung umfasst innere Betriebszustände, die äußere sensorisch erfassbare Umgebungssituation wie auch alle Kommunikationsbeziehungen. Kurz: die Menge der Informationen, die für die Erbringung von Anwendungsfunktionalitäten und Diensten von Bedeutung sind. Mittels Datenverarbeitung, Kommunikation und Aktuatorik kann der Kontext vom Rechnersystem auch gestaltet werden. Im allgemeinen Fall kann keine metrische Ausdehnung des Kontextes angegeben werden.

Der Betrieb solcher Umgebungen zeichnet sich durch eine autonome Arbeitsweise eingebetteter, ubiquitärer Rechnersysteme mit keinem oder nur geringem Administrationsaufwand aus. Damit bleibt die Einbettung maximal lange erhalten.

2.2.2 Appliances

Don Norman beschrieb in [12] drei Axiome als Kennzeichen einer Information Appliance. Demnach sollen Information Appliances den Kriterien

1. Einfachheit
2. Vielseitigkeit
3. Vergnüglichkeit

genügen. Diese designzentrische Definition wird in dieser Arbeit durch eine auf ubiquitäre Rechnersysteme bezogene, funktionsbasierte Definition erweitert.

Definition 2.1 (Appliances auf ubiquitären Rechnersystemen)**.** *Appliances werden auf ubiquitären Rechnersystemen ausgeführt und erbringen spezifische Anwendungsfunktionalitäten und Dienste in rechnerinstrumentierten Umgebungen. Sie umfassen folgende Basisfunktionalitäten:*

- *unmittelbare sensorische Erfassung der Umwelt*
- *lokale Informationsverarbeitung zur Erkennung des eigenen Einbettungszustandes (= Kontext), Entscheidungsfindung und Reaktion*
- *Kommunikation mit anderen Rechnersystemen*
- *Veränderung der Umwelt durch Aktuatorik*

Die Festlegung der Basisfunktionalitäten in Definition 2.1 ergeben sich aus der Szenarienbetrachtung in Abschnitt 2.1 und dem Verständnis der kognitiven Einbettung in Tabelle 2.2.

Als eine direkte Schlussfolgerung aus Definition 2.1 ergibt sich, dass Appliances Softwarerealisierungen darstellen. Appliances werden durch Software angetrieben. Die Basisfunktionalitäten lassen sich auf Softwarestrukturkomponenten ableiten. Sie repräsentieren Klassen, denen sich ausprogrammierte Softwarefunktionen, d.h. Code, zuordnen lassen. Im weiteren Verlauf des Kapitels dienen die Strukturkomponenten dazu, die Arbeitsweise ubiquitärer Rechnersysteme einheitlich zu erfassen. Tabelle 2.3 beschreibt die drei Strukturkomponenten *Sens* für die sensorische Erfassung, *Comp* für die Datenverarbeitung und *Comm* für die Netzwerkkommunikation. Eine spezielle Kommunikationskomponente unter dieser Betrachtung sind Speicher. Sender und Empfänger sind hierbei sehr eng verknüpft. Speicher sind daher transparent in *Comm* zusammengefasst.

Die Eingangs- und Ausgangsdefinitionen der Softwarestrukturkomponenten beschreiben Verknüpfungsmöglichkeiten beim Übergang zwischen den Strukturkomponenten. Es wird zwischen internen, d.h. nur auf dem Rechnersystem existierend, und externen Verknüpfungen zwischen Rechnersystemen unterschieden.

intern Verknüpfung zwischen Strukturkomponenten auf *ein und demselben* Rechnersystem. Beispiel: $Sens \longrightarrow Comp$ oder $Comp \longrightarrow Comm$.

extern Verknüpfung *zwischen den Rechnersystemen* über die Kommunikation. Beispiel: $Comm \longrightarrow Comm$. Das Dispatching zu weiteren Komponenten auf entfernten Rechnersystemen ist transparent.

Beide Verknüpfungsarten sind datenzentrisch, d.h. die Komponenten werden durch die erzeugten Daten miteinander zu Appliances komponiert. Bei externen Verknüpfungen erfolgt eine transparente Umwandlung in ein physikalisches Signal und wieder zurück, um die gesendeten Daten zurückzugewinnen. Mit dieser Art der datenflussorientierten Kopplung wird ein Höchstmaß an Flexibilität erreicht. Eine Appliance kann verteilt durch mehrere Rechnersysteme realisiert werden. Jedoch sind die Softwarestrukturkomponenten auf den Ort der Rechnersysteme beschränkt.

Bezeichner	**Komponente**	**Funktionalität**
Sens	Sensorische Erfassung	Sensoren erschließen physikalische Phänomene für das Rechnersystem. Sie liefern Informationen über den Umgebungskontext. *Eingang:* physikalische Signale der Umwelt *Ausgang:* (aufbereitete) Daten
Comp	Verarbeitung (Aktuatorik)	Es werden Algorithmen für die Erkennung des Einbettungszustandes (= Kontext), sowie entsprechende Entscheidungen und Systemreaktionen zur Anwendungsrealisierung und Diensterbringung ausgeführt. *Eingang:* Daten *Ausgang:* Daten Steuern die Verarbeitungsroutinen Aktuatoren an, so erfolgt die Umwandlung in ein physikalisches Signal. Das Signal verlässt das Rechnersystem. Die Verarbeitung ist abgeschlossen.
Comm	Kommunikation	Wird ein von einem Rechnersystem generiertes physikalisches Signal durch dasselbe oder ein weiteres Rechnersystem aufgenommen, so entsteht eine Kommunikation. Es handelt sich sowohl um einen Sensor (Empfänger) als auch um einen Aktuator (Sender). Entscheidend ist, dass die gesendeten Daten zurückgewonnen werden können. *Eingang(Sender):* Daten *Ausgang (Sender):* physikalisches Signal *Eingang (Empfänger):* physikalisches Signal *Ausgang (Empfänger):* Daten

Tabelle 2.3: Softwarestrukturkomponenten der Basisfunktionalitäten von Appliances

Zusammenfassend ergibt sich folgendes Gesamtbild einer Appliance auf ubiquitären Rechnersystemen.

- Basisfunktionalitäten sind durch Definition 2.1 festgelegt.
- Realisierung der Funktionalitäten erfolgt als Softwarestrukturkomponenten *Sens*, *Comp*, *Comm* gemäß Tabelle 2.3
- Softwarestrukturkomponenten sind über ihre erzeugten Daten miteinander verknüpft.

Die obige Betrachtung von Appliances als Komposition von Strukturkomponenten legt somit den Fokus auf die datenzentrische Arbeitsweise ubiquitärer Rechnersysteme für diese Arbeit. Appliances entstehen durch eine kombinierte Verarbeitung von Datenflüssen.

2.2.3 Plattformen für die technische Realisierung

Plattformen für die Realisierung ubiquitärer Rechnerumgebungen müssen die Basisfunktionalitäten von Appliances aus Definition 2.1 technisch umsetzen. Dies gelingt mit miniaturisierten Rechnersystemen, die

- frei programmierbar sind,
- Sensoren für die Kontexterfassung und Aktuatoren für die Umweltbeeinflussung integrieren und
- mit anderen in der Umgebung eingebetteten Rechnersystemen Informationen austauschen können.

Die Miniaturisierung erlaubt vor allem eine besonders leichte Umsetzung der kognitiven und physikalischen Einbettung. Unterstützt wird dieser Aspekt von drahtlosen Kommunikationsverbindungen. Eine Funkkommunikation ermöglicht Rechnerverbünde ohne direkte Sichtverbindung und erlaubt eine tiefe physikalische Einbettung.

Im Forschungsbereich drahtlose Sensornetzwerke wurden Plattformen wie beispielsweise Motes [13][14], BTNodes [15], ScatterWeb [16], oder Telos [17] geschaffen, die diese Anforderungen erfüllen. Plattformen, die im Umfeld des Forschungsbereichs Ubiquitous und Pervasive Computing entstanden, sind zum Beispiel Smart-Its [18], Particle Computer [19][20] oder SoapBox [21]. Miniaturisierte eingebettete Systeme wie EtherNut [22], Gumstix [23] oder das Plug&Lug System [24] wurden ebenfalls für die Realisierung von Appliances eingesetzt.

Der Aufbau dekomponiert sich in gemeinsame zentrale Bausteine wie Prozessor, Sensorik, Aktuatorik, Kommunikation und Energieversorgung. Bei allen Plattformentwürfen wurde zudem auf die *modulare Integration multipler Sensoren* geachtet. Als typische Sensoren für Appliances wurden in [25] Bewegungssensoren, Lichtsensoren, Temperatursensoren und Sensoren zur Messung einwirkender mechanischer Kräfte identifiziert. Diese werden oft in Kombination miteinander auf einer Plattform genutzt, um ein breites Spektrum an Kontextinformation zu erfassen.

Jede der erwähnten Plattformen hat spezifische technische Merkmale, die sich in gemeinsamen typischen Bereichen bewegen. Tabelle 2.4 beschreibt diese Bereiche und untergliedert sie in die fünf Plattformbausteine Prozessor, Speicher, Kommunikation, Sensorik / Aktuatorik und Energieversorgung. Aus den Einzelplattformen entsteht eine technische Spezifikation für die in dieser Arbeit betrachteten ubiquitären Rechnersysteme. Diese Arbeit betrachtet als Zielplattform Rechnersysteme mit

Eigenschaft	**Ausprägung**
Prozessor	Verwendung von Mikrocontrollern mit Taktraten zwischen 1 MHz und 40 MHz. Integriert sind Speicher und Peripheriebussysteme.
Speicher	Programmspeicher ROM: 128 KiloBytes, Variablenspeicher RAM: bis zu 10 KiloBytes. Zusätzlich ist weiterer Speicher in Form von speziellen Bausteinen (ca. 512 KiloBytes) oder Speicherkarten beigestellt.
Kommunikation	Drahtlose Funkkommunikationsschnittstellen und skalierbare Adhoc-Kommunikationsprotokolle werden eingesetzt. Im allgemeinen Fall gibt es keinen zentralen Koordinator.
Sensorik und Aktuatorik	Multiple Sensoren und Aktuatoren sind modular hinzufügbar und werden in Kombination eingesetzt [25]. Oftmals direkt mit dem Prozessor verbunden.
Energieversorgung	Rechnersysteme sind oftmals in mobile Objekte eingebettet. Der Betrieb mit Batterien ist daher vorherrschend.

Tabelle 2.4: Technische Spezifikation von Plattformen für die in dieser Arbeit betrachteten ubiquitären Rechnersysteme.

vergleichbaren Leistungsparametern wie in Tabelle 2.4 dargestellt. Als Plattform für die Beiträge dieser Arbeit sowie für die experimentellen Untersuchungen wird die in Abbildung 2.2 dargestellte Particle Computer Plattform [19] vorgeschlagen.

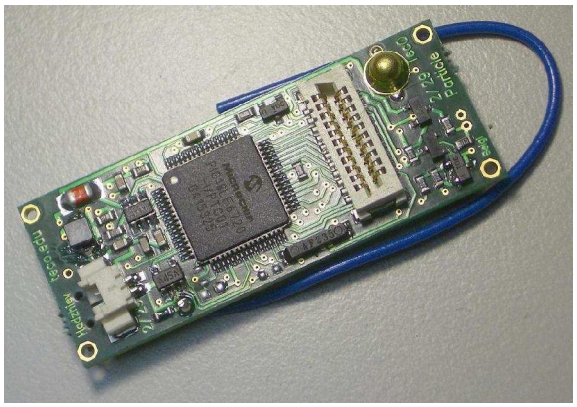


Abbildung 2.2: Die Particle Computer Plattform ist das ubiquitäre Rechnersystem für die Untersuchungen in dieser Arbeit.

2.2.4 Zusammenfassende Charakterisierung

Als Resultat der vorangegangen Abschnitte kann die zu Anfang in Abschnitt 2.1 entwickelte Hierarchie innerhalb ubiquitärer Anwendungsszenarien technisch wie folgt konkretisiert werden.

Umgebung: Rechnersysteme sind in die alltägliche Umgebung kognitiv und physikalisch eingebettet. Sie sind eng an den Kontext der Umwelt einschließlich des Nutzers gebunden. Kommunikationsbeziehungen sind nicht fest organisiert, sondern entstehen bedarfsorientiert.

Anwendungsfunktionalität: Appliances entstehen durch eine kombinierte Verarbeitung von Datenflüssen auf und zwischen eingebetteten Rechnersystemen. Ubiquitäre Umgebungen enthalten oftmals mehrere Appliances.

Realisierung: Es werden ressourcenbeschränkte, drahtlos kommunizierende, batteriebetriebene Miniatursensorsysteme eingesetzt, die oftmals mehrere Sensoren umfassen und nutzen können. Skalierbarkeit und Adhoc-Fähigkeit sind typische Merkmale von Kommunikationsverbindungen.

2.3 Problembeschreibung

Datenzentrische Appliances hängen während ihrer Ausführung stark vom zu verarbeitenden Datenaufkommen ab. Die Einbettung in die Umgebung erschwert es, zur Entwicklungszeit zuverlässige Prognosen darüber abzugeben. Der unbekannte Einfluss der Umgebung auf das Datenaufkommen ändert das Verhalten der Appliances, zum Beispiel hinsichtlich des Energieaufwandes, Ansprechbarkeit, Laufzeit und Kommunikationsverhalten. Diese Ungewissheit gefährdet den entwurfsgerechten Ablauf von Anwendungen und Diensten. Für einen Nutzer in der ubiquitären Rechnerumgebung erscheint das Applianceverhalten ungeordnet, nicht systematisch und unkontrollierbar. Die ubiquitäre Rechnerumgebung zerfällt in der Nutzerwahrnehmung wieder in Einzelsysteme.

2.3.1 Organisation der Datenverarbeitung

Durch Eingriff in den Ablauf der Datenverarbeitung können Appliances den unbekannten Einsatzbedingungen angepasst werden. Dies wird in folgendem Fallbeispiel bereits auf der Ebene der Strukturkomponenten von Appliances gezeigt.

Fallbeispiel. In mit der Umwelt und Kontext verbundenen ubiquitären Rechnersystemen beeinflussen sensorisch erfasste Informationen die anschließende Datenverarbeitung. Aus der exemplarischen Abfolge von Instanzen der Strukturkomponenten

$$Sens_1 \longrightarrow Comp_1 \longrightarrow Sens_2 \longrightarrow Comp_2 \longrightarrow Comm$$

kann die neue Folge

$$Sens_1 \longrightarrow Comp_1 \longrightarrow Comm \longrightarrow Sens_2 \longrightarrow (Comp_2 = 0)$$

gebildet werden, wenn die Auswertung des zweiten Sensors ergibt, dass er keinen Beitrag leistet. Die Ergebnisse der ersten Sensorauswertung werden früher kommuniziert. Außerdem wird der zweite Sensorwert nicht mehr übermittelt. Laufzeitverhalten und produziertes Datenaufkommen des Rechnersystems ändern sich. Die Umstellung ist sinnvoll, da die Appliance an Ansprechbarkeit gewinnt. Außerdem ergeben sich technische Verbesserungen, da weniger Energie für die Übermittlung aufgewendet werden muss.

Ubiquitäre Umgebungen enthalten oftmals mehrere Appliances, so dass Rechnersysteme mehrere und unterschiedliche Funktionalitäten durch die Softwarestrukturkomponenten erbringen. Appliances können auch opportunistisch entstehen [26][27], d.h. erst durch die Feststellung, dass bestimmte Funktionalitäten in der aktuellen Ausprägung der ubiquitären Rechnerumgebung existieren. Es liegt keine homogene Appliancelandschaft vor, sondern eine Umgebung mit vielen Beteiligten und verschiedene Aufgaben.

2.3.2 Komplexität der Organisation

Die Instanzen s der Strukturkomponenten lassen sich in $s!$ Abfolgen kombinieren. Nicht jede Kombination ist sinnvoll, jedoch kann die ernorme Anzahl nur schwer manuell verwaltet werden. Eine geeignete Abfolge, die dem aktuellen Ausführungskontext angepasst ist, muss daher durch einen Auswahlprozess erfolgen. Um das Zusammenspiel der Komponenten zu erfassen, müssen die Ausführungsparameter aller an der Datenverarbeitung beteiligten Strukturkomponenten auf allen Rechnersystemen in Erfahrung gebracht werden. Parameter sind beispielsweise Laufzeiten, der Energieaufwand und Kommunikationsverbindungen.

Diese Informationen werden von jedem Rechnersystem hinsichtlich seiner lokalen Datenverarbeitung bewertet. Daher muss jedes Rechnersystem die vollständige Information erfassen. Es ist keine Reduktion der Information durch ein Zwischensystem möglich, welches für alle angeschlossenen Systeme eine zusammenfassende Information bereitstellen könnte. Es ist somit kein Routingproblem.

Im allgemeinen Fall arbeiten ubiquitäre Rechnerumgebungen ohne zentralen Koordinator. Der notwendige Speicherbedarf *jedes* Rechnersystems ist $O(m \cdot n^2)$, wobei n die Anzahl der Rechnersysteme und m die Anzahl der Ausführungsparameter repräsentiert. Stark ressourcenbeschränkte Zielplattformen aus Abschnitt 2.2.3 können quadratische Aufwände nicht leisten. Bei geringer Anzahl an Rechnersystemen ist bereits ein enormer Speicheraufwand notwendig. Der Aufwand, diese Informationen in Umgebungen ohne feste Kommunikationsbeziehungen einzusammeln, ist ebenfalls nicht leistbar. In größeren Netzwerken, denen Rechnersysteme kurzfristig beitreten oder sie auch wieder verlassen, ist es zudem schwierig, die Aktualität der Informationen sicherzustellen. Über das Einsammeln von Laufzeitinformationen lässt sich das Zusammenspiel der Komponenten nicht praktikabel erfassen.

Es sei an dieser Stelle betont, dass diese Komplexität ein *strukturinhärentes Problem* ist, dessen Ursache die kombinierte Verarbeitung von Datenflüssen in Appliances ist. Sie ist in ubiquitären Rechnerumgebungen verankert und keine Folge spezifischer Implementierungen. Es stellt sich somit das zentrale Problem dieser Arbeit wie folgt dar:

Problem 2.1. *Das komplexe Zusammenspiel der Komponenten von Appliances kann in ubiquitären Rechnerumgebungen praktikabel nicht vollständig erfasst werden. Es ist ein Verfahren zu finden, so dass das Verhalten von Appliances in unbekannten und sich zeitlich ändernden Einsatzumgebungen entwurfsgerecht kontrollierbar bleibt.*

Die Problembeschreibung ist eng mit den Eigenschaften ressourcenbeschränkter Rechnersysteme verbunden. Kontrollierbarkeit bedeutet, dass die Datenverarbeitung den verfügbaren Ressourcen, wie Rechenzeit, Speicher- und Energievorrat, anzupassen ist. Im Zusammenhang mit der oftmals autonomen Arbeitsweise und dem geforderten geringen Administrationsaufwand, ist Kontrollierbarkeit eine Schlüsseleigenschaft für den geordneten und nachvollziehbaren Betrieb ubiquitärer Rechnerumgebungen.

2.4 Lösungsansatz

Für den erfolgreichen Betrieb ubiquitärer Rechnerumgebungen muss die Ausführung von Appliances betrachtet werden. Der Lösungsansatz unterteilt sich in drei Abschnitte, die in den folgenden Unterabschnitten erläutert werden.

1. *Datenzentrische Appliances sind in eine ausführbare Datenstruktur zu überführen.* Dazu werden Prozessklassen definiert, die unterschiedliches zeitliches Ausführungsverhalten widerspiegeln.

2. *Ein einheitliches Arbeitsprinzip ist festzulegen, welches einen geordneten und nachvollziehbaren Betrieb ubiquitärer Rechnerumgebungen abbildet.* Für datenzentrische Appliances ist die zeitnahe Verarbeitung von Informationen wesentlich.

3. *Ein Organisationsmechanismus wird benötigt, der das Arbeitsprinzip in ubiquitären Rechnerumgebungen durchsetzt.* Prozesse kooperieren und kollaborieren, um in unbekannten Einsatzumgebungen eine zeitnahe Datenverarbeitung zu gewährleisten.

Mit der Umsetzung des dritten Punkts wird die zentrale Problembeschreibung dieser Arbeit aus dem vorangegangenen Abschnitt aufgegriffen.

2.4.1 Prozesse und Prozessklassen

Die Reihenfolge der Ausführung von Funktionalitäten der Softwarestrukturkomponenten *Sens*, *Comp* und *Comm* werden in Prozessen, vergleichbar mit einer Liste von zu erbringenden Teilaufgaben, festgelegt. Die durch den Prozessablauf erbrachte Funktionalität wird Prozessaufgabe genannt. Dabei kann es durchaus zu zeitlich überlappenden Ausführungen durch den Parallelbetrieb mehrerer Rechnersysteme kommen. Prozessklassen kategorisieren schließlich in welchem zeitlichen Zusammenhang Prozesse ausgeführt werden. Darunter fallen zum Beispiel Startzeitpunkt und Ausführungsdauer. Die folgenden Klassen beschreiben die Struktur des Ausführungskontextes von Prozessen.

Periodische Prozesse: Periodische Prozesse werden in regelmäßigen Zeitabständen (Periodendauer) ausgeführt. Sie sind typisch für repetitive Aufgaben, zum Beispiel das Auslesen von Sensorwerten.

Aperiodische Prozesse: Daten von unregelmäßig auftretenden Ereignisse werden von aperiodischen Prozesse verarbeitet. Ein Startzeitpunkt kann nicht festgelegt werden.

Echtzeitprozesse: Sind feste Zeitschranken, nach denen die Prozessausführung abgeschlossen sein muss, gegeben, so wird von Echtzeitprozessen gesprochen. In dieser Arbeit wird die Periodendauer als Zeitschranke von Echtzeitprozessen angenommen.

Verteilte Prozesse: Der Austausch von Daten zwischen Rechnersystemen fällt in die Verantwortung von verteilten Prozessen. Erreichbarkeit von Kommunikationspartnern sowie die Funktionen zur Aufrechterhaltung des Netzwerkes müssen zeitlich abgestimmt werden.

Die Probleme der Ressourcennutzung ubiquitärer Rechnersysteme in unbekannten Einsatzumgebungen können nun in Bezug zum zeitlichen Zusammenhang der Prozessausführung konkretisiert werden. Die Probleme können zum Beispiel sein:

- Unbekanntes und zeitlich veränderliches Ausführungsverhalten der Prozesse
- Vorhersagen, zum Beispiel wann und welche Ereignisse eintreten, sind schwer zu treffen.
- Zeitliche Koordination von Kommunikationsaktivitäten über mehrere Rechnersysteme hinweg ist schwierig.
- Überlastung mit der möglichen Folge des permanenten Auslassens, sogenanntes Verhungern, von Prozessen kann auftreten.
- Best-effort Ressourcennutzung ohne Berücksichtigung oder Kenntnis des vollständigen Prozesskontextes.
- Unbekannte und nicht im Voraus quantifizierbare Wirkung auf den Energievorrat bei Prozessausführung.

Die Forderung nach Kontrollierbarkeit der Ressourcennutzung wird unter solchen Bedingungen direkt einsichtlich. Aufgabe dieser Arbeit ist es, Prozesse aller Klassen datenzentriert zu modellieren. Prozesse sollen über die von ihnen generierten Daten miteinander zu Appliances komponiert werden können.

2.4.2 Zeitnähe-Prinzip

Die Ressourcennutzung ist an die Prozessausführung gebunden. Für einen geordneten und nachvollziehbaren Betrieb ubiquitärer Rechnerumgebungen ist die Prozessausführung zu ordnen. Für datenzentrische Appliances wird das folgende Prinzip der zeitnahen Datenverarbeitung wie folgt eingeführt.

Definition 2.2 (Prinzip der zeitnahen Datenverarbeitung)**.** *Die Datenverarbeitung verfügbarer Daten soll so früh wie möglich beginnen.*

Diese Definition *impliziert nicht*, die Zeitspanne zwischen Datenverfügbarkeit und Datenverarbeitung zu minimieren. Liegen zwei Daten für jeweils einen Verarbeitungsprozess vor, so wird der Prozess zuerst gestartet, der das jüngste Datum verarbeitet. Die zeitnahe Datenverarbeitung realisiert eine Last-In-First-Out (LIFO) Verarbeitung. Sie repräsentiert eine Ordnungsrelation für die Erbringung datenzentrischer Appliances auf ubiquitären Rechnersystemen.

Zeitnahe Datenverarbeitung gestaltet die Arbeitsweise von Rechnersystemen so, dass immer aktuelle Informationen verarbeitet werden. Die Kopplung zwischen Umgebung und von den Rechnersystemen erbrachten Appliances wird somit enger. Im Betrieb von Appliances wird der aktuelle Kontext und die Interaktionen des Nutzers mit der Umgebung priorisiert. Dies erhöht die Nachvollziehbarkeit durch den Nutzer und verbessert die kognitive Einbettung.

Aufgabe dieser Arbeit ist es, die konkrete Ausprägung zeitnaher Datenverarbeitung im Zusammenhang mit den jeweiligen Prozessklassen zu spezifizieren.

2.4.3 Kooperation und Kollaboration

Das Zeitnähe-Prinzip formuliert eine Auswahl, nach der ubiquitäre Rechnersysteme gewonnene und vorliegende Daten verarbeiten. Die entstehende Ordnungsrelation kann durch Messung des zeitlichen Ausführungsverhaltens beobachtet werden. Kontrollierbarkeit entsteht, wenn diese Laufzeitinformationen ausgewertet und in einen Stellmechanismus *zurückgeführt* werden. Rückkopplung ist das Instrument zur zielgerichteten Organisation des Ausführungsverhaltens hinsichtlich des Zeitnähe-Prinzips und damit zur Anpassung der Ressourcennutzung.

Es können zwei Arten von Prozessverhalten unterschieden werden: (1) aufeinander wirkende Prozesse, (2) miteinander interagierende Prozesse. Kooperation ist der Mechanismus zur Organisation von aufeinander wirkenden Prozessen, Kollaboration ist der Mechanismus zur Organisation von miteinander interagierenden Prozessen. Durch Rückkopplung entstehen Regelkreissysteme, die Wirkung und Interaktion so einstellen, dass zeitnahe Datenverarbeitung erreicht wird. Laufzeitinformationen von Prozessen werden an Regler zurückgekoppelt. Sie stellen das Systemverhalten entsprechend den sich verändernden Ausführungsbedingungen nach.

Mit der Regelkreistheorie existiert eine mathematische Modellbildung, die verwendet werden kann, um eine leistungsfähige Prozessorganisation ubiquitärer Rechnersysteme zu beschreiben und überprüfbare Vorhersagen zu gewinnen. Kooperation und Kollaboration sind Organisationsmechanismen, die die zeitnahe Datenverarbeitung und kontrollierbare Ressourcennutzung in unbekannten Einsatzumgebungen der Rechnersysteme realisieren.

Aufgabe dieser Arbeit ist es, eine auf alle Prozessklassen anwendbare Theorie der kooperativen und kollaborativen Prozessorganisation zu formulieren. In Abhängigkeit von konkreten Fragestellungen innerhalb der jeweiligen Prozessklassen sind Rückkopplungsinformationen zu bestimmen, Stellgrößen zu identifizieren und Regelkreissysteme aufzustellen.

2.5 Zusammenfassung

In ubiquitären Anwendungsszenarien, wie in den Beispielen aus Abschnitt 2.1 erbringen Alltagsumgebungen elektronisch gestützte Funktionalitäten. Diese Anwendungen und Dienste werden unter dem Begriff Appliances zusammengeführt. Die Ausführungsplattform von Appliances sind Rechnersysteme, die so in die Umgebung eingebettet sind, dass sie für den Nutzer nicht mehr als Einzelsysteme von der Umgebung zu unterscheiden sind. Somit erscheinen auch die Anwendungsfunktionalitäten und Dienste als von der Umgebung erbracht.

Appliances wurden in Abschnitt 2.2 detailliert analysiert und charakterisiert. Ein erweiterter Appliancebegriff verwendet drei grundlegende Softwarestrukturkomponenten *Sens*, *Comp*, *Comm* und deren Verknüpfungsmöglichkeiten, um ein einheitliches Verständnis der Informationsverarbeitung für eine Implementierung auf eingebetteten, ubiquitären Rechnersystemen zu schaffen. Appliances sind demzufolge datenzentrisch organisiert, d.h. sie entstehen durch eine kombinierte Verarbeitung von Datenflüssen auf und zwischen eingebetteten Rechnersystemen. Ubiquitäre Rechnerumgebungen enthalten viele Beteiligte und realisieren viele verschiedene Aufgaben. In diesem komplexen Zusammenspiel gefährden unbekannte Einsatzbedingungen den entwurfsgerechten Ablauf der Appliances. Die Umgebung zerfällt in der Wahrnehmung wieder in Einzelsysteme. Kontrollierbarkeit ist daher eine Schlüsseleigenschaft für den geordneten und nachvollziehbaren Betrieb ubiquitärer Rechnerumgebungen.

Die Analyse formuliert in Abschnitt 2.3 das zentrale Problem: Komplexe und verteilte ubiquitäre Rechnerumgebungen sollen ohne die vollständige Erfassung der Ausführungsparameter aller Beteiligten kontrollierbar bleiben. Damit verbunden ist die effiziente Nutzung von Rechenzeit, Speicherkapazitäten und Energievorräten auf den ressourcenbeschränkten ubiquitären Rechnersystemen. Abschnitt 2.4 skizziert den Lösungsansatz. Es werden Kollaboration und Kooperation als Prozessorganisationsmechanismen bestimmt, die unter Verwendung rückgekoppelter Laufzeitinformationen die Datenverarbeitung zeitnah gestalten.

Ein erfolgreicher Ansatz muss für die Prozesse der spezifizierten Klassen eine zeitnahe Datenverarbeitung realisieren. Es ist daher eine Theorie der kooperativen und kollaborativen Prozessorganisation zu formulieren und deren erfolgreiche Anwendung auf konkrete Probleme der Gestaltung von Appliances mit ressourcenbeschränkten ubiquitären Rechnersystemen zu zeigen. Damit sind die Aufgaben für den weiteren Verlauf der Arbeit festgelegt. Als typische Plattform ubiquitärer Rechnersysteme und für die experimentellen Untersuchungen des Lösungsansatzes wurde die Particle Computer Plattform gewählt.

3. Prozessorganisation

Die vorangegangenen Betrachtungen ubiquitärer Rechnersysteme erfordern neue Mechanismen der Zusammenarbeit von Prozessen und Systemen. In diesem Kapitel werden die theoretischen und praktischen Grundlagen der kooperativen und kollaborativen Prozessorganisation ubiquitärer Rechnersysteme gelegt. Beide Formen basieren auf dem Rückkopplungsprinzip. Durch Abgleich der rückgekoppelten Information und Regelung lässt sich gezielt Einfluss in das Ausführungsverhalten nehmen, um Ablaufgarantien und optimale Ressourcennutzung zu ermöglichen. Die rigorose Beschreibung kooperativer und kollaborativer Prozesse ist von besonderem Interesse. Regelungstechnische Verfahren sind notwendig, um die mathematische Modellbildung und Analyse rückgekoppelter Systeme zu unterstützen. Die mit diesen Werkzeugen vorgenommenen Untersuchungen erlauben es, eine methodische Vorgehensweise der Modellbildung für spezifische kooperative und kollaborative Prozesse festzulegen

Im folgenden Kapitel wird der Ansatz der Arbeit in ein internationales, wissenschaftliches Umfeld eingeordnet. Es werden mathematische Grundlagen und ein Systemmodell zur Beschreibung der Prozessorganisation eingeführt, sowie die Übertragung auf unterschiedliche Prozessklassen methodisch festgelegt. Die gewonnene Systemarchitektur kooperativer und kollaborativer Prozesse wird vorgestellt und Implementierungsaspekte für ubiquitäre Rechnersysteme diskutiert. Das Ergebnis: Es steht eine *neue Theorie der Prozessorganisation ubiquitärer Rechnersysteme* zur Verfügung.

3.1 Kooperation und Kollaboration

In diesem Abschnitt wird das grundlegende Verständnis kooperativer und kollaborativer Arbeitsweise von Prozessen gebildet. Kooperation und Kollaboration sind Mechanismen, die die Prozessausführung hinsichtlich eines gegebenen Verhaltens respektive Musters organisieren. Die Prozessausführung wird beobachtet und über eine Rückkopplung mit einem Zielwert verglichen und entsprechend angepasst. Der Zielwert beschreibt das erwünschte Muster der Ausführung. Die Prozessaufgabe ist für diese Betrachtung nicht relevant. Beobachtung und Zielwert beziehen sich auf

die Wirkung der Prozessausführung auf das ubiquitäre Rechnersystem. Die Wirkung muss mit einer Metrik erfasst werden. Beispiele solcher Metriken sind die Prozessorauslastung und der zur Verfügung stehende Energievorrat. Die Rückkopplung erlaubt es, das Zusammenspiel zwischen Anpassung der Prozessausführung und der über die Zeit veränderlichen Ausführung zu erfassen. Die Rückkopplung ist das fundamentale Arbeitsprinzip kooperativer und kollaborativer Prozessausführung.

Die folgende Abbildung zeigt kooperative und kollaborative Prozesse in einer schematischen Übersicht. Prozesse werden als Systeme dargestellt, die in miteinander

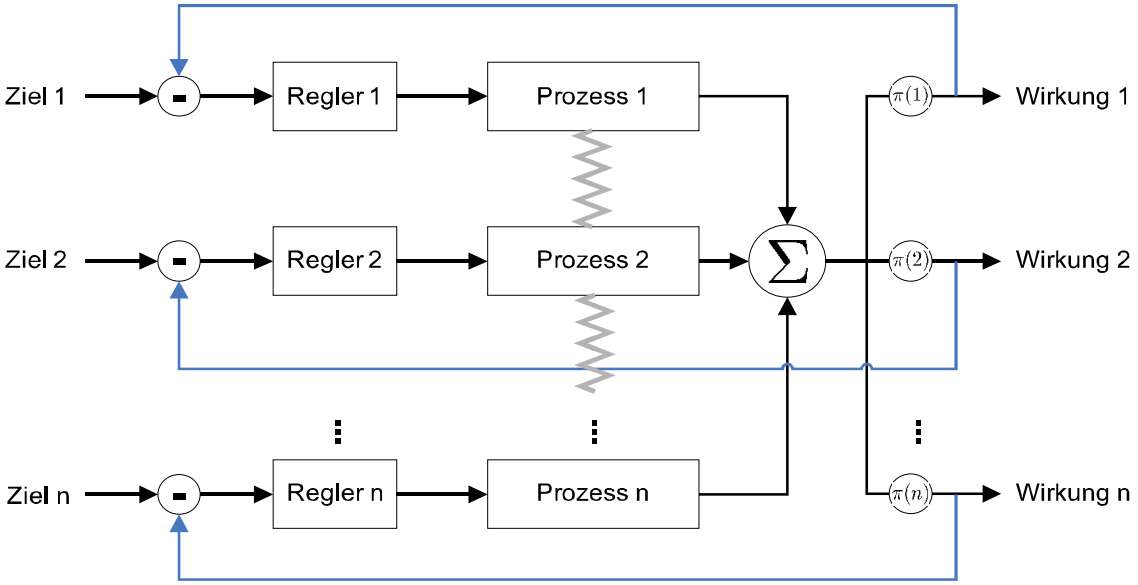


Abbildung 3.1: Schematische Übersicht kooperativer und kollaborativer Prozessausführung. Die Ausführung erzeugt eine Wirkung. Der Σ-Operator addiert die Wirkungen der Prozessausführung. Der π-Operator beschreibt die Wirkungen bezüglich der Einzelprozesse.

gekoppelte Regelkreise eingebettet sind. Der Regelkreis besteht aus einem Prozess, zum Beispiel dem Auslesen eines Sensors. Durch die Prozessausführung wird eine Wirkung auf andere Prozesse erzeugt, zum Beispiel die Verringerung eines gemeinsamen Energievorrates. Der Σ-Operator summiert die Einzelwirkungen der einzelnen Prozesse aufeinander auf. Das Ergebnis des folgenden π-Operators beschreibt die Wirkung des einzelnen Prozesses und die Wirkung aller anderen Prozesse in Bezug auf diesen. Diese Information wird zurückgeführt, mit einem Zielwert verglichen und bei Abweichung vom Zielwert ändert ein Regler zukünftige Wirkungen der Prozessausführung entsprechend. Im Beispiel könnte der Regler bei zu geringen Energieressourcen den Prozess verzögern oder abschalten.

Kooperation findet entlang jedes einzelnen Regelkreises statt. Die Regler sollen die Prozesse so einstellen, dass die jeweiligen Ziele bestmöglichst erreicht werden. Kooperation entsteht durch die Berücksichtigung der gegenseitigen Wirkungen aufeinander. Bis auf diese gegenseitige Wirkung sind kooperative Prozesse voneinander unabhängig.

Kollaborative Prozesse sind miteinander gekoppelt. Die Kopplung erfolgt durch Ressourcentransfer zwischen den Beteiligten. Kollaboration entsteht durch die organisierte Zusammenarbeit über diese Kopplung. Dabei soll ein gemeinsames Ziel er-

reicht werden. Ressourcen im physikalischen Sinn, zum Beispiel der Energievorrat eines Rechnersystems, können nicht transferiert werden. In Rechnersystemen sind Daten an Ressourcen gebunden, zum Beispiel der belegte Speicher in Datenpuffern. Mit einem Datentransfer geht somit ein Ressourcentransfer einher und realisiert die Kopplung. In diesem Zusammenhang führt Kollaboration auch zur Aufgabendelegation, wenn der Datentransfer die Nutzung weiterer Systemressourcen zur Datenverarbeitung erfordert. So bezieht eine drahtlose Kommunikation zwischen Rechnersystemen nicht nur den Sender ein, sondern es muss auch ein Empfänger und ein System zur Weiterverarbeitung betrieben werden. Die gemeinschaftliche Datenverarbeitung und -delegation macht Prozesse voneinander abhängig. Einzelziele werden zu einem gemeinsamen Ziel.

In der Abbildung 3.1 ist die Kollaboration zwischen den Prozessen als eine weitere Verbindung - symbolisch durch eine Feder - dargestellt. Kollaboration kann von einem Regelkreis nicht direkt beobachtet werden. Die Kopplung findet direkt zwischen den Prozessen statt und ist spezifisch auf die Prozessaufgaben ausgerichtet. Dennoch erzielt das Kollaborationsverhalten eine Wirkung bei der Prozessabarbeitung. Diese kann beobachtet werden und über den bereits beschriebenen Pfad an einen Regler zurückgeführt werden. In dieser Arbeit werden Kooperation und Kollaboration wie folgt verstanden:

Definition 3.1. *Kooperation und Kollaboration sind Muster der organisierten Prozessausführung ubiquitärer Rechnersysteme.*

Definition 3.2 (Kooperation)**.** *Das organisierte und zielgerichtete Zusammenwirken unabhängiger Prozesse aufeinander wird als Kooperation bezeichnet. Kooperation entsteht durch die Berücksichtigung der gegenseitigen Wirkungen der Prozessausführung aufeinander.*

Definition 3.3 (Kollaboration)**.** *Kollaboration bezeichnet die organisierte Zusammenarbeit von voneinander abhängigen Prozessen zum Erreichen eines gemeinsamen Ziels. Dabei sind die Prozesse über eine gemeinsame Datenverarbeitung und den Austausch von Ressourcen, zum Beispiel Daten oder Nachrichten, miteinander gekoppelt. Zusammenarbeit entsteht durch Delegation von (Teil-)Aufgaben an gekoppelte Prozesse.*

Erfolgsentscheidende Funktionen, die ein System zur Unterstützung von kooperativen und kollaborativen Prozessen zu erfüllen hat, sind:

Akquise von Laufzeitinformationen der Prozesse. Diese Informationen dienen als Leistungsmerkmal der Prozessausführung. Akquirierte Informationen bezeichnen die zu untersuchenden Wirkungen der Prozesse im allgemeinen Regelungsmodell.

Antizipation ist die Identifikation von auftretenden Störungen, die das Erreichen von Zielen verhindern können. Diese Informationen können aus der Akquise stammen, aber auch weitere Datenquellen einschließen.

Regelung hinsichtlich vorgegebener Ziele. Ein Regler verändert die Ausführungsparameter der Prozesse, zum Beispiel Perioden. Im Anschluss werden die in der Akquise gewonnen Informationen mit dem Regelungsziel verglichen. Weitere Regelungsaktivitäten sind davon abhängig wie gut das Ziel erreicht wurde.

3.2 Einordnung ins wissenschaftliche Umfeld

Das Zusammenwirken und die Zusammenarbeit von Rechnersystemen findet sich in anderen Forschungsbereichen wieder. Vor allem die Bereiche Middleware-Rahmenwerke und verteilte Systeme haben verschiedene Ansätze dafür entwickelt. Weiterhin werden Anwendungen der Regelungstechnik in Computersystemen untersucht.

Die Abbildung 3.2 zeigt die Einordnung des Ansatzes in Relation zu den oben genannten Bereichen. Diese Arbeit etabliert Kooperations- und Kollaborationsmechanismen als die tiefstliegende Schicht der Prozessorganisation integriert in Betriebssystem- und Laufzeitumgebungen.

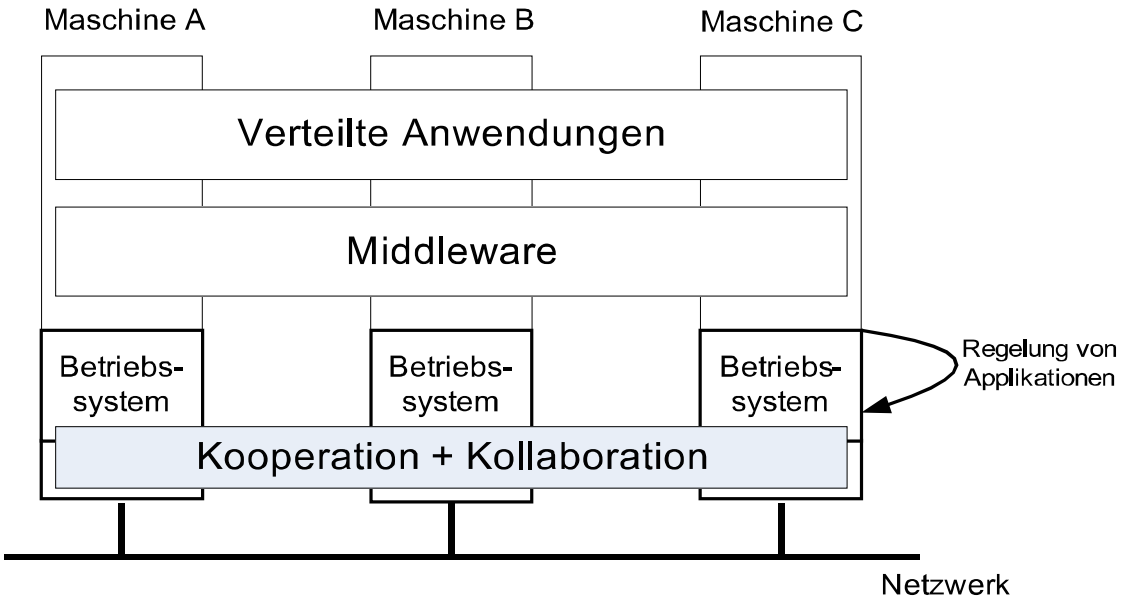


Abbildung 3.2: Kooperation und Kollaboration als fundamentaler Mechanismus der Prozessorganisation

3.2.1 Verteilte Systeme und Anwendungen

Ein verteiltes System gemäß der Definition von Tanenbaum [28] wird wie folgt beschrieben: *Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen.* Beispiele solcher Systeme sind das World Wide Web (WWW), verteilte Dateisysteme wie NFS [29] und Coda [30], verteilte Datei-Archive wie das peer-to-peer basierte BitTorrent [31] und verteilte Datenbanken wie Lotus Notes [32].

Wichtigstes Ziel ist die Transparenz, d.h. das Verbergen der Verteilung von Prozessen und Ressourcen. Fehlerzustände, Ausfälle, Ersatz und die Relokation von Systemen und Ressourcen sollen ebenfalls unbemerkt von der Anwendung behandelt werden. Die Realisierung eines verteilten Systems verwendet häufig eine weitere Softwareschicht, die Middleware, die verteilte Applikationen mit einzelnen darunterliegenden Betriebssystemen der vernetzten Plattformen verbindet.

Kooperations- und Kollaborationsmechanismen aus dieser Arbeit sind transparent für Prozesse. Sie können für jedes beteiligte Rechnersystem einer verteilten Anwendung verschiedene und systemspezifische Ziele verfolgen. Kollaboration erlaubt es,

die Ausführung verteilter Prozesse über Rechnersysteme hinweg zu koppeln, allein durch die Anpassung der lokalen Prozessausführung. Diese indirekte Einflussnahme ohne eine verbindende Middleware ist ein entscheidender Unterschied gegenüber verteilten Anwendungen.

3.2.2 Middleware

Middleware unterstützt die Entwicklung von verteilten Anwendungen durch die Bereitstellung von Mechanismen zur Interprozess-Kommunikation und Synchronisation über möglicherweise mehrere, heterogene und vernetzte Plattformen. Mechanismen wie Marshalling, das Dispatching zu Methoden, Ablaufsteuerung, Nachrichtentransport über verschiedene Netzwerke und Plattformen, werden in einem Middleware-Rahmenwerk implementiert und von den Anwendungen transparent benutzt. Diese Abstraktion ermöglicht es, verteilte Anwendungen einfacher zu erstellen, einfacher zu portieren und Details darunterliegender Betriebssysteme als Black-Box zu kapseln.

Einige spezielle Konzepte im Zusammenhang mit der Prozessorganisation werden im folgenden vorgestellt.

3.2.2.1 Adaptive und Reflective Middleware

Dynamische Systemlandschaften wie verteilte Multimedia-Anwendungen, Groupwaresysteme, eingebettete Echtzeitanwendungen oder Umgebungen mit mobilen Rechnersystemen erfordern Adaptionsmechanismen in Middleware-Rahmenwerken. Dies ist notwendig, um Ausfälle von Ressourcen zu kompensieren oder Ressourcen erneut einzubinden und nutzbar zu machen, die Leistungsfähigkeit hinsichtlich einer zu erbringenden Qualität (Quality of Service) zu optimieren oder Echtzeitanforderungen zu erfüllen.

Adaptive Middleware erlaubt das Verändern des Verhaltens einer verteilten Anwendung *nachdem* diese entwickelt wurde oder sogar bereits eingesetzt wird. Die Middleware bestimmt die Anzahl der Prozessanfragen, die Ausführungsreihenfolge und verwaltet die Verbindungen zu anderen Anwendungsteilen. Reflective Middleware wendet Instrumente der Introspektion und Adaption auf die Programmstruktur der Middleware an. Es wird dynamisch die Implementierung der Middleware zur Laufzeit verändert. So werden Netzwerkprotokolle, Sicherheitsrichtlinien, Algorithmen und andere Middleware-Mechanismen verändert oder ersetzt, um die optimale Systemleistung in verschiedenen Einsatzsituationen zu erreichen. Eine umfassende Übersicht von Middleware-Rahmenwerken mit Adaptionsmechanismen wird in [33] und [34] dargestellt.

Kooperations- und Kollaborationsmechanismen sind in der Laufzeitumgebung einer Plattform integriert und werden nicht nach außen exportiert. Sie können von den Adaptionsmechanismen der Middleware-Rahmenwerke nicht direkt genutzt werden. Unabhängig von der Applikation optimieren sie das Zusammenspiel aller Prozesse.

3.2.2.2 Organic Computing Middleware

Die Organic Computing Middleware hat die Aufgabe, selbständig Prozesse an Ressourcen verteilter Rechnersysteme zu delegieren und deren Zusammenarbeit zu organisieren. Dies erfolgt unter Einsatz der Prinzipien des Organic Computings [35], indem Mechanismen biologischer Systeme für Computersysteme nutzbar gemacht werden. Die self-X Eigenschaften [36], unter anderem self-configuration, self-healing und

self-optimization, ermöglichen eine dynamische Zuordnung von Prozessen auf Ressourcen, um komplexe wie auch robuste Systeme im Falle des Ausfalls von Ressourcen zu realisieren. Die Middleware organisiert diese Prozesse hinsichtlich Echtzeit-Scheduling, Fehlertoleranz und optimalen Ausführungsbedingungen. Ebenso erfolgt automatisch eine Reorganisation, wenn einzelne Komponenten des verteilten Gesamtsystems ausfallen.

Die OSA+ Middleware [37] basiert auf einer Mikrokern-Architektur und wurde um einen Organic Manager erweitert [38][39]. Der Organic Manager steuert die Prozessverwaltung nach den self-X Prinzipien als eigenständiger Service [40]. Ein mögliches Einsatzszenario von OSA+ ist ein Warenlager. Darin transportieren Roboter Waren und Güter. Im Falle eines Ausfalls eines Roboters sorgt die Organic Middleware selbständig dafür, dass die verbliebenen Roboter die Aufgaben übernehmen. Die Umverteilung soll so erfolgen, dass die Kosten minimal sind.

Kooperations- und Kollaborationsmechanismen optimieren die Prozessausführung durch Veränderung von Prozessablaufparametern, zum Beispiel Ausführungsperiode, unter Berücksichtigung vorhandener Ressourcen. Fehlertoleranz wird nicht betrachtet. Kooperations- und Kollaborationsmechanismen berücksichtigen allerdings sich dynamisch verändernde Eigenschaften von Prozessen wie beispielsweise die Prozessausführungsdauer.

3.2.3 Regelung von Computersystemen

Regelkreise fanden zuerst Anwendung in der Flusskontrolle von zeitlich schwankendem Netzwerkverkehr in TCP-Netzwerken [41] und ATM-Netzwerken [42]. Regelungstechnik wurde für die Analyse des Random Early Detection (RED) Mechanismus in IP Routern angewendet [43], um geeignete RED Parameter für die Latenzzeit des Datenverkehrs und den Auslastungsgrad von Netzwerken zu bestimmen.

Serversysteme, zum Beispiel Webserver, E-Mail-Server und Datenbanken, kontrollieren mittels Regelkreisen Dienstgüteanforderungen (engl. Quality of Service, QoS) aus sogenannten service level agreements (SLA). Sie präferieren bestimmte Verbindungen, regulieren die Serverauslastung [44] [45] und optimieren Antwortzeiten [46]. Die Arbeiten führten zur Entwicklung von ControlWare [47], einer Middleware, die Grundprimitive für die QoS Regelung von Anwendungsprogrammen zur Verfügung stellt.

Stankovic [48] verwendet Regelkreise in Laufzeitumgebungen, um Echtzeitbedingungen verteilter Prozesse unter dynamischen Prozessorauslastungen und schwankenden Prozesslaufzeiten zu ermöglichen. Distributed Feedback Control Real-time Scheduling (DFCS) adaptiert ein Qualitätsmaß (QoS) der Prozesse, um die Auslastung verteilter Prozessoren zu maximieren und die Anzahl der Überschreitungen der Zeitschranke zu minimieren.

Control Kernel [49] ist ein Ansatz zur Erweiterung für Betriebssystemkerne für Regelungsaufgaben. Dies vereinfacht die Entwicklung von auf Mikrocontrollern implementierten Reglern für komplexe Industrieszenarien wie Produktionsstrassen oder Industrieanlagen. Die Regelung wird nicht auf den Prozessablauf in den Computersystemen angewendet.

Die Kooperations- und Kollaborationsmechanismen dieser Arbeit beschränken sich nicht auf einzelne Anwendungen. Sie sind ein Teil der Laufzeitumgebung des jeweiligen Rechnersystems und werden auf alle Prozesse angewendet. Kooperation und Kollaboration sind Dienstgüte-Mechanismen (QoS) der Prozessorganisation.

3.2.4 Einordnung und Charakterisierung

Kooperation und Kollaboration sind regelungstechnische Dienstgüte-Mechanismen (QoS) zur Organisation von Prozessen. Prozesse können den Grad der Kooperation und Kollaboration, und die damit verbundene Dienstgüte (QoS), einstellen. Im Gegensatz zu Middleware-Ansätzen ist der Eingriff in den Ablauf der Mechanismen nicht möglich. Im Vergleich zu Middleware-Ansätzen handelt es sich um tiefer liegenden Mechanismen - integriert im Betriebssystem bzw. der Laufzeitumgebung eines Rechnersystems. Kooperation und Kollaboration beeinflussen direkt die Art und Weise wie die Laufzeitumgebung Prozesse ausführt. Damit können alle Prozesse, unabhängig von der sie implementierenden Aufgabe, Kooperation und Kollaboration sofort nutzen. Prozesse können weiterhin darüberliegende Middleware-Mechanismen verwenden. Kooperations- und Kollaborationsmechanismen werden zu Systemeigenschaften.

Zusammenfassend kann man die Kooperations- und Kollaborationsmechanismen wie folgt einordnen und charakterisieren:

- Tiefste Schicht der QoS-orientierten Prozessorganisation in Rechnersystemen
- Transparente Anwendung auf alle Prozesse
- Bieten keine Schnittstelle an. Der Grad der Kooperation und Kollaboration ist einstellbar.
- Organisieren das Zusammenwirken und die Kopplung von Prozessen durch Rückkopplung und Regelung.

Die Rückkopplung ist ein mächtiges Instrument, welches es erlaubt, ein gewünschtes Prozessverhalten zu erzielen. Allerdings kann sich auch ein unerwünschtes Verhalten schnell potenzieren und das Rechnersystem unkontrollierbar machen. Daher müssen in den folgenden Abschnitten die Werkzeuge zum Umgang mit diesem Instrument erschlossen werden.

3.3 Grundlagen der Rückkopplung und Regelung

Prozesse eingebetteter ubiquitärer Rechnersysteme wechselwirken stark miteinander. Bei der Prozessausführung tritt dabei eine Rückkopplung auf. Das Rückkopplungsprinzip sagt aus, dass die aktuelle Prozessausführung auf die Ausführung zukünftiger Prozesse zurückwirken. Zum Beispiel steht der verbrauchte Energievorrat durch eine Sensordatenakquise für nachfolgende Prozesse nicht mehr zur Verfügung. Die rückgekoppelte Information über den aktuellen Energievorrat kann für die Planung zukünftiger Prozesse verwendet werden, um im Zusammenwirken aller Prozesse eine vorgegebene Betriebsdauer zu erreichen. Zur Beschreibung und Analyse der Kooperations- und Kollaborationsmechnismen werden im folgenden systemtheoretische und regelungstechnische Grundlagen eingeführt.

3.3.1 Regelkreise

In Abbildung 3.3 wird das Bild eines allgemeinen Regelkreises gezeichnet. Im Zentrum steht dabei die Regelstrecke, auch Zielsystem genannt. Die Rückkopplung leitet die Ausgangsgröße des Systems zurück zum Vergleich mit einer Eingangsgröße, die auch Sollwert oder Führungsgröße genannt wird. Das Ergebnis des Vergleichs, der Regelungsfehler, wird vom Regler verwendet, um den Eingang der Regelstrecke so zu verändern, dass der Regelungsfehler aufgehoben wird. Damit ist der Regelkreis in der Lage Störungen, die auf das Zielsystem wirken, zu kompensieren. Das Verhalten

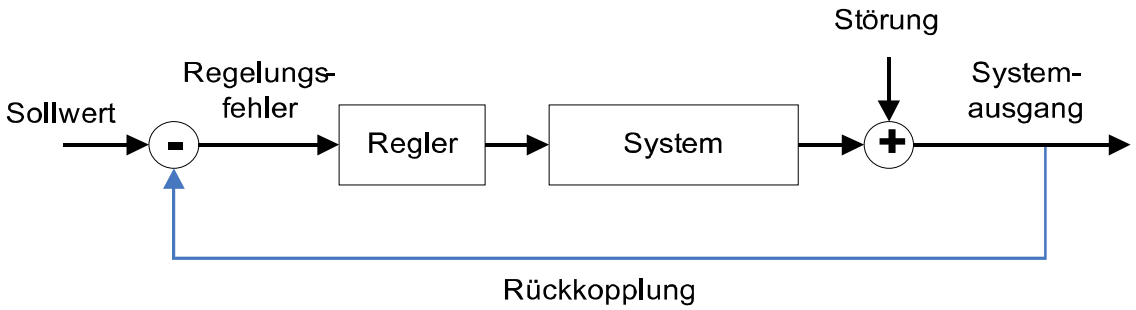


Abbildung 3.3: Blockschaltbild eines allgemeinen Regelkreises

von rückgekoppelten Regelungssystemen wird mittels Eigenschaften wie Stabilität, Genauigkeit und Geschwindigkeit beschrieben. Dazu werden die einzelnen Blöcke und ihr Zusammenspiel im Regelkreise mathematisch analysiert. Jeder Block wird im allgemeinen als System bezeichnet, welches Eingangssignale in Ausgangssignale umwandelt.

3.3.2 Mathematische Beschreibung

Die durch den Systembegriff hergestellte Verknüpfung zwischen Eingangs- und Ausgangsgrößen kann mit einer Operatorbeziehung ausgedrückt werden. Es bezeichnet

$$y(t) = F(u(t)) \tag{3.1}$$

die Umwandlung des Eingangssignals $u(t)$ in das Ausgangssignal $y(t)$ durch das System F.

3.3.2.1 Grundlegende Systemarten

Die einfachste Form sind *statische Systeme*. Dabei hängt das Ausgangssignal $y(t)$ zu jedem Zeitpunkt t nur vom Wert des Eingangssignals $u(t)$ ab. Ein einfaches Beispiel ist die Verstärkung eines Signals um einen konstanten Faktor, $y(t) = F(u(t)) = Ku(t)$. Im Zusammenhang mit dieser Arbeit kann ein statisches System einen Prozessausführungsparameter, zum Beispiel die Periode, bestimmen.

Im Vergleich dazu bestimmt sich der Ausgang von *dynamischen Systemen* durch vergangene Eingabesignale aus dem Intervall $[t-\tau, t]$, also $y(t) = F(u(t)) = f([u(t-\tau), u(t)])$. Ein einfaches Beispiel ist ein Integrator: $y(t) = \int_{\mu=t-\tau}^{t} u(\mu)$. In Rechnersystemen lässt sich durch einen Integrator das Verhalten von Datenpuffern modellieren.

Es sind ebenso Systeme mit mehreren Ein- und Ausgängen realisierbar. Sie werden entsprechend nach Single-Input Single Output (SISO), Single-Input Multiple-Output

(SIMO), Multiple-Input Single-Output (MISO) und Multiple-Input Multiple-Output (MIMO) klassifiziert. Für Nicht-SISO Systeme sind die Systemmodelle komplexer. Um die begrenzten Ressourcen der betrachteten eingebetteten ubiquitären Rechnersysteme zu berücksichtigen, beschränkt sich diese Arbeit auf SISO Systeme.

Zur weiteren mathematischen Beschreibung ist noch die Zeitentwicklung zu betrachten. In *diskreten Systemen* entwickelt sich der Systemausgang in festen, endlichen Zeitabschnitten. *Kontinuierliche Systeme* ändern ihren Systemausgang in infinitesimal kleinen Zeitschritten. Die mathematische Beschreibung erfolgt in diskreten Systemen mittels Differenzengleichungen, während für kontinuierliche Systeme Differentialgleichungen verwendet werden. Prozesse in Rechnersystemen werden in ihrem Verhalten wie diskrete Systeme betrachtet. In festen Zeitabschnitten, zum Beispiel bei periodischer Ausführung von Prozessen, verändert sich der Ausgang des Systems.

3.3.2.2 Lineare zeitinvariante Systeme

Lineare Systeme verknüpfen Ein- und Ausgang über eine strukturerhaltene Abbildung. Es müssen Homogenität und Additivität der Signale durch das System erfüllt werden. Homogenität bedeutet, dass die Verstärkung des Eingangssignals zu einer gleichen Verstärkung des Ausgangssignals führt. Additivität erhält die Eigenschaften bei der Superposition von Signalen. Linearität ist analog sowohl für zeitdiskrete wie kontinuierliche Systeme definiert. Aufgrund des Einsatzes in Rechnersystemen wird hier lediglich die Formulierung mit zeitdiskreten Signalen $u(k)$ und $y(k)$ gegeben. Der Systemoperator F muss folgende Bedingung erfüllen:

$$\textbf{Homogenität:} \quad F(au(k)) = aF(u(k)) = ay(k) \tag{3.2}$$

$$\textbf{Additivität:} \quad F(u_1(k) + u_2(k)) = F(u_1(k)) + F(u_2(k)) = y_1(k) + y_2(k) \tag{3.3}$$

Zusätzlich wird noch gefordert, dass die Form des Ausgangssignals unabhängig vom Zeitpunkt des einsetzenden Eingangssignals ist. Diese Eigenschaft bezeichnet man als *Zeitinvarianz*. Für jedes zeitinvariante System gilt

$$\forall k_0 : F(u(k + k_0)) = y(k + k_0) \tag{3.4}$$

Es wird nun die allgemeine Differenzengleichung für zeitdiskrete lineare zeitinvariante (LZI) Systeme angegeben:

$$a_1 y(k) + a_2 y(k-1) + \cdots + a_m y(k-m) = b_1 u(k) + \cdots + b_n u(k-n) \tag{3.5}$$

Die Theorie der LZI Systeme ist sehr gut ausgebaut [50]. In dieser Arbeit werden zur Beschreibung und Analyse von Kooperations- und Kollaborationsmechanismen LZI Systeme verwendet.

3.3.2.3 Z Transformation

Zur Analyse von Systemen werden mathematische Transformationen verwendet. Die Z Transformation wandelt ein zeitdiskretes Signal aus dem Zeitbereich in eine Darstellung in den Frequenzbereich um. Die Transformation ist mittels einer komplexen Zahl z wie folgt definiert:

$$U(z) = Z[u(k)] = \sum_{k=0}^{\infty} u(k) z^{-k} \tag{3.6}$$

Diese Transformation besitzt folgende wichtige Eigenschaften [50]:

$$\textbf{Linearität:} \quad Z[a_1u_1(k) + a_2u_2(k)] = a_1Z[u_1(k)] + a_2Z[u_2(k)] \tag{3.7}$$

$$\textbf{Verschiebung:} \quad Z[u(k-n)] = z^{-n}Z[u(k)] \tag{3.8}$$

$$\textbf{Faltung:} \quad u(k) * v(k) = Z[u(k)]Z[v(k)] \tag{3.9}$$

$$\textbf{Grenzwertsatz:} \quad \lim_{k\to\infty} y(k) = \lim_{z\to 1}(z-1)Y(z) \tag{3.10}$$

Diese Eigenschaften qualifizieren die Z Transformation zur Beschreibung linearer, zeitinvarianter, diskreter Systeme.

3.3.2.4 Transferfunktionen

Transferfunktionen beschreiben das Übertragungsverhalten von Systemen. Sie beschreiben mathematisch die Umwandlung des Eingangssignals in das Ausgangssignal. Mittels der Z Transformation lassen sich Transferfunktionen sehr elegant formulieren. Der Systemoperator F kann wie folgt in der Z Transformation dargestellt:

$$F(z) = \frac{Y(z)}{U(z)} \tag{3.11}$$

Die allgemeine Operatorbeziehung aus Gleichung 3.1 kann als eine Multiplikation der Transferfunktion mit einem beliebigen z-transformierten Eingangssignal dargestellt werden: $Y(z) = F(z)U(z)$. Die Transferfunktion des Systems $F(z)$, welches durch Konkatenation der Teilsysteme $F_1(z)$ und $F_2(z)$ entsteht, ist durch eine Multiplikation der einzelnen Transferfunktionen beschrieben: $F(z) = F_1(z)F_2(z)$. Mit diesen Voraussetzungen lässt sich nun die Transferfunktion für rückgekoppelte Systeme aufstellen. In Abbildung 3.4 ist der Regelkreis mit Signalbezeichnungen angegeben. Die

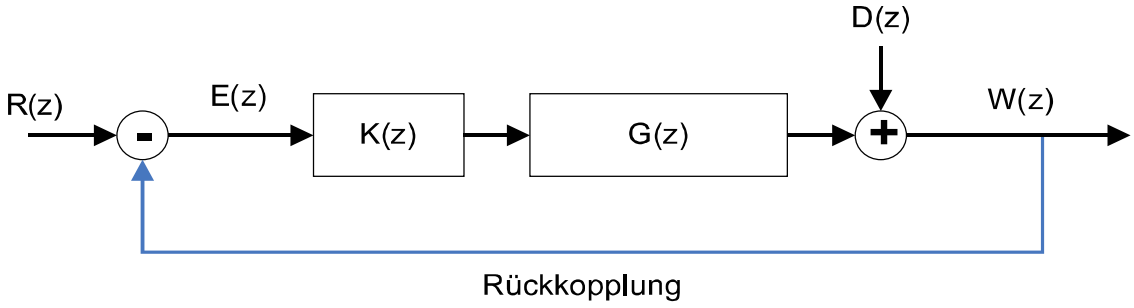


Abbildung 3.4: Blockschaltbild eines allgemeinen Regelkreises mit Signalbezeichnungen

Transferfunktion des Regelkreises $F(z)$ setzt den Sollwert $R(z)$ mit dem Ausgang $W(z)$ in Beziehung. Die Blöcke $K(z)$ und $G(z)$ sind wiederum Systeme, die hier nicht weiter spezifiziert sind. Die Störung wird mit $D(z) = 0$ angenommen. Es ergibt sich die Transferfunktion des Regelkreises gemäß der Herleitung im Anhang A:

$$F(z) = \frac{W(z)}{R(z)} = \frac{K(z)G(z)}{1 + K(z)G(z)} \tag{3.12}$$

Regelkreise mit Eingangssignal $r(k)$ und Ausgangssignal $w(k)$ können mit Transferfunktionen aus den Polynomen der Signale formuliert werden. Ausgehend von der allgemeinen Differenzengleichung 3.5 erhält man durch Z Transformation

$$\begin{aligned} a_0w(k) + a_1w(k-1) + \cdots + a_mw(k-m) &= b_0r(k) + \cdots + b_nr(k-n) \\ Z\left[a_0w(k) + a_1w(k-1) + \cdots + a_mw(k-m)\right] &= Z\left[b_0r(k) + \cdots + b_nr(k-n)\right] \\ a_0W(z) + a_1z^{-1}W(z) + \cdots + a_mz^{-m}W(z) &= b_0R(z) + \cdots + b_nz^{-n}R(z) \end{aligned}$$

Die allgemeine Übertragungsfunktion lässt sich als Quotient der Polynome aus Eingangs- und Ausgangssignalen wie folgt darstellen:

$$F(z) = \frac{W(z)}{R(z)} = \frac{b_0 + b_1z^{-1} + b_2z^{-2} + \cdots + b_nz^{-n}}{a_0 + a_1z^{-1} + a_2z^{-2} + \cdots + a_mz^{-m}} \tag{3.13}$$

Die aufgestellte Gleichung 3.13 betrachtet den Regelkreis als SISO LZI System und abstrahiert von allen internen Elementen des Regelkreises. Der Sollwert des Regelkreises wird durch das Eingangssignal $r(k)$ bzw. $R(z)$ repräsentiert und der Ausgang entspricht $w(k)$ bzw. $W(z)$. Die Gleichung ist Grundlage der Untersuchungen wichtiger Eigenschaften im nächsten Abschnitt.

3.3.3 Wichtige Eigenschaften rückgekoppelter Systeme

Die Untersuchung wichtiger Eigenschaften wie Stabilität, Schnelligkeit und Genauigkeit, erlaubt zugesicherte Aussagen über rückgekoppelte Wirkungen der Prozessausführung auf ubiquitären Rechnersystemen zu geben. Ubiquitäre Rechnersysteme bleiben kontrollierbar. Für die Analyse wird vorausgesetzt, dass Regelkreise zeitdiskrete LZI Systeme mit Transferfunktion wie in Gleichung 3.13 sind.

3.3.3.1 Stabilität

Ein System soll als stabil bezeichnet werden, wenn jedes beschränkte zulässige Eingangssignal ein beschränktes Ausgangssignal zur Folge hat [50]. Diese Eigenschaft bezeichnet man auch als BIBO (engl. bounded-input bounded-output) Stabilität. Für die Prozessorganisation ubiquitärer Rechnersysteme ist diese Eigenschaft besonders wichtig. Trotz eines Störeinflusses, zum Beispiel ein unvorhergesehenes Ereignis oder Prozessverhalten, kann das Laufzeitsystem die Kooperation und Kollaboration von Prozessen sicherstellen.

Die Polynome der allgemeinen Transferfunktion aus Gleichung 3.13 lassen sich mittels des Fundamentalsatzes der Algebra in eine Produktform ihrer Nullstellen bringen. In einem zweiten Schritt kann daraus eine Partialbruchdarstellung abgeleitet werden

$$F(z) = \frac{\prod_{i=1}^{m}(z - n_i)}{\prod_{j=1}^{n}(z - p_j)} = \frac{Az}{z - p_1} + \frac{Bz}{z - p_2} + \cdots + \frac{Nz}{z - p_n} \tag{3.14}$$

Die Größen n_i und p_j bezeichnen die Nullstellen des Zählers und Nenners. Letztere werden auch als Pole des Systems bezeichnet. A bis N sind unbekannte Teile des Zählers nach der Partialbruchzerlegung und können im konkreten Fall bestimmt werden. Für die Stabilitätsbetrachtung ist dies nicht notwendig. Gleichung 3.14 ist noch z-transformiert und wird nun in den Zeitbereich rücktransformiert. Das Systemverhalten des Regelkreises ist vollständig durch die Impulsantwort bestimmt.

Damit liefert die Rücktransformation $Z^{-1}[F(z)] = F(k)$ die Stabilitätsaussage für alle zulässigen Eingangssignale $r(k)$ des Regelkreises.

$$Z^{-1}[F(z)] = F(k) = Ap_1^k + Bp_2^k + \cdots + Np_n^k \tag{3.15}$$

Die Rücktransformierte besteht aus einer Summe von Exponentialfunktionen der Pole des Systems. Ist $|p_i| < 1$, so sieht man sofort, dass Gleichung 3.15 gegen 0 konvergiert. Geometrisch interpretiert liegen für ein solches System die Pole innerhalb des Einheitskreises der komplexen Zahlenebene.

Das System ist BIBO stabil, gdw. $\forall p_i : |p_i| < 1$.

Lassen sich Prozesse ubiquitärer Rechnersysteme in einen solchen Regelkreis einbetten, der diese Bedingung erfüllt, dann kann das Laufzeitsystem die Kooperation und Kollaboration der Prozesse garantieren.

3.3.3.2 Genauigkeit

Die Genauigkeit gibt Aufschluss darüber wie genau der Regelkreis einen gegebenen Sollwert erreicht. Für ubiquitäre Rechnersysteme ist dies ein Gütemaß der kooperativen und kollaborativen Prozessausführung. Es wird vorausgesetzt, dass das System BIBO stabil ist. Als Sollwerteingabe wird die Schrittfunktion $\frac{z}{z-1}$ gewählt, die das Regelkreissystem stimuliert. Mit dem Grenzwertsatz aus Gleichung 3.10 ergibt sich für die Transferfunktion $F(z)$ des Regelkreises

$$\begin{aligned} \lim_{k\to\infty} w(k) = w_{\text{stat}} &= \lim_{z\to 1}(z-1)F(z)\frac{z}{z-1} \\ &= \lim_{z\to 1} F(z)z \\ &= F(1) \end{aligned} \tag{3.16}$$

Im stationären Zustand mit konstantem Sollwert r_{stat} ist $F(1) = \frac{w_{\text{stat}}}{r_{\text{stat}}}$ die Transferfunktion des Regelkreises. Damit lässt sich folgende Aussage über den stationären Regelfehler e_{stat}, d.h. die Genauigkeit der Regelung, herleiten:

$$\begin{aligned} e_{\text{stat}} &= \lim_{k\to\infty}[r(k) - w(k)] \\ &= r_{\text{stat}} - w_{\text{stat}} \\ &= r_{\text{stat}} - F(1)r_{\text{stat}} \\ &= r_{\text{stat}}(1 - F(1)) \\ e_{\text{stat}} = 0 &\Leftrightarrow F(1) = 1 \end{aligned} \tag{3.17}$$

Das Ausgangssignal des Regelkreissystems erreicht exakt den gegebenen Sollwert, wenn $F(1) = 1$ gilt. Lassen sich Prozesse ubiquitärer Rechnersysteme in einen solchen Regelkreis einbetten, der diese Bedingung erfüllt, so kann höchste Güte der Prozesskooperation und -kollaboration unter einem gegebenen Sollwert garantiert werden. Für die allgemeine Transferfunktion aus Gleichung 3.13, die als Grundlage für die Betrachtungen der Eigenschaften verwendet wird, wird der Regelfehler theoretisch erst nach unendlich vielen Zeitschritten exakt 0. Im praktischen Fall sind die Abweichungen vom Sollwert nach einigen Schritten aber schon sehr klein, so dass oftmals nur eine größte Fehlergrenze ϵ, zum Beispiel $\epsilon = 2\%$, gefordert wird.

3.3.3.3 Schnelligkeit

Bei Änderung des Sollwertes oder unter dem Einfluss einer Störung steuert der Regler die Regelstrecke, damit der neue Sollwert erreicht wird und die Störung ausgeglichen wird. Voraussetzung ist, dass BIBO Stabilität des Regelkreises vorliegt. Die Schnelligkeit gibt an, nach welcher Zeitspanne, d.h. Anzahl an diskreten Zeitschritten, der entsprechende Zielwert anliegt. Für die Prozessorganisation bestimmt die Schnelligkeit die Zeitspanne, in der ein vorgegebenes kooperatives oder kollaboratives Verhalten der Prozesse erreicht wird.

Aus Gleichung 3.15 ist ersichtlich, dass der Term mit dem höchsten Exponenten und dem größten Betrag der Polstelle das System dominiert und damit die Schnelligkeit bestimmt. Angenommen p_1 besitzt den größten Betrag aller Pole. Es existiert eine Fehlergrenze ϵ um den Grenzwert 0, den der Term Ap_1^k nach k Schritten erreicht. Für $\epsilon = 0.02$ gilt $0.02 \approx Ap_1^k$. Umgestellt nach k, ergibt sich die Anzahl der Zeitschritte in Gleichung 3.18:

$$k \approx \frac{\ln 0.02 - \ln A}{\ln p_1} \tag{3.18}$$

Die Gleichung 3.18 beschreibt die Schnelligkeit der Prozessorganisation ubiquitärer Rechnersysteme.

3.3.4 Reglerentwurf

Die Wahl des Reglers bestimmt die oben analysierten Eigenschaften. Ein geeigneter Regler kann zu sehr schnellen, genauen und stabilen Systemen führen. Im schlimmsten Fall kann ein Regler aber auch zu Instabilität führen. In diesem Fall würden die Kooperations- und Kollaborationsechanismen der Prozessorganisation versagen. Die Rückkopplung würde zu einer unkontrollierbaren Prozessausführung auf ubiquitären Rechnersystemen führen.

3.3.4.1 Aufgabe

Der Regler hat die Aufgabe, die Ausgabe des Systems und den Sollwert in Übereinstimmung zu bringen. Es muss der Regelungsfehler $e(k) = 0$ erreicht werden. Dazu implementiert der Regler ein Regelungsgesetz, welches als Eingabe den Regelungsfehler $e(k)$ verwendet und als Ausgabe das Signal $u(k)$, welches das System so stimuliert, dass der Systemausgang dem Sollwert entspricht. Die klassischen Regelungsgesetze sind der P-Regler, der PI-Regler und der PID-Regler.

3.3.4.2 Methode des Reglerentwurfs

Regelungsgesetze sind parametrisierbare Gleichungen. Das Ziel des Reglerentwurfs ist es, diese Parameter zu bestimmen. Die Abbildung 3.5 illustriert die methodische Herangehensweise des Entwurfs von Regelungssystemen. Zu Beginn wird ein dynamisches Systemmodell erstellt. Dies geschieht analytisch auf Basis von (physikalischen) Gesetzmäßigkeiten oder mittels der Methoden der Systemidentifikation. Anschließend wird ein geeignetes Regelungsgesetz gesucht und parametrisiert. Auch hierbei gibt es mehrere Methoden. Der Regler wird hinsichtlich der Systemanforderungen wie Stabilität, Schnelligkeit und Genauigkeit überprüft und eventuell iterativ in mehreren Schritten angepasst, um die Anforderungen zu erfüllen.

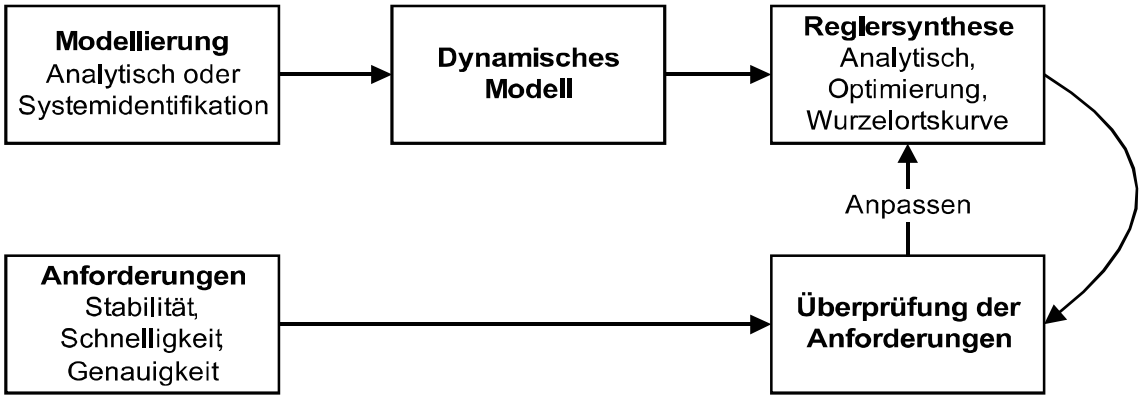


Abbildung 3.5: Illustration des Reglerentwurfs

Wie die allgemeine Systemgleichung 3.12 des rückgekoppelten Regelkreises $F(z) = \frac{K(z)G(z)}{1+K(z)G(z)}$ zeigt, beeinflusst der Regler $K(z)$ die Pole des Regelkreissystems. Generell verfolgt der Reglerentwurf das Ziel, durch geeignete Parameter in den Regelungsgesetzen die Pole so zu setzen, dass die Anforderungen an Stabilität, Schnelligkeit und Genauigkeit erfüllt werden. Folgende Methoden haben sich zur Parametrisierung der Regelungsgesetze bewährt:

Analytische Methode: Es wird die Systemantwort im Zeitbereich auf eine Treppenfunktion berechnet. Gemäß dieser Gleichung können die Parameter des Regelungsgesetzes nach den Anforderungen bestimmt werden. Diese Methode ist exakt, die Gleichungen können jedoch sehr komplex sein, was eine Parametrisierung erschwert.

Optimierung: Die Parameter des Regelungsgesetzes werden als Optimierungsproblem dargestellt. Die Zielfunktion wird anhand der Anforderungen formuliert. Optimierungsprobleme können komplexe Situationen erfassen, benötigen oftmals jedoch große Rechenleistungen. Da die Synthese offline, d.h. vor Inbetriebnahme des ubiquitären Rechnersystems erfolgt, ist dieser Ansatz akzeptabel.

Wurzelortskurve: Die Wurzelortskurve präsentiert grafisch Trajektorien aller möglicher Polstellen des Systems. Das Setzen der Pole bestimmt die Eigenschaften. Diese Methode ist nur für P-Regler anwendbar.

Die Auswahl einer geeigneten Methode hängt von der Modellierung des Regelkreissystems ab. Es lässt sich im Voraus nicht sagen, welche Methode am geeignetsten ist.

3.3.4.3 Regelungsgesetze und Einsetzbarkeit

Tabelle 3.1 listet die klassischen Regelungsgesetze und ihre besonderen Merkmale auf. Der Einsatz auf eingebetteten ubiquitären Systemen ist für alle Regler möglich. Wenn die Systemmodellierung des rückgekoppelten Regelkreises mit einem P-Regler die Bedingung $F(1) = 1$ aus Abschnitt 3.3.3.2 erfüllt, ist der P-Regler aufgrund seiner Einfachheit zu bevorzugen.

Regler	Regelungsgesetz	Vorteile	Nachteile
P-Regler	$u_P(k) = Ke(k)$	sehr einfach	mögliche stationäre Regeldifferenz
PI-Regler	$u_{PI}(k) = u_P(k) + \frac{1}{T_I}\sum_{k=0}^{\infty} e(k)$	keine stationäre Regeldifferenz	Schwankung der Systemantwort, möglicherweise instabile Systeme
PID-Regler	$u_{PID}(k) = u_{PI}(k) + T_D\frac{e(k)-e(k-1)}{T}$	Differentialglied kann PI-Regler stabilisieren	diskrete Approximation ungenau, nihiliert Vorteil

Tabelle 3.1: Regelungsgesetze

3.4 Modellierung von Kooperation und Kollaboration

Im vorangegangenen Abschnitt wurden mit der Regelungstheorie die mathematischen Grundlagen zur Behandlung von Rückkopplungen in Rechnersystemen gelegt. Es wird nun gezeigt, wie Prozesse in Regelkreise eingebettet werden, um Kooperation und Kollaboration zu ermöglichen.

3.4.1 Systemidentifikation und Systemdesign

Im ersten Schritt nach Abbildung 3.5 gilt es, ein Systemmodell zu bestimmen. Dies ist für Prozesse in Rechnersystemen nicht trivial. Eine Anforderung dieser Arbeit ist die Abstraktion von den konkreten Prozessaufgaben, um die größtmögliche Flexibilität für alle Prozesse zu erreichen. Auf der anderen Seite sind dadurch nur wenige Informationen über Prozesse verfügbar. Erschwerend kommt hinzu, dass es keine physikalischen Bilanzgleichungen gibt, da Prozesse Softwarekomponenten sind.

Als Alternative bietet sich die Systemidentifikation [51] an. Dabei lässt man die Prozesse in einem realen System ablaufen und beobachtet deren Wirkungen auf die Systemressourcen. Bei veränderten Eingaben, zum Beispiel zu verarbeitende Daten, erwartet man ein Muster in den Wirkungen, welches man durch eine Gleichung darstellen kann. Identifikationsverfahren sind meistens sehr rechenaufwendig, da die gesammelten Daten mit einem gewählten Modell in Übereinstimmung gebracht werden müssen. Ein weiterer Nachteil ist die lange Beobachtungszeit und die Wiederholung der Beobachtung bei Veränderungen, Hinzufügen oder Entfernen von Prozessen. Außerdem können Prozesse noch interne, nicht beobachtbare Zustände aufweisen, die das Verhalten trotz gleicher Eingaben verändern.

Da die Implementierungen der Prozesse auf ubiquitären Systemen auch während des Betriebs als austauschbar angenommen werden, ist eine Systemidentifikation für die Modellierung nicht geeignet. Es muss ein System entworfen werden, welches die Wirkung von Prozessen aufeinander modelliert.

3.4.2 Budget/Kosten Modell

Die Wirkungen von Prozessen werden als Kosten auf ein abstraktes Budget modelliert. So stellt beispielsweise der Energieverbrauch eines ubiquitären Rechnersystems beim Prozess der Sensordatenakquise Kosten dar, die auf den Gesamtenergievorrat des Systems zurückwirken. Der Vorrat wird durch ein Budget ausgedrückt. Oder der Empfang von Daten (= Kosten) füllt einen Datenpuffer (= Budget). Bei diesen Beispielen wird der Prozess selbst nicht modelliert. Dies erlaubt ein Vorgehen, welches unabhängig von der Implementierung von Prozessen ist und damit die zentrale Anforderung nach einem fundamentalen Organisationsprinzip für Prozesse erfüllt.

Kooperation und Kollaboration ergeben sich nun durch die kontrollierte Änderung des Budgets, die jeder Prozess während seiner Laufzeit verursacht. Bei der Kooperation werden die Prozesse vom Laufzeitsystem so beeinflusst, dass sie bestimmte Kosten verursachen. Dabei werden die Prozesse unabhängig voneinander beeinflusst. Kollaboration koppelt die Prozesse, indem zwischen den Prozessen Budget transferiert wird. Die Kosten der Prozessausführung werden zusätzlich von den Zugeständnissen anderer Prozesse beeinflusst. Dadurch entsteht eine Wechselwirkung zwischen Budgetlieferanten und Budgetempfängern bei der Prozessausführung.

Nach der Ausführung der Prozesse wird das tatsächlich entstandene Budget zur Berechnung der Kosten der nächsten Ausführung an den Regler zurückgeführt. Das rückgekoppelte Budget/Kosten-Modell wird in Abbildung 3.6 durch das Blockbild eines Regelkreises illustriert. Ziel des Regelkreises ist es, ein vorgegebenes Soll-Budget

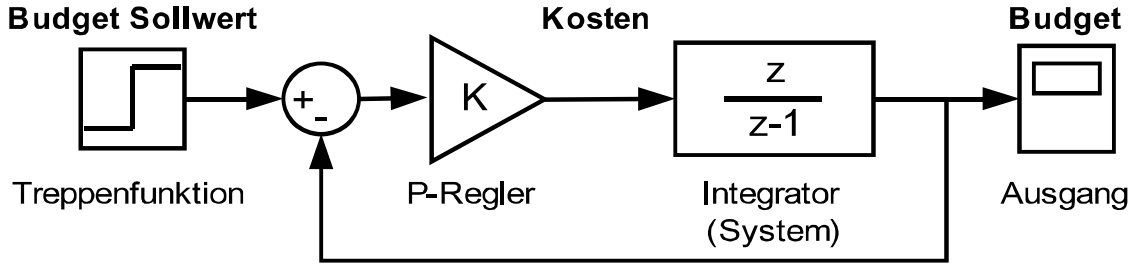


Abbildung 3.6: Modellierung der Wirkungen von Prozessen als Kosten auf ein Budget in einem Regelkreis

zu erreichen. Das Budget steht für entweder für einen oder für alle Prozesse gemeinsam zur Verfügung. Ein Regler gibt vor, durch welche Kosten in jedem Schritt das Budget zu erreichen ist. Der Budget-Sollwert im Zusammenhang mit dem Regler bestimmt die Stärke der Einflussnahme bei der Kooperation und der Kollaboration von Prozessen. Je größer der Sollwert, desto höher die möglichen Kosten pro Zeitschritt im gleichen Zeitraum.

Der Regelkreis ist in der Lage Störungen auszugleichen. Damit können auch unvorhergesehene Ereignisse und die Prozessdynamik berücksichtigt werden. Der Budgettransfer bei Prozesskollaboration kann von einer solchen Störung nicht unterschieden werden. Daher ist der Modellierungsansatz für beide Prozessorganisationsformen konsistent.

3.4.3 Mathematische Modellierung und Eigenschaften

Dieser Abschnitt formuliert unter Verwendung der vorangegangenen mathematischen Grundlagen die Modellbeschreibung und zeigt wichtige Eigenschaften des

Budget/Kosten-Modells. Die Regelstrecke ist ein Integrator, ein Element, welches die Kosten aufsummiert. Die Z Transformierte des Integrators ist $G(z) = \frac{z}{z-1}$. Als Regelungsgesetz wird ein einfacher P-Regler $P(z)$ mit Verstärkungsfaktor K eingesetzt. Damit ergibt sich nach Gleichung 3.12 die Transferfunktion des Regelkreises

$$F(z) = \frac{P(z)G(z)}{1 + P(z)G(z)} = \frac{K\frac{z}{z-1}}{1 + K\frac{z}{z-1}} = \frac{Kz}{z(K+1) - 1} \tag{3.19}$$

Es werden nun die Eigenschaften Stabilität, Schnelligkeit und Genauigkeit mit dem Instrumentarium aus Abschnitt 3.3.3 untersucht. Die Stabilitätsbedingung ist dabei die wichtigste Eigenschaft. Das Budget eines instabilen Rechnersystems ist nicht kontrollierbar. Ein Budget, zum Beispiel ein Datenpuffer, würde im Falle von Instabilität überfüllt werden und Daten würden verloren gehen.

Um die Stabilitätsbedingung aus Abschnitt 3.3.3.1 zu erfüllen, müssen die Pole der Transferfunktion innerhalb des Einheitskreises liegen. Dazu müssen die Nullstellen des Nenners der Transferfunktiongleichung 3.19 berechnet werden. Es ist

$$z(K+1) - 1 = 0 \Rightarrow z = \frac{1}{K+1} \tag{3.20}$$

Für $K > 0$ liegen die Pole innerhalb des Einheitskreises und das System ist BIBO stabil.

Die Genauigkeit der Regelung wird gemäß der Bedingung aus Abschnitt 3.3.3.2 abgeleitet. Für BIBO stabile Systeme ist der stationäre Regelungsfehler 0, genau dann wenn $F(1) = 1$ ist. Es wird die Transfergleichung 3.19 für $z = 1$ berechnet.

$$F(1) = \frac{K}{K + 1 - 1} = 1 \tag{3.21}$$

Das Budget/Kosten-Modell wird keinen stationären Regelungsfehler haben. Genauer: $\forall K > 0$ wird kein stationärer Regelungsfehler auftreten. Die Bedingung erklärt sich aus der Voraussetzung der BIBO Stabilität.

Zur Untersuchung der Schnelligkeit wird die Impulsfunktion im Zeitbereich der Transfergleichung 3.19 verwendet. Gemäß der Herleitung im Anhang B ist die Impulsfunktion

$$I(k) = \left(\frac{1}{K+1}\right)^k \frac{K}{K+1}$$

Für BIBO stabile Systeme wird die Schnelligkeit in Zeitschritten nach Gleichung 3.18 aus Abschnitt 3.3.3.3 berechnet. Die größte Polstelle ist $p = \frac{1}{K+1}$ und es ist $A = 1$:

$$k \approx \frac{\ln 0.02 - \ln A}{\ln p_1} = \frac{\ln 0.02}{\ln \frac{1}{K+1}} \tag{3.22}$$

Da BIBO Stabilität gefordert wird, muss $K > 0$ gefordert werden. Damit ist die Gleichung wohldefiniert. Das Modell wird umso schneller den Regelungsfehler kompensieren, je größer die Verstärkung K des P-Reglers ist.

Für die Reglersynthese ergeben sich nun zwei Schlussfolgerungen: 1.) Es muss $K > 0$ gelten. 2.) Die Wahl des Verstärkungsfaktors des Reglers hängt nur von der geforderten Schnelligkeit ab. Der Regler kann sehr einfach bestimmt werden. Für $K \to \infty$

kompensiert der Regelkreis den Fehler unendlich schnell. In realen Anwendungen ist die Reglerverstärkung nach oben begrenzt und kann einen Wert $K_{\max}$ nicht überschreiten. Für das schnellste, stabile Budget/Kosten Modell empfiehlt sich, $K_{\max}$ als Verstärkung des P-Reglers zu synthetisieren. Der Regelungsfehler wird am schnellsten nach $k \approx \frac{\ln 0.02}{\ln \frac{1}{K_{\max}+1}}$ Zeitschritten bis auf 2% kompensiert. Alternative Regelungsgesetze wie PI oder PID haben keinen Vorteil im Budget/Kosten Modell.

Die Graphen in den Abbildungen 3.7 und 3.8 zeigen das Regelungsverhalten des Budget/Kosten Modells für einen Sprung auf den Sollwert = 1 mit Verstärkungsfaktor $K = 0.5$ des P-Reglers. Nach 10 Zeitschritten ist der Budget-Sollwert zu 98% erreicht. In Abbildung 3.8 erfolgt bei Zeitschritt $t = 4$ eine Störung, die durch die Regelung kompensiert wird.

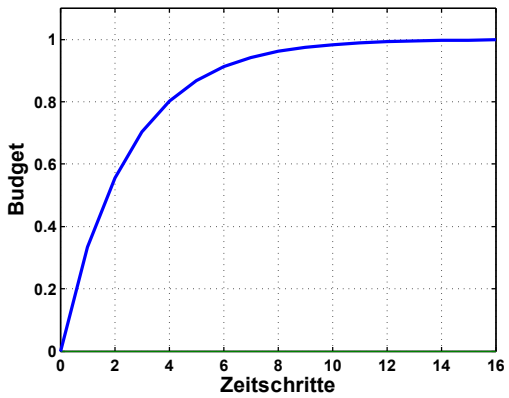


Abbildung 3.7: Regelungsverhalten des Budget/Kostenmodells für $K = 0.5$

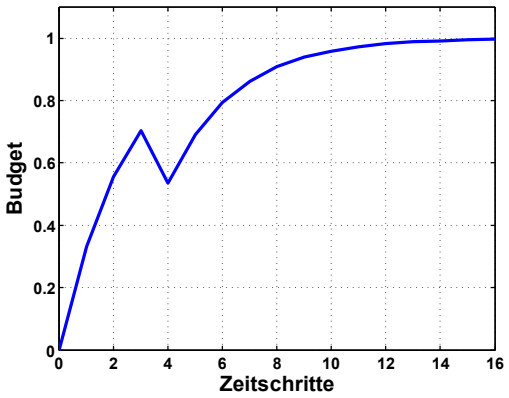


Abbildung 3.8: Regelungsverhalten des Budget/Kostenmodells bei Störung im Zeitschritt $t = 4$

Die folgende Tabelle 3.2 fasst die Eigenschaften des Budget/Kosten Modells zusammen und erklärt die Implikationen für die kooperative und kollaborative Prozessorganisation.

Eigenschaft	**Reglerbedingung**	**Implikationen für die Prozessorganisation**
Stabilität	$\forall K > 0 \Rightarrow$ stabil	Kooperations- und Kollaborationsverhalten von Prozessen wird garantiert erreicht
Genauigkeit	$\forall K > 0 \Rightarrow$ kein stationärer Regelungsfehler	Höchste Güte des Kooperations- und Kollaborationsverhalten wird erreicht
Schnelligkeit	$\forall K > 0$, je gößer K, desto schneller Fehlerkompensation	Kooperations- und Kollaborationsverhalten wird am schnellsten nach $k \approx \frac{\ln 0.02}{\ln \frac{1}{K_{\max}+1}}$ Zeitschritten erreicht

Tabelle 3.2: Eigenschaften des Budget/Kosten Modells für eingebettete ubiquitäre Rechnersysteme

3.4.4 Methodisches Vorgehen bei der Übertragung auf Prozessklassen

Für die Anwendung des Budget/Kosten Modells auf die verschiedenen Prozessklassen müssen die spezifischen Eigenschaften der Prozesse mit den Komponenten des Budget/Kosten-Modells assoziiert werden. So ist zum Beispiel zu klären, welche Eigenschaften der Prozesse durch das Budget repräsentiert werden. Im allgemeinen wird von einer zuvor aufgestellten Problemformulierung ausgegangen, die durch das Zusammenwirken oder die Zusammenarbeit von Prozessen gelöst werden soll. Dazu wird eine Vorgehensweise definiert.

1. **Identifikation des Budgets:** Als Budget wird eine Ressource oder eine Kombination von Ressourcen des ubiquitären Rechnersystems ausgewählt, die während der Prozessausführung beeinflusst wird. Beispiele sind Datenpuffer, Energievorräte, aber auch Zähler. Das Budget symbolisiert die aktuelle Dienstgüte der Prozessausführung. Es ist die zu regelnde Größe, die während der Laufzeit erfasst, zurückgekoppelt und mit dem Ziel verglichen wird.

2. **Bestimmung der Kosten:** Es ist zu bestimmen, auf welche Art eine Prozessklasse auf die Ressource einwirkt. Diese Wirkung wird durch die Kosten erfasst und verändert das Budget bei jeder Prozessausführung. Beispiele sind Änderungen der Füllstände von Puffern, des Energievorrates oder von Zählern. Kosten müssen nicht notwendigerweise direkt messbar sein.

3. **Festlegung des Ziels:** Das Ziel legt die zu erreichende Dienstgüte der Prozessausführung fest. Es dient als Kriterium, mit dem Kooperation und Kollaboration bewertet werden und gegebenenfalls bei Abweichung korrigierende Maßnahmen bei der Prozessausführung ergriffen werden. Beispiele für Ziele sind den Pufferfüllstand auf einem konstanten Niveau zu halten, den Energievorrat bis zu einem bestimmten Zeitpunkt aufgebraucht zu haben oder einen bestimmten Zählerwert zu erreichen. Das Ziel ist ein Budget-Sollwert oder der Verlauf einer Sollwert-Funktion.

3.5 Architektur rückgekoppelter ubiquitärer Rechnersysteme

Es werden die Komponenten der internen Organisationsstruktur für eine kooperative und kollaborative Prozessausführung vorgestellt und diskutiert.

3.5.1 Prozessklassen

Nach der Analyse in Kapitel 2 werden alle Prozesse in vier Klassen der periodischen, aperiodischen, echtzeitfähigen und verteilten Prozesse eingeordnet. Die Rückkopplungsmechanismen wirken auf einzelne Prozesse, werden aber klassenspezifisch entworfen, um die notwendige Abstraktion von den konkreten Prozessaufgaben zu erreichen.

Periodische und aperiodische Prozesse sind die grundlegenden Prozesse, die bei der Realisierung ubiquitärer Anwendungen verwendet werden. Periodische Prozesse führen Aufgaben in regelmäßigen Zeitabständen, der Periode, aus. Aperiodische Prozesse behandeln Unterbrechungen (engl. Interrupts) der Hardware der ubiquitären

Rechnersysteme und Ereignisse, die im Softwareablauf auftreten. Als Ergebnis können periodisch starre Prozesse flexibel verzweigen. Echtzeitprozesse werden in dieser Arbeit als periodische betrachtet, wobei die Zeitschranke mit der Ausführungsperiode gleich gesetzt wird. Veteilte Prozesse tauschen in periodischen Kommunikationsphasen Daten über die Kommunikationsschnittstellen der ubiquitären Rechnersysteme miteinander aus.

Gemein sind allen Prozessen Eigenschaften wie Ausführungsperiode, Ausführungszeitpunkt und Ausführungsdauer. Letztere ist dynamisch und ist im allgemeinen nicht im Voraus bekannt. Aperiodische Prozesse werden mit periodischen zur weiteren Verarbeitung der Ereignisdaten in Verbindung gesetzt. Daher kann man ihnen ebenfalls die genannten Eigenschaften zuordnen.

In der Konsequenz erlaubt dies, alle Prozessklassen in einem einheitlichen, *periodischen Ausführungsmodell* darzustellen. Der Regelungseingriff erfolgt jeweils nach der Ausführung eines Prozesses und vor der Ausführung des nächsten. Es können auch mehrere Regler involviert sein, wenn die Prozessausführung eine entsprechende Wirkung auf mehrere Budget-Ressourcen des Rechnersystems ausübt. Die Verankerung der Regelung zwischen den Prozessausführungen ermöglicht es, die Regelkreise als ein zusammengehöriges Softwaremodul in ein *gemeinsames Laufzeitsystem* der kooperativen und kollaborativen Prozessorganisation einzugliedern.

Eine wichtige Ressource ubiquitärer Rechnersysteme ist der Energievorrat. Alle Prozessklassen unterliegen einem Energiemanagement, um den Vorrat bestmöglich zu nutzen. Obwohl das Management keine eigenständige Prozessklasse bildet, wirken doch alle Prozesse auf den Energievorrat. Daher lässt sich auch das Energiemanagement als ein Regelkreis nach dem Budget/Kosten Modell darstellen. Es fügt sich gleichberechtigt in das Regelungsmodul des Laufzeitsystems ein.

3.5.2 Periodisches Ausführungsmodell

Alle Prozesse der zuvor besprochenen Klassen lassen sich in ein periodisches Ausführungsmodell einordnen. Es bildet sich eine Hierarchie wie in Abbildung 3.9 dargestellt. Dabei treibt das Laufzeitsystem regelmäßig periodische Prozesse an.

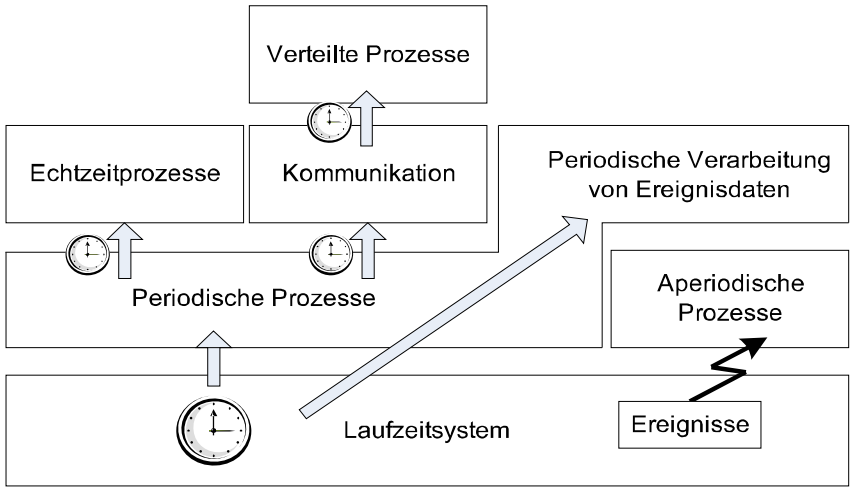


Abbildung 3.9: Hierarchie des periodischen Ausführungsmodells

In der gleichen Weise werden Echtzeitprozesse ausgeführt. Verteilte Prozesse erzeugen Nachrichten für periodisch stattfindende Kommunikationsphasen.

Die aperiodische Ereignisbehandlung unterbricht periodisch laufende Prozesse. Durch die Aufteilung in die Behandlung, den eigentlichen aperiodischen Prozess, und die periodische Verarbeitung der Ereignisdaten, wird die Unterbrechung minimal gehalten und in das periodische Ausführungsmodell eingegliedert.

3.5.3 Laufzeitsystem

Das periodische Ausführungsmodell führt zur Konsequenz, dass alle Prozessklassen in einem gemeinsamen Laufzeitsystem behandelt werden können. In dem Laufzeitsystem ist das Budget/Kosten Modell mit dem Regelkreis installiert. Prozesse werden

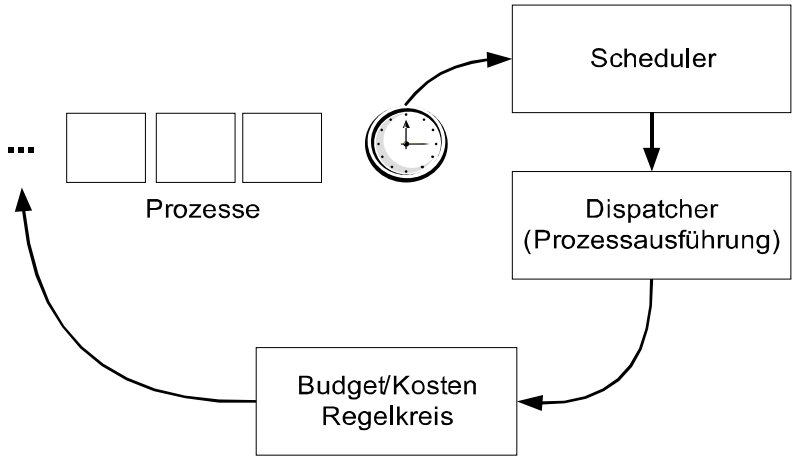


Abbildung 3.10: Laufzeitsystem

gemäß ihrer Periode dem Scheduler übergeben, der sie in eine Ausführungsreihenfolge bringt. Für Echtzeitprozesse kann zum Beispiel nach Deadline oder Raten-Prioritäten sortiert werden. Anschließend werden sie im Dispatcher zur Ausführung gebracht. Das Budget/Kosten Modell berechnet nach jeder Prozessausführung die angefallenen Kosten und modifiziert die Ablaufparameter. Die Parameter sind nicht allgemein festgelegt, sondern spezifisch für die Prozessklassen. Gemäß der definierten Vorgehensweise aus Abschnitt 3.4.4 ergibt sich eine prozessklassenspezifische Implementierung des Budget/Kosten Modells. Die spezifischen Implementierungen wirken jeweils als Störungen im jeweiligen Regelkreis aufeinander. Die analytischen Betrachtungen des Modells in Abschnitt 3.4.3 zeigen, dass Störungen bis zur Regelungsfehlerfreiheit kompensiert werden. Alle Implementierungen des Budget/Kosten Modells operieren daher im Laufzeitsystem orthogonal zueinander.

3.6 Implementierungsbetrachtungen

Da eingebettete ubiquitäre Rechnersysteme ressourcenbeschränkt sind, wird die technische Implementierungsfähigkeit des Ansatzes betrachtet. Es wird die Implementierung des Laufzeitsystems aus Abbildung 3.10 diskutiert. Messungen an einer Implementierung eines leichtgewichtigen Laufzeitsystems, die dem vorgestellten Entwurf folgen, wurden in [52] vorgenommen. Sie liefern quantitative Resultate zur Bestätigung der folgenden Betrachtungen.

3.6.1 Dispatcher

Der Dispatcher führt die Prozesse in der Reihenfolge aus, die vom Scheduler vorgegeben wird. Für jeden Prozess wird eine zuvor definierte Einsprungfunktion ausgeführt. Die Periodizität des Ausführungsmodells wird vom Dispatcher realisiert. Der Dispatcher kennt die Systemzeit und stößt gemäß der Periode die Übergabe von Prozessen an den Scheduler an. Bei der Übergabe der Prozesse zwischen Dispatcher und Scheduler werden jeweils Warteschlangen verwendet. Der Dispatcher durchläuft folgende Schritte:

Zum Ausführungszeitpunkt t_k des Prozesses P_i:

1. Akquiriere auszuführenden Prozess P_i von Scheduler
2. Führe Prozess P_i aus,
   - $t_k \rightarrow t_{k+1} = t_k + C_i$, C_i ist Ausführungsdauer von P_i
3. Berechne neuen Ausführungszeitpunkt für P_i nach Periode T_i: $t_{i,k+1} = t_k + C_i + T_i$
4. Behandle noch ausstehende Prozesse: $\forall P_j$ mit $t_{j,k} <= t_{k+1}$, übergebe P_j an Scheduler, Laufzeit V pro Vergleich mit ausstehenden Prozessen
5. Rufe Scheduler auf, Laufzeit $C_{\text{Sched.}}(\#j)$

Der Aufwand des Dispatchers berechnet sich nach diesem Ablauf zu
$O\left(C_i + jV + C_{\text{Sched.}}(\#j)\right)$.

3.6.1.1 Scheduler

Die Aufgabe des Schedulers besteht darin, die Prozesse in eine Ausführungsreihenfolge zu bringen. Die bekommt eine besondere Bedeutung, wenn aufgrund der Periode mehrere Prozesse zur Ausführung anstehen. Dazu wird ein Reihenfolgekriterium verwendet, welches die Prozesse untereinander priorisiert. Im einfachsten Fall wird die zeitliche Reihenfolge, mit der der Dispatcher Prozesse übergibt, verwendet. Diese bezeichnet man als eine First-In-First-Out (FIFO) Strategie. Andere Strategien sind beispielsweise Echtzeitstrategien, die die Ausführungsreihenfolge so bestimmen, dass geforderte Ausführungszeitschranken eingehalten werden.

Der Scheduler sortiert Prozesse in einer Warteschlange. Jeder neu hinzugefügte Prozess muss lediglich in die bestehende, bereits sortierte Liste, eingeordnet werden. Der Aufwand ist somit $O(n)$, wobei n die maximale Länge der Warteschlange bezeichnet.

Das Laufzeitsystem trennt klar zwischen den Komponenten und verwendet jeweils spezifische Kriterien für die Komponenten. Dies erlaubt die Austauschbarkeit des Schedulers, um andere Strategien zu unterstützen. Für das restliche System ist dieser Vorgang transparent.

3.6.2 Budget/Kosten Modell

Es werden bis zu vier Prozessklassen unterstützt. Zusätzlich soll auch der Energieverbrauch aller Prozesse organisiert werden. Es operieren somit bis zu 5 Regelkreise nebeneinander.

3.6.2.1 Budget

Das Budget ist die Regelgröße, die nach jeder Prozessausführung erfasst werden muss. Dies kann durch direkte Messung, zum Beispiel des Energievorrates, geschehen oder durch Berechnung, zum Beispiel der Wert eines Zählers. Der Aufwand der Erfassung mit anschließendem Vergleich mit dem Sollwert wird als konstant betrachtet: $O(C_{\text{Budg.}})$. Der Speicheraufwand zur Budgetfortschreibung ist ebenfalls konstant und wird zur Entwicklungszeit festgelegt.

3.6.2.2 Regler

Für den Einsatz auf eingebetteten ubiquitären Systemen ist der P-Regler aufgrund seiner Einfachheit vorteilhaft. Der Rechenaufwand ist somit konstant: $O(C_{\text{Reg.}})$. Der Systementwurf muss allerdings so gestaltet sein, dass das rückgekoppelte Regelkreissystem keine stationäre Regeldifferenz aufweist. Gemäß des Abschnitts 3.3.3.2 muss $F(1) = 1$ für das System gelten.

Als Alternative ist der PI-Regler geeignet. Auch dieser ist noch einfach zu implementieren. Er erfordert aber zusätzlich noch Ressourcen zur Fehlerintegration. Der PID-Regler ist noch leistungsfähiger, aber die diskrete Approximation des Fehleranstiegs kann für die betrachteten Plattformen ubiquitärer Rechnersysteme nur mit vergleichbarem großen Einsatz an Speicher- und Rechenressourcen akkurat durchgeführt werden.

Der Gesamtaufwand des Budget/Kosten Modells pro Prozessausführung ist $O(5[C_{\text{Budg.}} + C_{\text{Reg.}}])$.

3.7 Implikationen

Die Modellierung von Kooperation und Kollaboration als regelungstechnisches Problem in Rechnersystemen stellt eine neue Theorie der Prozessorganisation ubiquitärer Rechnersysteme zur Verfügung. Diese Theorie hat die in Tabelle 3.3 zusammengefassten Auszeichnungsmerkmale und Implikationen für die Gestaltung von Anwendungen auf ubiquitären Rechnersystemen.

3.8 Zusammenfassung

In diesem Kapitel wurden die Begriffe Kooperation und Kollaboration für ubiquitäre Rechnersysteme geklärt und mathematisch im Budget/Kosten Modell formuliert. Ein regelungstechnischer Ansatz der Umsetzung dieser Organisationskonzepte wurde erfolgreich in die Ausführungsumgebungen eingebetteter ubiquitärer Rechnersysteme integriert. Wichtige Instrumente zur Analyse des kooperativen und kollaborativen Prozessverhaltens hinsichtlich Stabilität, Genauigkeit und Schnelligkeit wurden bereitgestellt. Als Ergebnis steht eine ausformulierte Theorie der kooperativen und kollaborativen Prozessorganisation für eingebettete ubiquitäre Rechnersysteme zur Verfügung. Eine entscheidende Erkenntnis für die Implementierung ist, dass die Regler für die analysierten Prozessklassen unabhängig voneinander implementiert werden können und dabei trotzdem die durch die Analyse zugesicherten Eigenschaften erhalten bleiben. Mit der Methode aus Abschnitt 3.4.4 werden in den folgenden Kapiteln die Anwendungen der Theorie auf die analysierten Prozessklassen untersucht.

Merkmal	**Erklärung**	**Implikation**
Berücksichtigung der Prozesswirkung	Fokus auf die Wirkung von Prozessen aufeinander und auf Systemressourcen	Effiziente Ressourcennutzung durch berücksichtigende (kooperative) und gemeinschaftlich gekoppelte (kollaborative) Prozessausführung
Rückgekoppelte Prozessausführung und Regelung	Erfassung und Veränderung des Prozessverhaltens zum Erreichen einer Dienstgüte (QoS)	Einsatz von Rechnersystemen in Umgebungen mit unbekannten Parametern der Datenverarbeitung, z.B. Ausführungsdauer oder Ereignisfolge
Kosten/Budget Modell	Beschreibung der Einbettung von Prozessen in Regelkreise	Mathematische Instrumente des Entwurfs und der Analyse kooperativer und kollaborativer Prozessausführungen
Stabilitätsanalyse	Gewährleistet die Kontrollierbarkeit der Prozessorganisation	Grenzen der Ressourcennutzung (engl. resource bounds) werden garantiert, z.B. Puffer werden nicht überladen
Schnelligkeit	Schnelligkeit auf Veränderungen der Prozessorganisation zu reagieren; wird durch maximal mögliche Reglerverstärkung beschränkt	Adaptionsfähigkeit auf veränderte Laufzeitbedingungen, wie Ausführungsdauer, plötzliche Ereignissfolgen, hohes Nachrichtenaufkommen
Genauigkeit	Fähigkeit der Prozessorganisation, einen vorgegebenen Sollwert zu erreichen.	Ermöglicht die Abwägung zwischen Dienstgüte (QoS) und Geschwindigkeit der kooperativen und kollaborativen Prozessausführung
Einflussnahme	Sollwert und Reglersynthese erlauben, Geschwindigkeit und Genauigkeit zu bestimmen	Berücksichtigung von resource bounds und Anpassungsgeschwindigkeit der Anwendung
Realisierbarkeit	Mehrere Regelkreise operieren nebeneinander, dem Dispatcher nachgeschaltet	Modulares und leichtgewichtiges Laufzeitsystem für eingebettete ubiquitäre Rechnersysteme

Tabelle 3.3: Merkmale und Implikationen der Theorie der kooperativen und kollaborativen Prozessorganisation ubiquitärer Rechnersysteme

4. Periodische und Aperiodische Prozesse

In ubiquitären Rechnersystemen sind periodische und aperiodische Prozesse grundlegende Basisbausteine zur Realisierung von Anwendungen. Regelmäßig wiederkehrende Ausführungen von Prozessaufgaben verknüpft mit der Flexibilität von Ereignissen erlauben vielfältige Prozessabläufe.

Die Organisation periodischer Prozesse ist vor allem durch Attribute wie Periode, Startzeit und Jitter festgelegt. Sie misst sich weniger an einer erfolgreichen Datenverarbeitung. Dieses Kapitel wendet die neue Theorie der kooperativen und kollaborativen Prozessorganisation auf periodische und aperiodische Prozesse an. Ein besonderer Schwerpunkt liegt auf der *zeitnahen und erfolgreichen Datenverarbeitung* unter gegebenen Ressourcenbeschränkungen in ubiquitären Rechnersystemen. Dazu wird ein neues Systemmodell für diese Prozesse geschaffen. Es wird die bisherige periodische und aperiodische Prozessverarbeitung unterstützt, aber zusätzliche Einflussmöglichkeiten zur Regelung der Prozesse geschaffen.

Dieser Vorgehensweise wird eine Analyse des wissenschaftlichen Umfeldes vorangestellt. Zentrale Komponente ist dabei der Scheduler, der die Ablaufreihenfolge der Prozesse plant. Bisherige Scheduling-Ansätze werden klassifiziert und den Anforderungen ubiquitärer Rechnersysteme gegenübergestellt. Daraus leiten sich dann die Entwurfsparameter für das regelbare Systemmodell ab. Auf Basis des Budget/Kosten Modells aus Abschnitt 3.4 kann das Ablaufverhalten periodischer und aperiodischer Prozesse so gestaltet werden, dass kooperative und kollaborative Muster der Prozessausführung entstehen. Rückgekoppelte Organisationsprinzipien für kooperatives und kollaboratives Verhalten werden in den Implementierungen des Betriebssystems Particle OS und der Appliance AwarePen umgesetzt und die erzielten Ergebnisse bewertet.

4.1 Grundlagen und Einordnung ins wissenschaftliche Umfeld

Periodische und aperiodische Prozessausführung ist ein etabliertes Feld mit entsprechender Terminologie. In dieser Arbeit wird der Prozess als ein komplexes Kon-

strukt mit vielen verschiedenen Befehlen und einem komplexen Laufzeitverhalten verstanden. So enthält beispielweise ein Prozess zur Gewinnung verbesserter Lokationsinformationen [53], viele informationsverarbeitende Aktivitäten, die periphere Ultraschallsensoren des Rechnersystems und sogar weitere ubiquitäre Rechnersysteme einbeziehen. Andere Autoren [54] betrachten kleinere Ausführungseinheiten, die sogenannten Tasks, aus denen Prozesse aufgebaut sind.

Es ist zu erwähnen, dass die Begriffe in der Literatur nicht immer getrennt werden. So verwendet Buttazzo in [55] die Begriffe Task und Prozess als Synonyme. In den folgenden Abschnitten wird zunächst das Task-Modell diskutiert, bevor bei Einordnung ubiquitärer Rechnersysteme der Übergang zum Prozessbegriff erfolgt. Die getroffenen Aussagen über die Ablauforganisation (engl. Scheduling) von Tasks und Prozessen sind äquivalent.

4.1.1 Das Task-Modell

Ein Task ist definiert als ein Tupel $\tau_i = \{P_i, C_i, T_i, d_i, r_i\}$, wobei P_i die Funktion bezeichnet, die ausgeführt werden soll, C_i ist die Ausführungszeit des Task, T_i die Periode, d_i die Zeitschranke (engl. Deadline) für Echtzeittasks und r_i bezeichnet die Ressourcen, die durch den Task belegt werden. Tasks können periodisch oder aperiodisch definiert sein. Für aperiodische Tasks wird die Angabe der Periode T_i ausgelassen. Die wiederholte Ausführung eines Task i bildet eine Sequenz von Instanzen oder Jobs $J_i = j_{i,1}, j_{i,2}, \ldots$. Ein Job $j_{i,0}$ wird zu einer bestimmten Zeit, der Ankunfts- oder Aktivierungszeit $a(j_{i,0}) = a_{i,0}$, ausgeführt. Bei periodischer Ausführung werden die nachfolgenden Jobs zu den Zeitpunkten $a_{i,n} = nT_i$ ausgeführt. Zu jeder Aktivierungszeit gibt es auch eine Terminierungszeit $f_{i,n}$ zu der eine Jobausführung beendet wird. Ein Scheduler bestimmt eine Ordnung aller Jobs von allen Tasks.

Rangordnungen (engl. precedence constraints) werden zusätzlich in Form von gerichteten, azyklischen Graphen mit Tasks als Knoten und der Rangordnung als Kanten beschrieben. Ein Graph formuliert Forderungen wie die Ausführung eines Jobs von τ_i hat immer vor der Ausführung von τ_j zu erfolgen. Die Abbildung 4.1 stellt die Komponenten des Task-Modell in einer Übersicht dar.

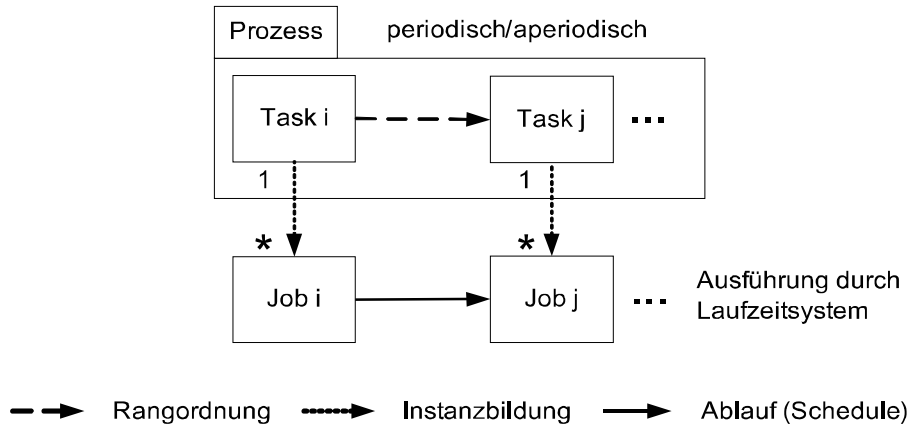


Abbildung 4.1: Übersicht der Komponenten des Task-Modells

4.1.2 Verfahren der Ablaufplanung

Die Ablaufplanung, engl. Scheduling, ist die zentrale Komponente der Organisation periodischer und aperiodischer Prozesse. Das Schedulingproblem ist nach [55] die Zuordnung von Jobs auf Ressourcen und Prozessoren unter Einhaltung der Rahmenbedingungen wie Rangordnungen und Zeitschranken. In dieser allgemeinen Form ist das Schedulingproblem NP-vollständig [56]. Durch Beschränkungen der Allgemeinheit kann jedoch die Komplexität reduziert werden. Dazu zählen Beschränkung auf 1-Prozessorsysteme, Verzicht auf Rangordnungen, Vorgabe fester Prioritäten, gleichzeitige Aktivierung von Tasks und weitere.

Es existiert eine große Vielfalt von Schedulingalgorithmen. Eine sehr umfassende und ausführliche Übersicht und Beschreibung bietet [57]. Für diese Arbeit wird eine Einteilung gemäß Abbildung 4.2 vorgenommen.

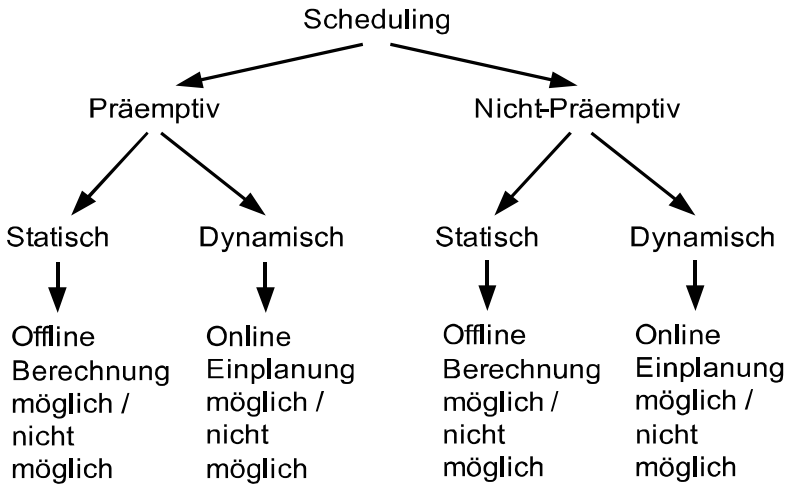


Abbildung 4.2: Einteilung von Schedulingalgorithmen

Es werden folgende Klassen unterschieden:

- **Präemptiv:** Ein im Moment ausgeführter Job kann unterbrochen werden, um den Prozessor einem anderen Job zuzuordnen. Die Unterbrechung kann durch den Schedulingalgorithmus ausgelöst worden sein. Beim Wechsel müssen genutzte Register und Speicherbereiche gesichert werden, um mit der Ausführung zu einem späteren Zeitpunkt fortzufahren.

- **Nicht-Präemptiv:** Wird ein Job vom Prozessor ausgeführt, so belegt er ihn bis der Job terminiert. Die nächsten Schedulingentscheidungen können erst nach Beendigung des Jobs getroffen werden.

- **Statisch:** Statische Schedulingalgorithmen bestimmen den Ablaufplan anhand von Parametern, die vor der Ausführung festgelegt wurden. Die Periode eines Tasks, kann ein Planungskriterium für einen statischen Algorithmus sein.

- **Dynamisch:** Die Ablaufplanung bestimmt sich nach Parametern, die sich zur Laufzeit verändern können. Zum Beispiel kann die Ausführungszeit eines Tasks bestimmend für die nächste Ausführungsreihenfolge der Jobs sein.

- **Online:** Die Online-Planung berücksichtigt auch das Hinzukommen oder Entfernen von Tasks. Die Task-Menge wird verändert, was eine Neubewertung und eventuell sogar einen Wechsel des bisherigen Schedulingalgorithmus notwendig macht. Dynamische Algorithmen können online-fähig sein.
- **Offline:** Offline Algorithmen berechnen den Ablaufplan vor der Ausführung von Tasks. Der Ablaufplan kann als Tabelle abgelegt werden und nur durch einen Dispatcher ausgeführt werden. Offline-Planung ist daher extrem leichtgewichtig zu implementieren. Sich dynamisch verkürzende Ausführungszeiten führen im Offline-Fall dann zu Leerlaufzeiten des Prozessors. Statische Algorithmen können auf Offline-Algorithmen eingeschränkt werden.

4.1.3 Kostenfunktionen

Die Leistungsfähigkeit von Schedulingalgorithmen wird durch Kostenfunktionen bewertet. Typische Beispiele sind die durchschnittliche Antwortzeit der Taskausführungen, oder die maximale Verspätung (engl. maximum lateness) bezüglich einer Zeitschranke. Für Echtzeitsysteme ist letztere besonders interessant. Ist die maximale Verspätung, $\max_i(f_i - d_i)$, für eine gegebene Task-Menge in jedem Fall negativ, dann kann garantiert werden, dass keine Zeitschranke verletzt wird.

Will man die Leistungsfähigkeit flexibler und ausdruckstärker bewerten, kann dies durch Nützlichkeitsfunktionen (engl. utility functions) geschehen [58][59]. Nützlichkeitsfunktionen sind komplexere Kostenfunktionen und ordnen Zeitpunkten der Taskausführung numerische Werte, sogenannte Nützlichkeiten oder Wichtigkeiten der Ausführung, zu. Im allgemeinen Fall können auch multivariate Funktionen mit zusätzlichen Faktoren, zum Beispiel Energieverbrauch, als Eingaben verwendet werden. Die Abbildungen 4.3 und 4.4 zeigen zwei Beispiele. Die Nützlichkeit von Prozessen, die aus Tasks zusammengesetzt sind, wird als kumulativer Wert mit $\sum_{\tau_i \in \text{Prozess}} v(t)$ angegeben. Für die Prozessausführung auf eingebetteten, ubiquitären Rechnersyste-

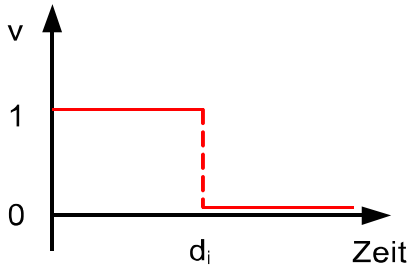


Abbildung 4.3: Nützlichkeitsfunktion für harte Echtzeittasks. Der Task verliert seine Nützlichkeit nach der Zeitschranke $t = d_i$.

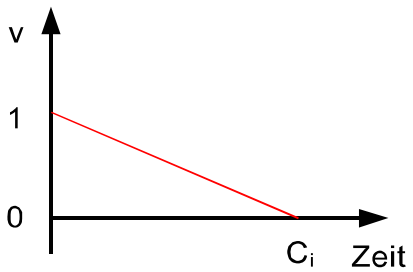


Abbildung 4.4: Nützlichkeitsfunktion für nicht-prämptiven Task, der andere Tasks verzögert.

men formulieren Nützlichkeitsfunktionen die Anforderung nach erfolgreicher Datenverarbeitung. So bewertet beispielsweise die Funktion in Abbildung 4.4 die Schnelligkeit einer nicht-präemptiven Datenverarbeitung. Je schneller, desto wertvoller für das System, da nachfolgende Prozesse zeitnäher starten können.

Kosten- oder Nützlichkeitsfunktionen sind keine Maßstäbe für Garantien. Man kann also von einer Kostenfunktion nicht notwendigerweise auf Ausführungsgarantien,

zum Beispiel Echtzeitverhalten, schließen. Es wird lediglich eine Bewertung der Ausführung vorgenommen. In jedem Anwendungsfall muss eine *für den Einsatzbereich geeignete Funktion* ausgewählt werden.

4.1.4 Schedulingverfahren für periodische Prozesse

Es werden Schedulingverfahren für die Anwendung auf Plattformen für eingebettete, ubiquitäre Rechnersysteme vorgestellt. Es wird sich dabei auf 1-Prozessorsysteme beschränkt. Die grundlegende Arbeitsweise aller Verfahren ist sehr ähnlich: Der Scheduler bestimmt durch eine interne Strategie Prioritäten der auszuführenden Jobs. Damit wird eine Reihenfolge hergestellt nach der die Jobs ausgeführt werden. Für ubiquitäre Rechnersysteme steht dabei das Verhalten hinsichtlich unbekannter und sich ändernder Laufzeitbedingungen und die effiziente Nutzung der beschränkten Ressourcen im Vordergrund.

Die Verfahren lassen sich in offene und geschlossene unterteilen. In offenen Verfahren verwendet die Schedulingstrategie aktuelle Parameter eines Tasks wie zum Beispiel seine Aktivierungszeit oder Periode. Geschlossene Verfahren bewerten die Ausführung mittels einer Kostenfunktion und führen das Ergebnis zu einem Regler zurück. Dieser ändert die Taskparameter so, dass der Scheduler eine Reihenfolge mit einem gewünschten Ausführungsverhalten erzeugt.

4.1.4.1 Offene Verfahren

Die Abbildung 4.5 zeigt die Komponenten offener Schedulingverfahren. Die Tasks

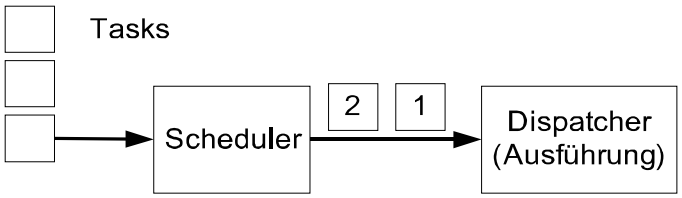


Abbildung 4.5: Schema von offenen Schedulingverfahren

werden durch die Strategie des Schedulers in eine Prioritätsreihenfolge gebracht und dann in dieser Reihenfolge ausgeführt. Für den Einsatz auf eingebetteten, ubiquitären Rechnersystemen werden die folgenden Priorisierungsstrategien untersucht:

Einfache Priorisierungsschemata Das einfachste Verfahren besteht darin, die Jobs in der Reihenfolge auszuführen, in der sie aktiviert wurden. Die Priorität der Jobausführung ergibt sich direkt aus dem Aktivierungszeitpunkt und sortiert die Jobs in einer FIFO Schlange. Die Jobs werden nicht-präemptiv ausgeführt. Die FIFO Strategie ist für ressourcenbeschränkte Geräte besonders geeignet, da sie sich leichtgewichtig als einfache Warteschlange implementieren lässt. Das eingebettete Betriebsystem TinyOS [13] verwendet einen FIFO Scheduler. Allerdings lassen sich weder Zeit- noch Ausführungsgarantien geben und es besteht die Gefahr von verhungernden Tasks [60].

Jobs statische Prioritäten zuzuordnen kann zeitkritische Jobs bevorzugen. Die Reihenfolge der Jobausführung ist vorgegeben und lässt sich einfach implementieren. Das AwareCon System [61] implementiert ein präemptives 2-Task-

System aus einen Kommunikationstask und einem Task für die Sensordatenakquise und Datenverarbeitung. Die Jobs des Kommunikationstasks erhalten die höchste Priorität und unterbrechen die Jobs des zweiten Arbeitstasks.

Echtzeit In Echtzeitverfahren wird die Einhaltung von Zeitschranken der Tasks zugesichert. Präemptive Verfahren wie Rate Monotonic Scheduling (RMS) oder Earliest Deadline First (EDF) [62] bestimmen die Jobprioritäten statisch respektive dynamisch während der Laufzeit, so dass der entstehende Ablaufplan die Zeitschranken garantiert einhält. EDF kann dabei den Prozessor auch maximal auslasten und damit die höchste Effizienz erreichen. Nicht-präemptive Alternativen unterliegen starken Beschränkungen der Effizienz oder der Wahl der Ausführungsperioden. Mikrocontroller-Betriebssysteme wie AmbientRT [63], FreeRTOS [64], XMK [65] implementieren präemptives Echtzeitscheduling.

Echtzeit kann nur zugesichert werden, wenn die Ausführungsdauer aller Tasks garantiert wird. Daher wird die maximale Ausführungsdauer, engl. Worst-Case-Execution-Time (WCET), verwendet. Sie stellt eine potentielle Fehlerquelle dar, wenn sie vom Entwickler falsch geschätzt wurde. Auf Mikrocontrollern ist eine zuverlässige Unterbrechung und Wiederaufnahme von Jobs schwierig zu realisieren, da keine Schutzmechanismen für Tasks angeboten werden. Im allgemeinen Fall wird ein Zeitgeberinterrupt verwendet, damit der Scheduler die Kontrolle zurück erlangt. Jedoch zeigte [66], dass es unmöglich ist zu garantieren, dass ein Task den Interrupt nicht (versehentlich) abschaltet.

In einer heterogenen Systemlandschaft ubiquitärer Rechnersysteme mit austauschbaren Prozessen ist es daher sehr aufwendig, die Voraussetzungen zum Einsatz von Echtzeitverfahren zu erfüllen. Eine zu hohe WCET wird außerdem Rechnerressourcen einplanen, die im Betrieb ungenutzt bleiben.

Kooperatives Scheduling Bei kooperativen Verfahren geben Tasks die Kontrolle über den Prozessor ab und der Dispatcher wählt den nächsten Task aus. Damit ist das Schedulingverfahren zum Teil implizit in der Anwendung enthalten und kann nicht klar vom Laufzeitsystem getrennt werden. Auf der anderen Seite kann mit geringem Aufwand ein präemptives Verhalten mit dynamischer Wahl des nächsten Jobs implementiert werden. Eingebettete Betriebssysteme wie SOS [67], Contiki [68] und BTNut [15] implementieren diese Verfahren für Sensorknotenplattformen. Kooperatives Scheduling verlangt, dass die Ablaufplanentscheidungen offline, d.h. vor der ersten Aktivierung von Tasks, vom Entwickler getroffen werden. Dazu ist detailliertes Wissen über die gesamte Applikation und ihren Ablauf zu jedem Zeitpunkt notwendig. Für online austauschbare Prozesse kann das nicht effizient geleistet werden. Kooperation im Sinne dieser Arbeit sollte nicht mit dieser Strategie verwechselt werden.

Die Tabelle 4.1 bewertet die Priorisierungsstrategien. Die offenen Verfahren zeigen Nachteile bei der Behandlung der Dynamik der Ausführung von Tasks. Gemäß der Analyse in Kapitel 2 ist die Berücksichtigung und Behandlung der Dynamik jedoch entscheidend für ubiquitäre Anwendungen. Die Dynamik wird entweder durch obere Abschätzungen der Laufzeit effektlos oder gar nicht behandelt (FIFO). Kooperative Verfahren können nicht für beliebige Task-Mengen verallgemeinert werden. Geschlossene Verfahren bieten durch Rückkopplung eine Struktur an, Ausführungsdynamiken einheitlich zu behandeln.

	Priorisierungs-strategie	Bewertung für ubiquitäre Rechnersysteme
Einfach	FIFO, statisch	keine (FIFO) bzw. feste (statisch) Priorisierung durch Entwickler, Dynamik der Taskausführung wird nicht direkt behandelt, sehr einfach zu implementieren
Echtzeit	EDF, RMS	garantiert die Einhaltung von Zeitschranken, WCET ist schwierig zu bestimmen, zu hohe WCET führt zu schlechter Ressourcennutzung
Kooperativ	Schedulingpunkte, in der Applikation integriert	erfordert tiefe Kenntnisse des Ablaufverhaltens aller Applikationsteile, dynamisches Verhalten wird berücksichtigt, anfällig gegen Änderungen in der Zusammensetzung der Task-Menge, einfach zu implementieren

Tabelle 4.1: Bewertung der Strategien für Scheduler eingebetteter Systeme. Die kooperative Strategie unterscheidet sich vom Kooperationsbegriff wie er in dieser Arbeit verwendet wird und sollte nicht verwechselt werden.

4.1.4.2 Geschlossene Verfahren

Diese Schedulingverfahren verwenden einen Regelmechanismus zur feingranularen Qualitätsregelung (QoS) von Anwendungen durch die Ablaufplanung. Der Einsatz der geschlossenen Verfahren verspricht Flexibilität und Zuverlässigkeit in Einsatzbereichen, in denen Parameter der Tasks zum Teil unbekannt und zusätzlich dynamisch sind. Die Abbildung 4.6 zeigt die Komponenten in einer schematischen Übersicht. Die Ausführung der Jobs in der ermittelten Reihenfolge wird durch eine Kostenfunk-

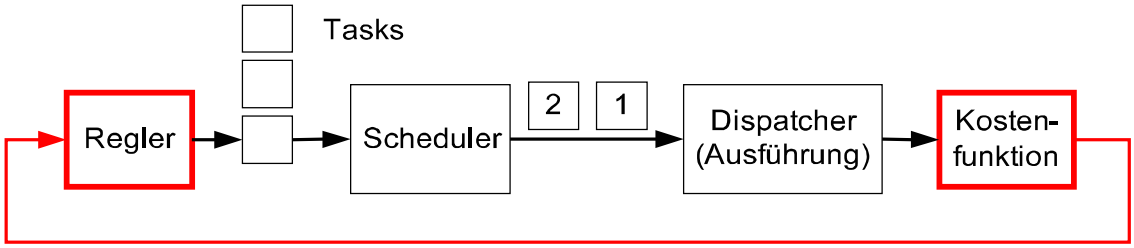


Abbildung 4.6: Schema von geschlossenen Schedulingverfahren mit Rückkopplung

tion bewertet. Diese Quantifizierung wird mit der Zielgröße verglichen. Ein Regler verändert die Taskparameter, um die Schedulingstrategie zu beeinflussen. Detailliertes Wissen über die eingesetzte Schedulingstrategie ist unbedingt notwendig. Zum Einsatz kommen die Strategien aus Tabelle 4.1. Es ist zu bemerken, dass im allgemeinen nicht die Schedulingstrategie verändert wird. Sie ist fest im Scheduler implementiert. Der Erfolg der Einflussnahme zeigt sich erst nach der Ausführung und Neubewertung durch die Kostenfunktion.

Einsatzbereiche sind primär echtzeitfähige Multimedia-Applikationen, die auf vernetzten Plattformen wie Settop Boxen laufen, und unvorhersehbaren Verzögerungszeit- und Bandbreitenschwankungen und inhaltsabhängigen Verarbeitungszeiten unterliegen. Optimale Verfahren wie [69], die auch Rangordnungen der Tasks berücksichtigen, und regelungstechnische Verfahren [70] bewerten mittels Kostenfunktionen

Bildqualität und Zeitschrankenverletzungen und modifizieren Taskparameter und Prioritäten. Die Autoren in [71] und [48] erweitern den Rahmen und konstruieren ein QoS-basiertes einheitliches Rahmenwerk für Echtzeitprozesse.

Bei den vorgestellten QoS-basierten Verfahren wird durch die Regelung die Ausführungsdauer der Jobs modifiziert. Voraussetzung ist, dass Tasks sich in einen notwendigen Teil und optionalen Teil gliedern lassen. Letzterer verbessert durch seine Ausführung das Ergebnis. Dies prädestiniert den Ansatz für Multimedia-Anwendungen. Für eingebettete, ubiquitäre Rechnersysteme kann dieses Task-Modell nicht angewendet werden.

Die Rückkopplungsmechanismen der geschlossenen Verfahren können als Grundlage für Kooperations- und Kollaborationprinzipien dieser Arbeit dienen. Es muss jedoch auch eine geeignete Priorisierungsstrategie des Scheduler geben, die für das Laufzeitsystem auf ubiquitären Rechnersystemen beeinflussbar ist.

4.1.5 Scheduling von nicht-periodischen Prozessen

Nicht-periodische Prozesse verarbeiten Daten von Ereignissen, die während des Betriebs von Rechnersystemen auftreten. Es wird zwischen sporadischen und aperiodischen Tasks unterschieden.

Sporadische Tasks haben unbekannte Aktivierungszeitpunkte. Es lässt sich eine kleinste Zeitspanne zwischen zwei Ausführungen definieren. Zusätzlich ist die größte Ausführungszeit aller Jobs $WCET_i = \max_k C_{i,k}^{\text{sporadic}}$ eines sporadischen Tasks bekannt. Beide Informationen sind bereits vor der ersten Ausführung des ersten Jobs eines sporadischen Tasks bekannt.

Aperiodische Tasks haben unbekannte Aktivierungszeitpunkte. Es gibt keine Beschränkung hinsichtlicher einer kleinsten Zeitspanne zwischen zwei Ausführungen. Zusätzlich ist die größte Ausführungszeit aller Jobs $WCET_i = \max_k C_{i,k}^{\text{aperiod.}}$ aperiodischer Tasks bekannt.

Sporadische Tasks können als periodische Tasks mit einer Periode gleich der minimalen Ankunftszeit zwischen zwei Ausführungen aufgefasst werden [72]. Damit können sie unmittelbar wie periodische Tasks mit Echtzeitgarantien eingeplant werden.

In dieser Arbeit ist der Fokus auf der Verbindung von aperiodischen Tasks mit dem periodischen Ausführungssystem. Entscheidend sind zwei Entwurfskriterien:

1. Die Störung periodischer Tasks soll gering sein.
2. Die Antwortzeit bis zur Verarbeitung aperiodischer Tasks soll gering sein.

Die einfachste Methode ist, aperiodische Ereignisse im Hintergrundprozess zu verarbeiten. Dieser Ansatz garantiert, dass periodische Tasks bevorzugt werden, aperiodische aber mit der kleinsten Priorität bearbeitet werden. Das führt zu langen Antwortzeiten. Eine effektive Methode ist es, die Verarbeitung von aperiodischen Tasks durch einen periodischen Server wie in Abbildung 4.7 in das periodische Ausführungssystem einzuordnen. Der Vorteil ist die Einfachheit des Ansatzes. Der Server hat eine Ausführungsdauer, auch Kapazität genannt, in der die Verarbeitung

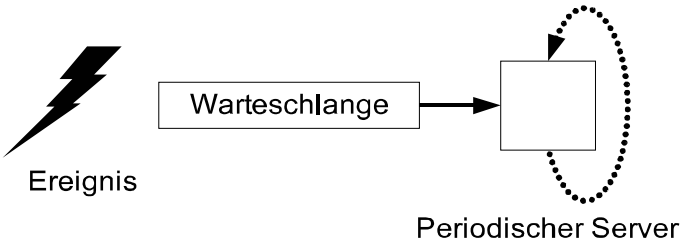


Abbildung 4.7: Schematische Darstellung eines Servers zu Behandlung aperiodischer Ereignisse

erfolgt. Verzögerungen periodischer Tasks können so begrenzt werden. Buttazzo erstellt in [55] eine umfassende Übersicht der verschiedenen Server-Methoden. Es ist bewiesen, dass Optimalität hinsichtlicht der Minimierung der Antwortzeit aperiodischer Tasks mit dynamischen Servern erreicht werden kann [73]. Für die statische Ablaufplanung kann kein Optimum gefunden werden [74]. In Überlastsituationen, in denen aperiodische Tasks die Serverkapazität überschreiten, werden Tasks in der Einplanung ausgelassen. Antwortzeiten können nicht mehr garantiert werden.

4.1.6 Anforderungen eingebetteter, ubiquitärer Rechnersysteme

Ubiquitäre Rechnersysteme führen Prozesse aus, die aus einer Rangordnung von Tasks bestehen. In der Laufzeitumgebung werden Prozesse als Black-Box betrachtet, deren Ausführung nicht abgebrochen oder unterbrochen werden kann. Aus dieser Perspektive erfordert die Ablaufplanung von Prozessen ein nicht-präemptives Schedulingverfahren, wobei *ein Prozess als ein großer komplexer Task* repräsentiert wird.

Erschwerend für die Ablaufplanung ist der Einsatz von ubiquitären Rechnersystemen in unbekannten Szenarien. Dort kann das Zeitverhalten der Prozesse im Vorraus nur ungenügend angegeben werden. Gründe dafür sind datenabhängige Ausführungszeiten und Überlastsituationen aufgrund von gleichzeitiger periodischer und aperiodischer Datenverarbeitung, die den Systemablauf unvorhersehbar verzögern. In [52] werden die Schedulingverfahren FIFO, RMS und EDF für Prozesse mit unbekannten Laufzeitverhalten untersucht. Die Autoren berichten von Leerlaufzeiten, ineffizienter Ressourcennutzung, Verhungerung von Prozessen und dem Domino-Effekt. In dynamischen Schedulingverfahren kann es zu wiederholten Verzögerungen beim Start der Jobausführung kommen. Ereignisse werden nicht verarbeitet, was dominoartig zum Verhungern weiterer Prozesse führt. Es werden die folgenden Anforderungen und Rahmenbedingungen für ubiquitäre Rechnersysteme abgeleitet:

Technische Rahmenbedingungen resultieren aus dem Aufbau ubiquitärer Rechnersysteme und dem unbekannten Zeitverhalten der Prozesse. Für die Ablaufplanung ergeben sich folgende Rahmenbedingungen:

- Ubiquitäre Rechnersysteme sind 1-Prozessorsysteme.
- Prozesse sind komplexe, ununterbrechbare Tasks mit endlichen Ausführungszeiten. Es kann eine WCET angegeben werden.
- Ablaufplanung erfordert eine dynamische, nicht-präemptive Schedulingstrategie mit Online-Einplanbarkeit neuer Prozesse.

Einschränkungen sind die Folge der in Kapitel 2 analysierten Art und Weise der Datenverarbeitung auf ubiquitären Rechnersystemen

- Prozesse können nur aus internen Gründen abbrechen, aber nicht vom Laufzeitsystem unterbrochen oder abgebrochen werden
- Ein Task-Modell mit Einhaltung spezifischer Ausführungszeiten von Prozessen kann nicht realisiert werden. Die QoS-orientierten Schedulingverfahren aus Abschnitt 4.1.4.2 sind nicht anwendbar.
- Das Auftreten von Überlastsituation bei aperiodischer und periodischer Datenverarbeitung kann nicht ausgeschlossen werden.

Entwurfsparameter stellen die Forderungen auf, die das Systemmodell für ubiquitäre Rechnersysteme charakterisieren. Im Vordergrund steht die effiziente Behandlung von nicht-präemptiven Prozessen mit unbekanntem Zeitverhalten.

- Nachregelung der Ausführungsreihenfolge der Prozesse durch geschlossene Schedulingverfahren.
- Finden einer geeigneten Nützlichkeitsfunktion zur Erfolgsmessung der Datenverarbeitung. Der Schwerpunkt liegt auf der Quantifzierung der zeitnahen Datenverarbeitung.
- Verwendung eines Prozessdesign mit Unterstützung der zeitnahen Datenverarbeitung
- Behandlung von Überlastsituationen in der Datenverarbeitung und Verhinderung des Domino-Effekts
- Einsatz rückgekoppelter Kooperation- und Kollaborationsmechanismen, die die Priorisierungsstrategie des Schedulers unabhängig von der Ausführungsdauer der Prozesse beeinflussen.

4.2 Kooperation und Kollaboration für die periodische und aperiodische Prozessorganisation

Ziel der Kooperation und Kollaboration von Prozessen ist es, eine zeitnahe Datenverarbeitung zu erreichen. Der Regelungsmechanismus soll den Scheduler so beeinflussen, dass die entstehende Ausführungsreihenfolge das Zusammenwirken und die Zusammenarbeit von Prozessen hinsichtlich der zeitnahen Datenverarbeitung unterstützt. Das Rückkopplungsprinzip erlaubt es, Störungen zu identifizieren und zu kompensieren, so dass der Einsatzbereich ubiquitärer Rechnersysteme bestmöglich unterstützt wird. Die Abbildung 4.8 zeigt die Komponenten des Ansatzes.

4.3 Serviceorientierter Systementwurf

Ein Service ist eine neue, einheitliche Abstraktion aus der alle Prozesse für das zugrundliegende periodische Ausführungsmodell ubiquitärer Rechnersysteme gebildet werden. Services sind eine Erweiterung von Tasks. Es wird eine Funktionalität, zum Beispiel die Sensordatenakquise, gekapselt und mit einem Ein- und Ausgabe-Puffer verbunden. So entsteht eine uniforme Schnittstelle. Services laufen nicht-präemptiv und in ihrer Ausführung unabhängig voneinander ab. Es gibt keinen Service, der

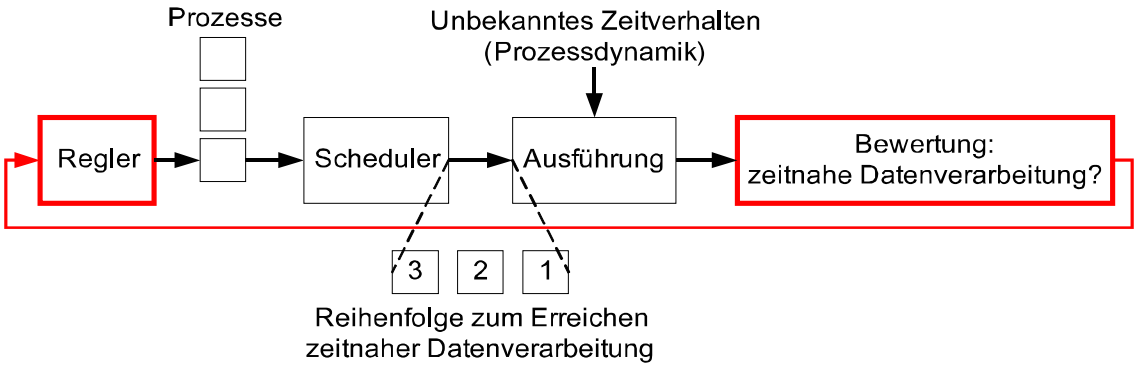


Abbildung 4.8: Einsatz von Kooperation und Kollaboration zur Ablaufplanung

einen anderen aufruft. Eine Serviceausführung wird immer von einem darunter liegenden Laufzeitsystem angetrieben. Der serviceorientierte Systementwurf eröffnet Eingreifpunkte für die Regelung der Ablaufplanung. Er ermöglicht und unterstützt die darunterliegenden rückgekoppelten Kooperations- und Kollaborationsmechanismen zum Erreichen einer erfolgreichen Datenverarbeitung. Der Begriff *Service* wird absichtlich anstelle der Bezeichnung *Dienst* verwendet, um zwei Entwurfskriterien zu verdeutlichen:

1. Unabhängigkeit: Services sind unabhängige Einheiten und kapseln einheitlich komplexe Funktionalitäten.

2. Ausführbarkeit: Services zu benutzen bedeutet sie auszuführen, d.h. ähnlich einer Funktion findet eine Verzweigung der Ausführung in den Service hinein statt. Dies geht über die reine Nachrichtenübergabe hinaus.

4.3.1 Service-Modell

Der schematische Aufbau eines Service ist in Abbildung 4.9 gezeigt. Zentral ist die

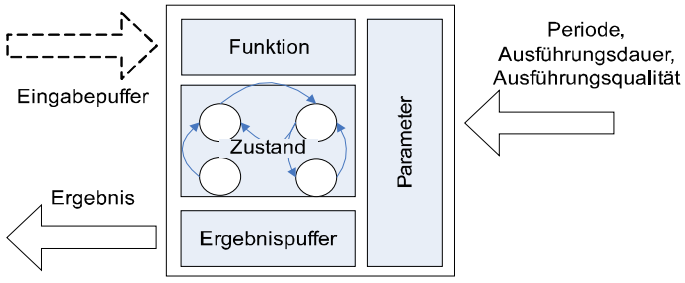


Abbildung 4.9: Eine schematische Sicht eines Service - unabhängige und einheitliche Kapselung einer Funktionalität eines ubiquitären Rechnersystems

Servicefunktion, über die das Laufzeitsystem den Service zur Ausführung bringt. Servicefunktionen werden immer periodisch und nicht-präemptiv ausgeführt. Dabei konsumiert der Service Eingabedaten über einen assoziierten Puffer, führt die Funktion aus und schreibt das Ergebnis in den Ergebnispuffer. Im Gegensatz zum Eingabepuffer steht der Ergebnispuffer unter voller Kontrolle des Service. Ein leerer Puffer zeigt an, dass die Ausführung nicht erfolgreich war.

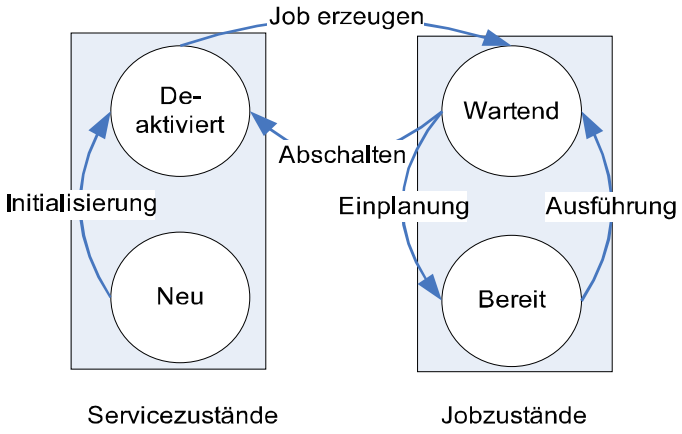


Abbildung 4.10: Zustände und Transitionen im Lebenszyklus von Service und Jobs

Ein Service und seine Jobs durchlaufen im Lebenszyklus verschiedene Zustände wie in Abbildung 4.10 gezeigt. Servicezustände werden eingenommen, wenn Services nicht ausführbar sind. Zur Ausführung werden Jobs erzeugt, die vom Laufzeitsystem organisiert werden. Dabei wird zwischen Jobzuständen gewechselt.

Servicezustände: Im Zustand NEU wird ein Service im System registriert. Die Laufzeitparameter werden initialisiert und der Dateneingabepuffer wird assoziiert. Im Anschluss wechselt er in den Zustand DEAKTIVIERT.

Jobzustände: Zur Ausführung der Services, werden jeweils Jobs erzeugt, die im Zustand WARTEND auf die Einplanung warten. Hat der Scheduler die Ausführungsreihenfolge festgelegt, dann wechseln sie in den Zustand BEREIT. Der Dispatcher bringt die Jobs nacheinander zur Ausführung. Nach der Ausführung werden implizit neue Jobs erzeugt, die den Zustand WARTEND einnehmen.

Wird ein Service abgeschaltet, werden die zugehörigen Jobs entfernt und dem Laufzeitsystem die Kontrolle entzogen. Es wird der Servicezustand DEAKTIVIERT erreicht.

Ein Service kann vielfältig konfiguriert werden. Darunter sind reguläre Taskparameter wie Ausführungsperiode, Zeitschranken und Ausführungsdauer. Zusätzliche erweiterte Parameter beschreiben die Ausführungsqualität oder verweisen auf spezifische Kostenfunktionen. Das Laufzeitsystem mit den rückgekoppelten Kooperations- und Kollaborationsmechanismen verwendet diese Parameter für Entscheidungen zur Ablaufplanung.

Formal sind Services eine Erweiterung des Task-Modells aus Abschnitt 4.1.1. Ein Service ist ein Tupel $s_i = \{P_i, In_i, Out_i, Par_i\}$, wobei P_i die Servicefunktion ist, In_i ist der Eingabepuffer, Out_i der Ausgabepuffer und Par_i die Menge der Parameter. Eine Ressourcenbeschreibung wie im Task-Modell wird ausgelassen, da die nicht-präemptive Ausführung Ressourcenkonflikte vermeidet. Jedoch können Konflikte auftreten, wenn Ressourcen zwischen verschiedenen Serviceausführungen noch mit anderen Services geteilt werden. Diese müssen applikationsspezifisch unter Verwendung von Anzeigern an den Ressourcen gelöst werden.

4.3.2 Prozessausführungsmodell Servicegraph

Services sind in ihrer Ausführung unabhängig voneinander, d.h. kein Service kann einen anderen aufrufen. Mittels eines Graphen werden die Services in Beziehung gesetzt und bilden Prozesse. Das Laufzeitsystem bestimmt die Ausführungsreihenfolge und kann regelnd eingreifen.

In diesem Abschnitt werden die Prozesse als Servicegraphen formalisiert. Ein Servicegraph ist ein gerichteter, azyklischer Graph $G = (S, B, E)$ mit Services $S = \{s_1, .., s_n\}$, Puffer (engl. Buffers) $B = \{b_1, .., b_m\}$ und Kanten (engl. edges) $E = \{e_1, .., e_l\}$.

Definition 4.1 (Servicegraph)**.** *Ein Servicegraph ist eine Inzidenzstruktur $\mathcal{G} = \langle S, B, E, \mathcal{I} \rangle$ mit der Inzidenzrelation $\mathcal{I} \subseteq S \times B \times E$. Die Inzidenzrelation setzt S, B und E über einen $n \times m \times l$ Inzidenzkubus Γ wie folgt in Beziehung:*

1. *Eine Kante $e_k \in E$ verbindet gerichtet (positiv inzident) genau einen Service $s_i \in S$ mit einem Buffer $b_j \in B$, also $s_i \xrightarrow{e_k} b_j$. D.h. für e_{k_1}, e_{k_2} gilt, wenn $\gamma_{ijk_1} = \gamma_{ijk_2} = 1$, dann $k_1 = k_2$. Zur Vereinfachung soll die Beziehung zwischen Service und Buffer im folgenden über eine $n \times m$ Inzidenzmatrix H bezeichnet werden mit $h_{ij} = 1$ für die Beziehung $s_i \rightarrow b_j$ und $h_{ij} = -1$ für die Beziehung $b_j \rightarrow s_i$.*
2. *Jeder Buffer ist nur mit genau einem Service als Datenquelle verbunden. Für $s_{i_1}, s_{i_2} \in S, b_j \in B$ gilt, wenn $h_{i_1 j} = 1$ und $h_{i_2 j} = 1$, dann $i_1 = i_2$.*
3. *Mehrere Services können als Datensenke mit ein und dem selben Buffer verbunden sein. D.h. $\exists j$ und $\{s_i | i \in \{1..n\}\} \subseteq S$ mit $b_j \rightarrow s_i$ für das gilt $h_{ij} = -1$.*

Der Servicegraph ist das Pendant zu den Rangordnungen; engl. precedence constraints; im Task-Modell aus Abschnitt 4.1.1. Dieses Design hat den Vorteil, dass eine Applikation in Teilstücke - die Pfade im Graph - zerlegt werden kann, die eine *abgeschlossene, service-orientierte Funktionalität* der Gesamtapplikation präsentieren. Diese Teile werden auch Prozesse genannt. Mittels des Servicegraphen wird der Prozessbegriff gebildet.

Definition 4.2 (Prozesse)**.** *Die Relation $s_i \xrightarrow{*} s_j$ mit $Pred(s_i) = \emptyset$ und $Succ(s_j) = \emptyset$, die mit der Inzidenzstruktur $\mathcal{G}$ wie in Definition 4.1 gebildet werden kann, bildet eine Menge von Pfaden $\mathcal{P}$ von der Wurzel s_i bis zu einem Blatt s_j in dem Graphen. Die Elemente der Menge $\mathcal{P}$ werden als Prozesse p_k bezeichnet.*

Die Abbildung 4.11 zeigt einen Beispielgraphen mit den Services $s_1, \ldots, s_5$, die vier Prozesse bilden. Blatt-Services bezeichnen das Ende des Prozesses und werden auch als Datensenke bezeichnet. Nach ihrer Stellung im Graph werden zwei Arten von Services unterschieden:

Periodische Services bilden die Wurzel des Graphen. Der Start periodischer Services initiiert die Ausführung der Prozesspfade.

Datenorientierte Services sind alle Services außer der Wurzel. Sie werden aktiviert, sobald Eingangsdaten vorhanden sind. Diese können von periodischen und anderen datenorientierten Services bereitgestellt werden. Falls keine Eingangsdaten vorliegen, bricht der Service ab. Obwohl sie keine Periode haben, erscheinen sie bei Ausführung im Kontext des Servicegraphen wie periodische.

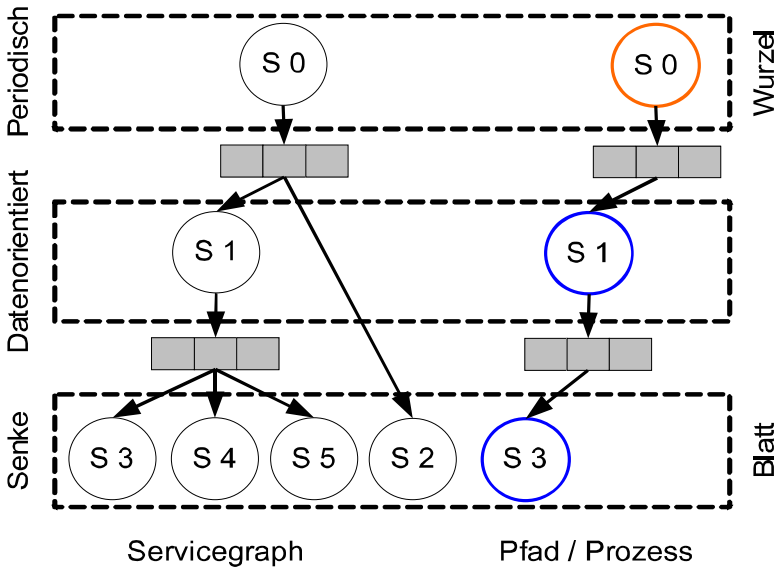


Abbildung 4.11: Servicegraph und ein Prozess als Pfad im Graph

4.3.2.1 Prozessausführung und Scheduling

Das prozessorientierte Applikationsdesign mit den Services als Knoten erlaubt die Feststellung von einzelnen Prozesspfaden, die während der Laufzeit abzulaufen sind. Diese können bereits offline mittels einer Tiefensuche im Graphen identifiziert werden und zur Laufzeit als Prozesse dem Scheduler übergeben werden.

Die Aufgabe des Schedulers ist es, zur Laufzeit eine Reihenfolge der auszuführenden Prozesse zu wählen. Im Servicegraphen besitzen Prozesse gemeinsame Teilpfade und Verzweigungen. Die Ausführung beginnt zeitorientiert beim periodischen Prozess an der Wurzel. Verzweigungen sind datenorientierte Services, an denen die Reihenfolge datenabhängig entschieden wird. Die Schedulingstrategie kombiniert zeitorientierte und datenabhängige Ausführungsreihenfolgen. Im Vergleich dazu sind die Schedulingverfahren aus Abschnitt 4.1.4 nur zeitorientiert. Die Ausführung der Prozesse erfolgt gemäß den Anforderungen aus Abschnitt 4.1.6 unterbrechungsfrei. Die Schedulingstrategie ist nicht-präemptiv.

Diese Aufgabe wird als Schedulingproblem für prozessorientierte, ubiquitäre Rechnersysteme formuliert.

Problem 4.1 (Scheduling serviceorientierter Prozesse)**.** *Prozessorientierte, ubiquitäre Rechnersysteme sollen die Ausführungsreihenfolge von Prozessen sowohl zeitorientiert wie auch datenorientiert organisieren.* (p_k) *ist eine vom Scheduler erzeugte Folge von Prozessen mit der erzeugenden Funktion* $\mathcal{F} : \mathbb{N} \rightarrow \mathcal{P}$, $n \mapsto p_k$. $\mathcal{F}$ *wird auch Schedulingstrategie genannt und muss folgenden Anforderungen genügen:*

1. $\forall s_i, k : Pred_k(s_i) = \emptyset$ *werden periodisch ausgeführt mit* $a(s_{i_1}) < a(s_{i_2})$ *gdw. für* $n_1 < n_2$*:* $\mathcal{F}(n_1) < \mathcal{F}(n_2)$ *gilt, d.h. Prozess* p_{k_1} *wird vor Prozess* p_{k_2} *ausgeführt.*
2. $\forall s_i : s_i \in p_k$ *wird die Reihenfolge* $(Pred_k(s_i), s_i, Succ_k(s_i))$ *ununterbrechbar ausgeführt.*

$Pred_k(s_i)$, $Succ_k(s_i)$ *bezeichnen Vorgänger- und Nachfolger-Service von* s_i *im Prozesspfad* p_k.

An den Verzweigungspunkten erlaubt der Graph die Wiederverwendung von Prozessergebnissen aus den Puffern und die Vermeidung von Redundanzen. Die Pufferinhalte bleiben bis zur nächsten Ausführung der periodischen Wurzel erhalten. Die Datenverarbeitung wird dadurch strukturell unterstützt. Dies ist in den Rangordnungen des Task-Modells nicht vorgesehen.

4.3.2.2 Graphentransformation

Die Definition des Servicegraphen in Abschnitt 4.3.2 lässt zu, dass ein Service von mehreren Puffern Eingangsdaten verarbeiten kann (Abbildung 4.12, links). Dies ist zum Beispiel bei der Datenverarbeitung mehrerer Quellen notwendig. Beim Prozessscheduling wird nun der Service mehrmals ausgeführt, da mehrere Pfade darin zusammenlaufen und Prozesse nicht abgebrochen werden können. Diese Redundanz wird verhindert, indem der Servicegraph in einen *Baum* umgewandelt wird. Abbildung 4.12 illustriert den Vorgang. Die Transformation muss zuerst die zusam-

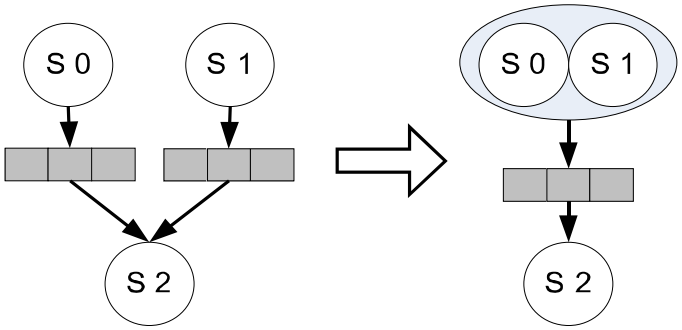


Abbildung 4.12: Transformation des Servicegraphen in einen Baum. s_0 und s_1 werden in einen Service verschmolzen und zwei Prozesse werden zu einem einem zusammengelegt.

menlaufenden Pfade im Servicegraphen identifizieren und im zweiten Schritt die Vorgängerservices zu einem neuen Pfad verschmelzen.

Dazu wird die Adjazenzmatrix V_{ij} des Servicegraphen aufgestellt, die die in Pfaden organisierten Services enthält. Die Puffer werden nicht betrachtet. Es ist $v_{ij} = 1$, gdw. $\exists^{=1} b_l \in B$ mit $s_i \xrightarrow{e_m} b_l$ und $b_l \xrightarrow{e_n} s_j$ gemäß der Inzidenzrelation $\mathcal{I}$. In V_{ij} ist die Wurzel die Spalte j mit $\sum_i v_{ij} = 0$. Blätter sind die Zeilen i mit $\sum_j v_{ij} = 0$. Ausgehend von den Blättern werden zusammenlaufende Pfade gesucht und die Vorgänger verschmolzen.

1. **Identifikation:** Für zusammenlaufende Pfade existieren Services s_j mit Spaltensumme $\sum_i v_{ij} > 1$, d.h. es gibt mehr als einen Vorgänger zu einem Blatt oder dessen Vorgängern.

2. **Verschmelzung:** Es sei s_j der Service gemäß vorheriger Identifikation. Es werden die Services $Pred(s_j) = \{s_{i_1}, \ldots, s_{i_n}\}$ in vier Schritten verschmolzen

   (a) Generiere neuen Service $s_{i_1 i_2}$ mit Servicefunktion f

   (b) Füge $Pred(s_j)$ Services in f in folgender Reihenfolge ein: wenn $v_{s_{i_1} s_{i_2}} = 1$, dann (s_{i_1}, s_{i_2}).

(c) Addiere Zeile i_2 auf i_1 mit $v_{i_1 j} = (v_{i_1 j} + v_{i_2 j}) \mod 2$ und $v_{i_1 i_1} = 0$.

(d) Elimiere Zeile und Spalte i_2 aus V_{ij}. Service $s_{i_1 i_2}$ nimmt die Position von s_{i_1} in V_{ij} ein.

Das Verfahren wird iterativ fortgesetzt, bis der Identifikationsschritt keine neuen Services mehr findet. Im Anhang C ist ein illustratives Beispiel der Transformation aufgezeigt. Es ist ausreichend, zu verschmelzende Services hintereinander auszuführen, um sie in einem Pfad zu vereinigen. Bestehende assoziierte Puffer und Services bleiben erhalten. Im entstehenden Baum gibt es so viele Prozesse wie Blätter s_j mit $Succ(s_j) = \emptyset$.

4.3.3 Systemarchitektur

Services und Prozesse sind Strukturmaßnahmen, mit denen Applikationen auf ubiquitären Rechnersystemen entworfen und organisiert werden. Der Servicegraph ist *horizontal* in die Systemarchitektur eingebettet, was Abbildung 4.13 illustriert. Die

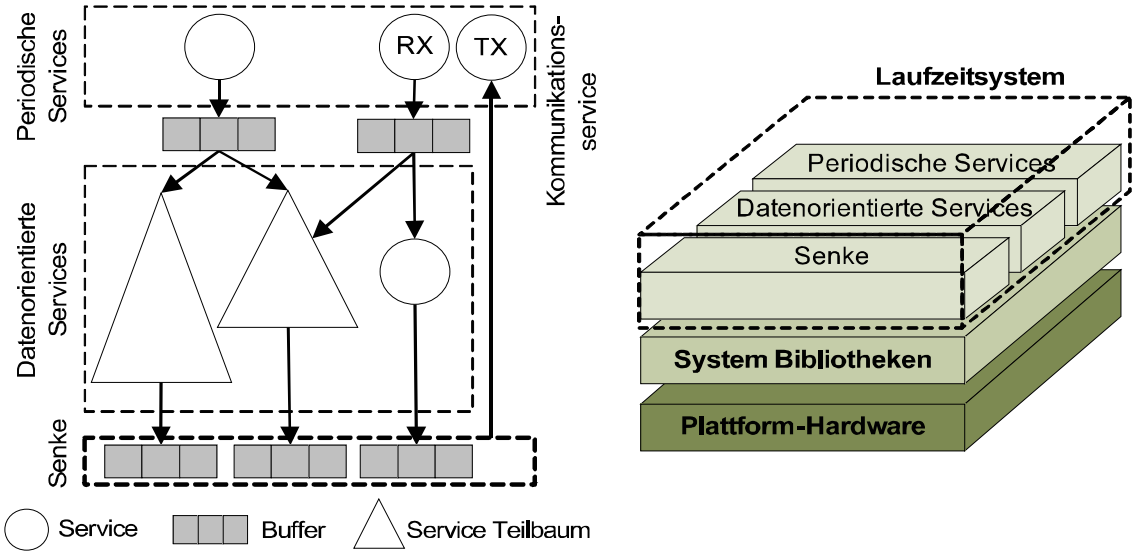


Abbildung 4.13: Servicegraph und horizontale Einbettung des Servicegraphen in die Systemarchitektur

Architektur ist geschichtet. Über der Plattform-Hardware sind Systembibliotheken für den Plattformzugriff angesiedelt. Services sind in ihrer Funktionalität nicht beschränkt und können alle darunterliegenden Systemfunktionen nutzen. Die Funktionskapselung in Services und die Graphenstruktur erlauben die einfache *Austauschbarkeit* von Services und Prozessen innerhalb einer Applikation.

Eine besondere Stellung nimmt der Kommunikationsservice ein, der in Sende- (TX) und Empfangskomponente (RX) gegliedert ist. Die Datensenke beendet die Prozesspfade im Graph. Die datenorientierten Services der Blätter schreiben die Daten in Puffer der Senke oder beenden sich. Einer der Puffer ist der assoziierte Eingabepuffer für den TX-Service. Umgekehrt kann der RX-Service die Daten anderen Services zur Verfügung stellen.

4.4 Periodische Prozesse

In diesem Abschnitt wird die kooperative und kollaborative Organisation periodischer Prozesse eingeführt. Prozesse werden gemäß des serviceorientierten Systementwurfs formuliert. Die Bewertung der erfolgreichen Datenverarbeitung wird anhand eines neues Qualitätsmaßes - *der Datenechtzeit* - vorgenommen. Die Datenechtzeit berücksichtigt die zeitnahe Datenverarbeitung auf eingebetteten, ubiquitären Rechnersystemen.

4.4.1 Motivierendes Beispiel

Die Appliance AwarePen [4][75][76] ist ein Stift mit eingebettetem Rechnersystem, der komplexe Informationen für Anwendungen wie automatische Meeting-Annotationen und Steuerung von Geräten [77] liefert. Periodisch werden Daten von zwei Beschleunigungssensoren akquiriert, gefiltert, und zu Interaktionsmustern verarbeitet. Weitere Sensorik wie Mikrofon und Lichtsensoren bestimmen Ort und Umgebungssituation. Die Abbildung 4.14 illustriert die Prozesse. Die Anwendungsumgebung des

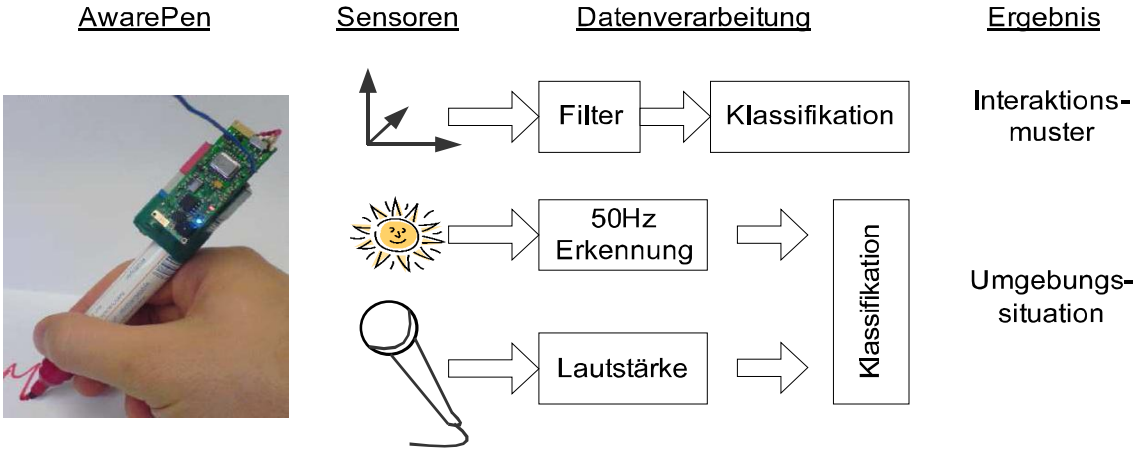


Abbildung 4.14: Links: AwarePen (Quelle: [75]), Rechts: Periodische Prozesse der Informationsverarbeitung

AwarePen ist nicht bekannt, so dass keine Prozesspriorisierung angegeben werden kann. Daher muss *jeder einzelne* Prozess Daten *zeitnah* verarbeiten, um qualitativ hochwertige Informationen für Anwendungen bereitzustellen.

4.4.2 Problembeschreibung

Zentral für die periodische Prozessorganisation ubiquitärer Rechnersysteme ist die folgende Problemformulierung.

Problem 4.2 (Periodische Prozessorganisation)**.** *In unbekannten Einsatzumgebungen ohne weitere äußere Administration der Prozessausführung, bestimme eine Ausführungsreihenfolge periodischer Prozesse so, dass das Zusammenwirken von Prozessen zu einer hohen Qualität der Datenverarbeitung* ***jedes einzelnen*** *Prozesses führt.*

Die Problemformulierung bezieht sich nicht auf die Gesamtqualität der Datenverarbeitung. In unbekannten Einsatzumgebungen, in denen keine Überwachung oder

Administration von Prozessen stattfinden kann, darf kein Prozess dauerhaft benachteiligt werden. Dies soll an folgendem Beispiel erläutert werden.

Es seien p_1, p_2 zwei Prozesse, deren zeitnahe Datenverarbeitung durch die monoton fallende Qualitätsfunktion $Q : a(j_{p_i}) \mapsto q$ mit $Q \in [0; 1]$ bewertet wird. Der vom Scheduler zuerst aktivierte Job erhält die größte Qualität, d.h. wenn $a(j_{p_1}) < a(j_{p_2})$, dann $Q(p_1) > Q(p_2)$. Der Prozessjob an der zweiten Stelle erhält eine Qualität $Q < 1$ verursacht durch die Ausführungsdauer des zuerst ausgeführten Prozesses. Die Abbildung 4.15 zeigt die Qualitäten der Prozesse in Abhängigkeit von zwei Einflussfaktoren: (1) Schedulingreihenfolge und (2) Wirkung der variierenden Ausführungsdauer von p_2 auf p_1. Die Ausführungsdauer von p_1 wird als konstant angenommen, um das Schedulingproblem noch 3-dimensional darstellen zu können. Auf der xy-Ebene ist $Q(p_1)$ zusätzlich als Kontur projiziert. An den Rändern der Kur-

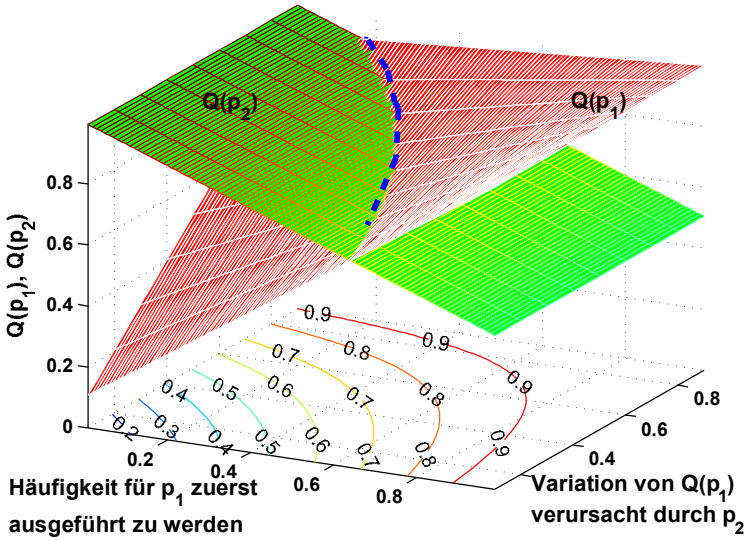


Abbildung 4.15: Bewertung der zeitnahen Datenverarbeitung bei verschiedenen Ausführungsreihenfolgen zweier Prozesse. Im idealen Fall soll die Prozesseinplanung die Schnittlinie der Flächen erreichen. Einflussfaktoren: (1) Schedulingreihenfolge und (2) Wirkung der variierenden Ausführungsdauer von p_2 auf p_1

ven in Abbildung 4.15 gehen die Einzelqualitäten der Prozesse stark auseinander. Eine für ubiquitäre Rechnersysteme geeignete Schedulingstrategie muss die Qualitätsdifferenz zwischen den Prozessen minimieren. Im Beispiel wird das Minimum entlang der eingezeichneten Schnittlinie der beiden Flächen erreicht, d.h. p_1 und p_2 werden in *fairer Reihenfolge* eingeplant. Fairness wird durch ein geregeltes Zusammenwirken und -arbeiten, d.h. Prozesskooperation und -kollaboration, erreicht. Der Regelungsmechanismus muss die Ausführungsreihenfolge beeinflussen, so dass die Schwankungen kompensiert werden.

4.4.3 Datenechtzeit - Bewertung kooperativer und kollaborativer Prozesseorganisation

Für ubiquitäre Rechnersysteme steht im Vordergrund die zeitnahe Datenverarbeitung. Die Zeit von der Entstehung eines Datums bis zu seiner nächsten Verarbeitung

ist beschränkt. Zeitnähe bedeutet, dass die Daten möglichst frühzeitig verarbeitet werden.

Definition 4.3 (Datenechtzeit)**.** *Die Datenechtzeit beschreibt eine Zeitschranke d, bis zu der die Daten nach ihrer Erzeugung verarbeitet werden können. Für eine datenechtzeitfähige Verarbeitung gilt: $a_{i,n} < d$, wobei $a_{i,n}$ die Aktivierungszeit des aktuellen Jobs des verarbeitenden Services s_i ist.*

Die Bewertung der zeitnahen Datenverarbeitung über die Datenechtzeit erfolgt mittels einer Qualitätsfunktion. Es ist $Q : \mathcal{S} \rightarrow \mathbb{Q}$, $a_{i,n} \mapsto q$ eine monoton fallende Funktion auf dem Bereich $[0; d]$ mit normierten Wertebereich $Q \in [0; 1]$.

Intuitiv gesprochen, drückt die Datenechtzeit eine Alterung eines Datums aus. Ist die Zeitschranke überschritten, so soll das Datum nicht weiterverarbeitet werden. Je kurzfristiger die Weiterverarbeitung erfolgt, desto höher ist die Qualität der Ausführung. Die Datenechtzeit bewertet somit die vom Scheduler getroffene Reihenfolge der Prozessausführung.

Die Qualitätsfunktion der Datenechtzeit hat für periodische und datenorientierte Services unterschiedliche Ausprägungen. Die Qualität eines periodischen Service s_i spiegelt den aktuellen Jitter der Ausführung wider.

Definition 4.4 (Qualitätsfunktion periodischer Services)**.** *Die Zeitschranke der Datenechtzeit ist $d = T_i$. Die Qualität eines periodischen Service s_i zum Zeitpunkt der aktuellen Ausführung $t_{i,n}$ ist definiert als die lineare, monoton fallende Funktion auf dem Bereich $[0, T_i]$:*

$$Q^T(s_i) = 1 - \frac{\min(t_{i,n} - a_{i,n}, T_i)}{T_i}$$

mit $a_{i,n} = a_{i,n-1} + T_i$. Wird der Service über eine Periode verzögert, so wird er ausgelassen und erhält die Qualität $Q^T(s_i) = 0$.

Die Datenechtzeit datenorientierter Services wird durch die Aktualität der Daten in den assoziierten Eingabepuffern bestimmt. Diese Daten entstehen am Ende der Ausführungsdauer des direkten Vorgängerservice s_j zum Zeitpunkt $t_{j,n} + C_j$.

Definition 4.5 (Qualitätsfunktion datenorientierter Services)**.** *Die Zeitschranke des datenorientierten Service s_i ist implizit über den periodischen Wurzelknoten s_k des Prozesspfades p_k, in dem s_i enthalten ist, gegeben. Es ist $d_i = T_k$ mit $k \in Pred(i)$ und s_k ist periodischer Service. Die Qualität der aktuellen Ausführung $t_{i,n}$ von s_i ist definiert als die lineare, monoton fallende Funktion auf dem Bereich $[0; T_k]$:*

$$Q^D(s_i) = maxQ(i) - \min\left(\frac{t_{i,n} - (t_{j,n} + C_j)}{T_k}, maxQ(i)\right)$$

mit $j = Pred_k^1(i)$ als der direkte Vorgänger s_j von s_i im Prozesspfad p_k. Mit $maxQ(i) \in [0; 1]$ kann die Datenechtzeit nochmal verschärft werden, zum Beispiel zur Realisierung von spezifischen Prioritäten.

Die Abbildung 4.16 zeigt die Qualitätsfunktion eines datenorientierten Service.

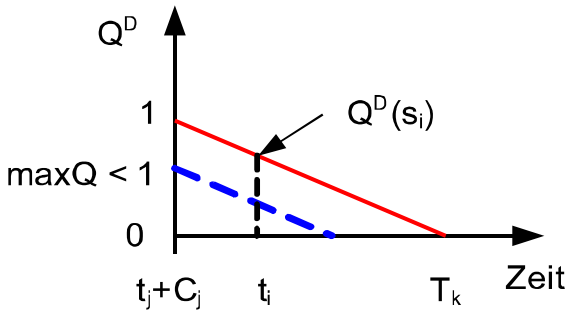


Abbildung 4.16: Qualitätsfunktion datenorientierter Services. Es wird die zeitliche Differenz zwischen Erzeugung der Vorgängerdaten $t_j + C_j$ und Beginn der Verarbeitung von s_i zum Zeitpunkt t_i im Vergleich zur Wurzelperiode T_k bewertet.

4.4.4 Datenechtzeiten von Prozessen im Servicegraph

Um die Prozessorganisation zu bewerten, müssen auch Prozesse im Servicegraph mittels Datenechtzeiten bewertet werden. Wird ein Prozesspfad in dem Servicegraph in die Tiefe abgelaufen, so repräsentiert er die zeitnahste Ausführung aller Services auf diesem Pfad und damit die Ausführung mit der höchsten erreichbaren Qualität. Prozesspfade, die davon verzweigen, erfahren eine Qualitätsminderung, da Daten an den Verzweigungen verzögert verarbeitet werden. Die Berechnung der Datenechtzeit von Prozessen soll die Gesamtheit der Qualitätsminderungen berücksichtigen und folgende Anforderungen erfüllen:

Kommutativität Die Qualitätsminderung bei Vertauschung zweier hintereinander ausgeführter Pfade mit gleicher Ausführungszeit ist gleich.

Unabhängigkeit von der Serviceanzahl Die Anzahl der Services im Pfad hat keinen Einfluss auf die Qualität des Prozesspfades.

Gemeinsame Teilpfade Qualitätsminderungen in gemeinsamen Teilpfaden von Prozessen sollen nicht mehrfach bewertet werden.

Qualitäten sind monoton fallend. Die, und nur die, Verzögerung der Datenverarbeitung an den Verzweigungsstellen führt zu Qualitätsminderungen. Erfolgte Qualitätsminderungen können im Laufe der Prozessausführung nicht kompensiert werden.

Diese Forderungen werden erfüllt, wenn jeder Service die Informationen über Qualitätsminderungen seiner Vorgänger enthält. Mit $maxQ(i) = Q^{T,D}(Pred_k^1(i))$ wird die Qualität des direkten Vorgängers von s_i vererbt. Die Qualität eines Service bei Ausführung in einem Prozesspfad k ist

$$Q_k(s_i) = \begin{cases} Q^D(s_i) = maxQ(i) - \min\left(\frac{t_{i,n}-(t_{j,n}+C_j)}{T_k}, maxQ(i)\right) & \text{, falls } Pred_k^1(i) \neq \emptyset \\ Q^T(s_i) & \text{sonst} \end{cases} \tag{4.1}$$

Die Zeitschranke der Datenechtzeit eines Prozesses p_k wird nur durch den periodischen Wurzelknoten $s_k \in p_k$ bestimmt. Es ist $d_k = T_k$. Die Qualität der Ausführung eines Prozesses p_k mit Servicequalitäten gemäß Gleichung 4.1 ist

$$Q^P(p_k) = Q^D(s_i) \text{ mit } s_i \in p_k \text{ , } Succ_k(s_i) = \emptyset \text{ und } Q^D(s_i) = Q_k(s_i) \tag{4.2}$$

Ist die Datenechtzeit eines Service nicht erfüllt, so folgt aus den Definitionen 4.4 und 4.5 sofort $Q^P(p_k) = 0$. Aus den vorangegangenen Gleichungen leitet sich folgendes Korollar ab.

Korollar 4.1. *Die Qualität der Prozessausführung ist nur von der Summe der zeitlichen Verzögerungen bei Aktivierung der Services entlang des Prozesspfades abhängig.*

Beweis. Alle Services auf Prozesspfad p_k sind datenechtzeitfähig, d.h. es ist $\frac{t_{i,n}-(t_{j,n}+C_j)}{T_k} = \min\left(\frac{t_{i,n}-(t_{j,n}+C_j)}{T_k}, maxQ(i)\right)$. Die fortgesetzte Anwendung von Gleichung 4.1 ergibt dann

$$\begin{aligned} Q^P(p_k) &= Q^D(s_i) \\ &= Q^T(s_k) - \frac{1}{T_k}(t_{Succ^1_k(k),n} - (t_{k,n} + C_k)) \ldots - \frac{1}{T_k}(t_{i,n} - (t_{Pred_k(i),n} + C_{Pred^1_k(i)})) \end{aligned}$$

Mit $D_{Pred_k(i),i} = t_{i,n} - (t_{Pred_k(i),n} + C_{Pred^1_k(i)})$ als Bezeichnung für die Aktivierungsverzögerung, folgt die Behauptung

$$Q^P(p_k) = Q^T(s_k) - \frac{\sum_{i \in p_k} D_{Pred^1_k(i),i}}{T_k}$$

□

4.4.5 Systemmodell kooperativer und kollaborativer periodischer Prozesse

In den vorangegangenen Abschnitten wurden die Grundlagen der Bewertung der Prozessausführung gelegt. Damit kann ein Systemmodell kooperativer und kollaborativer periodischer Prozesse aufgestellt werden.

Im unmittelbar ersten Schritt wird die Methode zur Übertragung des Budget/Kosten Modells auf spezifische Prozessklassen aus Abschnitt 3.4.4 angewendet. Es sind Budget, Kosten, Ziel zu identifizieren. Bei Ausführung wirken die periodischen Prozes-

Budget	Qualitätsvorrat für Bewertung und Priorisierung von Prozessen
Kosten	Erzielte Qualität eines Prozesses bei *zeitnaher* Ausführung
Ziel	Alle Prozesse arbeiten mit gleicher Qualität.

Tabelle 4.2: Entsprechungen des Budget/Kosten Modells für periodische Prozesse

se gegenseitig aufeinander und erzeugen gemäß der Schedulingreihenfolge jeweils Ausführungsqualitäten, die über die Budgets für jeden Prozess erfasst werden. Die Abbildung 4.17 illustriert das Systemmodell als Blockdiagramm. Der Scheduler sortiert zuerst die Servicegraphen gemäß ihrer Aktivierungszeit in FIFO Strategie, und plant die Prozesse p_i eines Graphen in absteigender Reihenfolge gemäß ihres jeweiligen Budgets B_i ein. Der datenorientierte Teil der Schedulingstrategie $\mathcal{F}$ wird wie folgt angegeben:

$$n < m \rightarrow \mathcal{F}(n) = p_i < \mathcal{F}(m) = p_j \text{ mit } B_i > B_j \tag{4.3}$$

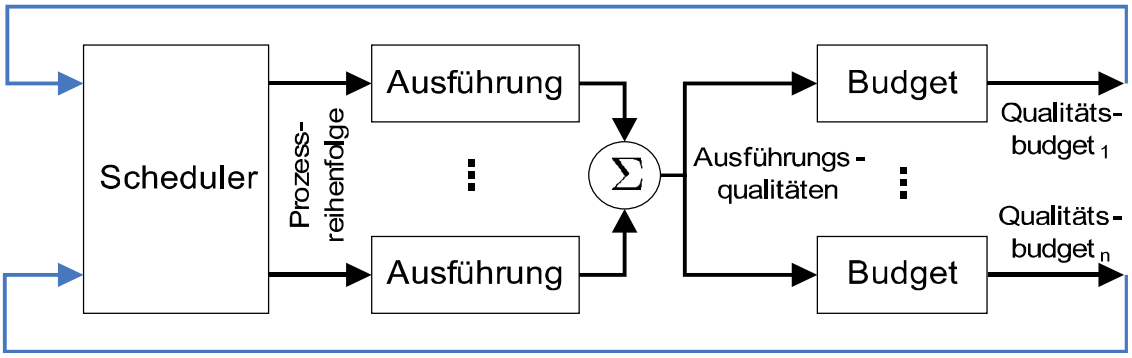


Abbildung 4.17: Systemmodell periodischer Prozesse. Der Σ Operator symbolisiert die Wirkung der Ausführung auf die Ausführungsqualitäten der Prozesse.

Im Fall $B_i = B_j$ werden p_i und p_j direkt nacheinander in einer zufälligen Reihenfolge ausgeführt. Erzielte Ausführungsqualitäten werden als Kosten vom Budget abgezogen und dieses an den Scheduler für die nächste Aktivierung zurückgeführt.

Bei Anwendung der Schedulingstrategie entsteht der Verlauf der Budgets in Abbildung 4.18. Die Länge eines Zeitschritts W entspricht der Periode des Servicegraphen $[(w-1)W; wW] = T$, der die Prozesse enthält. Entsprechend werden bei zeitlicher Betrachtung über mehrere Perioden Größen wie Budget und Ausführungsqualität mit Zeitindex w als Parameter versehen und die Prozesspfadidentifikatoren werden als Subindizes geführt. In Abbildung 4.18 lassen sich zwei Zeiträume unterscheiden.

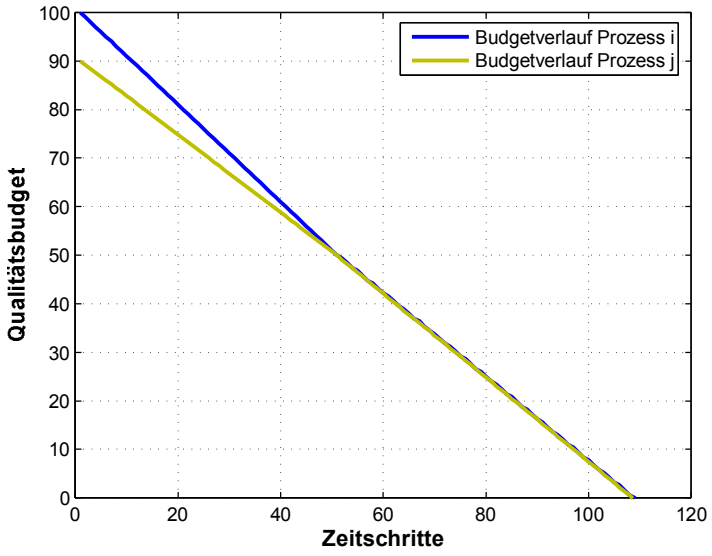


Abbildung 4.18: Verlauf der Budgets zweier Prozesse, die nach der Schedulingstrategie $\mathcal{F}$ aus Gleichung 4.3 eingeplant wurden.

Darüber wird folgendes Theorem formuliert.

Theorem 4.1. *Prozesse weisen durch die Schedulingstrategie $\mathcal{F}$ in Gleichung 4.3 folgendes Verhalten auf:* (1). *verschiedene Qualitätsbudgets werden sich angleichen;* (2). *nach der Angleichung werden die Budgets den Bereich $\epsilon = |B_i - B_j| < 1$ nicht verlassen.*

Beweis. Es seien p_i und p_j zwei beliebige, datenechtzeitfähige Prozesse eines Servicegraphens. Es sei p_i immer derjenige Prozess an erster Stelle im Schedule mit $B_i(w) > B_j(w)$.

Behauptung (1)**:** Angleichen der Budgets. Für Prozess p_i ist

$$\begin{aligned} B_i(w+1) &= B_i(w) - 1 \\ \Delta B_i(w) &= -1 \end{aligned} \tag{4.4}$$

Für Prozess p_j ist

$$\begin{aligned} B_j(w+1) &= B_j(w) - Q_j^P(w) \\ &= B_j(w) - \left(1 - \frac{D_{ij}(w)}{T}\right) \\ \Delta B_j(w) &= -1 + \underbrace{\frac{D_{ij}}{T}}_{<1, p_i, p_j \text{ datenechtzeitfähig}} \end{aligned} \tag{4.5}$$

D_{ij} ist die Aktivierungszeitverzögerung von p_j, da im Schedule p_j nach p_i eingeplant ist. Es ist $\Delta B_i(w) < \Delta B_j(w)$ und es folgt die Behauptung (1).

Behauptung (2)**:** Erhaltung des Gleichgewichts im Bereich ϵ. Es ist $B_i(w) = B_j(w)$. Aus den Gleichungen 4.4 und 4.5 folgt, dass für datenechtzeitfähige Prozesse die größte erreichbare Budgetdifferenz $\frac{D_{ij}}{T} < 1$ beträgt. Es folgt sofort $|B_i(w) - B_j(w)| < 1$. □

Die Schedulingstrategie aus Gleichung 4.3 erfüllt die Problemstellung der periodischen Prozessorganisation aus Abschnitt 4.4.2. Nach Theorem 4.1 werden die Prozesse so eingeplant, dass alle Prozessqualitätsbudgets sich in einer Umgebung $\epsilon < 1$ aufhalten. Dies führt zu einer ähnlich hohen Qualität *jedes einzelnen* Prozesses. Das Schedulingverfahren respektive der Scheduler wird daher auch Fair Quality Scheduling/-er (FQS) genannt.

Über die zwei Zeiträume kann die Entwicklung des Budgets mit einer linearen, monoton fallenden Funktion beschrieben werden. Die Konsequenz: Die miteinander verwobenen Prozesseregelkreise können in einzelne Regelkreise aus Abbildung 4.19 aufgeteilt werden, deren Zielverhalten dem einer linearen, monoton fallenden Funktion entspricht. Die Wirkungen der anderen Prozesse sind lediglich Störungen der Ausführungsqualität, die durch den Scheduler kompensiert werden müssen. Die sepa-

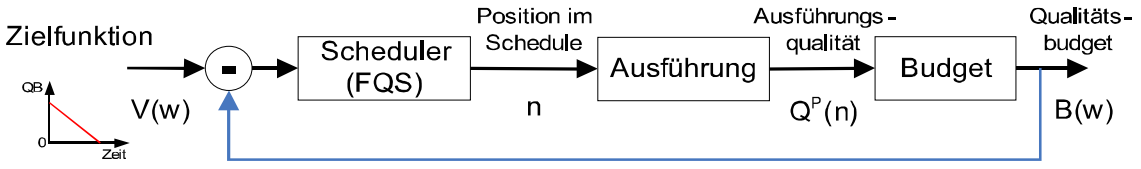


Abbildung 4.19: Separater Regelkreis für einen Prozess nach Aufteilung des periodischen Systemmodells

raten Regelkreise beschreiben das Laufzeitverhalten der FQS-Strategie für einzelne Prozesse. Sie sind für die Betrachtungen zur Prozesskooperation und -kollaboration ausreichend.

4.4.6 Prozesskooperation

Nach Definition 3.2 bedeutet Kooperation ein zielgerichtetes Zusammenwirken. Es sei $V_i(w)$ die monoton fallende, lineare Zielfunktion des Verlaufs des Qualitätsbudgets von Prozess p_i. Das Kooperationsziel aller Prozesse ist $V_1(w) = \ldots = V_n(w)$, d.h. auch jeder einzelne Prozess muss das entsprechende Ziel $V_i(w)$ erreichen. Aufgabe ist es, zu untersuchen, ob die geregelte Prozessausführung das gewünschte Ziel erreichen kann und unerwartete Störungen, zum Beispiel Verzögerung durch andere Prozesse, kompensieren kann. Während der vorherige Abschnitt nur das Gesamtverhalten betrachtete, gewinnt die Regelkreisanalyse Aussagen über Stabilität, Geschwindigkeit und Genauigkeit des kooperativen Verhaltens jedes einzelnen Prozesses. Vor der Analyse des Prozessverhaltens muss noch Theorem 4.2 bewiesen werden.

Theorem 4.2. *Bezogen auf das Verhalten eines einzelnen Prozesses bei kooperativer Prozessausführung verhält sich der Fair Quality Scheduler wie ein P-Regler.*

Theorem 4.2 formuliert eine erstaunliche Aussage. Die Strategie von FQS aus Gleichung 4.3 sortiert die Prozesse eines Servicegraphens gemäß ihres aktuellen Qualitätsbudget. Wird jedoch nur ein Prozess aus dieser Menge betrachtet, dann kann das Verhalten von FQS als P-Regler wie in Gleichung 4.7 modelliert werden. Dies macht eine geschlossene mathematische Beschreibung des Prozessverhaltens unter FQS möglich.

Beweis von Theorem 4.2. Es wird die Schedulingstrategie aus Gleichung 4.3 und der Regelkreis in Abbildung 4.19 betrachtet. Es ist zu zeigen, dass FQS der Transferfunktion eines P-Reglers entspricht.

Es ist p_i ein datenechtzeitfähiger Prozess. Die Kosten der Ausführung von p_i gemäß Korollar 4.1 sind $Q^P(p_i) = Q^T(s_i) - \frac{\sum_{k \in p_i} D_{Pred_i^1(k),k}}{T_i}$. Vereinfachend sei angenommen, dass es keine unerwartete Verzögerung bei der Serviceausführung im Prozesspfad gibt. Es ist also $Q^T(s_i) = 1$ und die Verzögerung D entspricht der Position n von p_i im Schedule. Die Ausführungsqualität von p_i ist

$$\begin{aligned} Q_i^P(n) &= 1 - \frac{n}{T_i} \\ Z\left[Q_i^P(n)\right] &= -\frac{1}{T_i} \end{aligned} \tag{4.6}$$

$Q_i^P(n)$ verbindet die Schedulingstrategie $\mathcal{F}$ des FQS mit der Ausführungsqualität. $Q_i^P(n)$ ist ein statisches System, eine lineare Funktion. In der Z Domäne entspricht dies einer Verstärkung $-\frac{1}{T_i}$. Der Offset ist eine konstante (nicht unerwartete) Störung. Zur Ermittlung der Transferfunktion $S(z)$ des Schedulers wird der Regelkreis in der Z Domäne betrachtet. Die Transferfunktion des geschlossenen Systems bestehend aus Scheduler $S(z)$, Ausführung $A(z)$ und Budget $G(z)$ ist

$$P(z) = \frac{B(z)}{V(z)} = \frac{S(z)A(z)G(z)}{1 + S(z)A(z)G(z)}$$

mit der Zielfunktion $V(z)$ als Eingangssignal und dem Budgetverlauf $B(z)$ als Ausgangssignal.

Bei kooperativer Prozessausführung muss das Verhalten von $P(z)$ dem Verhalten des Budget/Kosten-Modells $F(z)$ aus Abschnitt 3.4.3 entsprechen. Mit der Transferfunktion des Budget/Kosten-Modells $F(z) = \frac{Kz}{z(K+1)-1}$ aus Gleichung 3.19, $A(z) = Z\left[Q_i^P(n)\right] = -\frac{1}{T_i}$ und $G(z) = \frac{z}{z-1}$ ergibt sich

$$F(z) = P(z)$$
$$\frac{Kz}{z-1+Kz} = -\frac{S(z)z}{T_i(z-1)\left(1-\frac{S(z)z}{T_i(z-1)}\right)}$$

Umgestellt nach $S(z)$ folgt die Lösung für den Scheduler mit

$$S(z) = -KT_i \tag{4.7}$$

Für Prozesse im gleichen Servicegraphen folgt mit $\forall i, j : T_j = T_i = \text{konst.}$ sofort die Behauptung. Für Prozesse verschiedener Graphen gelten eigene P-Regler mit den jeweiligen Perioden als Reglerverstärkung. □

Der Regelkreis kann nun vollständig in Abbildung 4.20 angegeben werden. Die Ausführungsqualität $Q_i^P(n)$ verursacht eine konstante Störung der Größe 1. Die System-

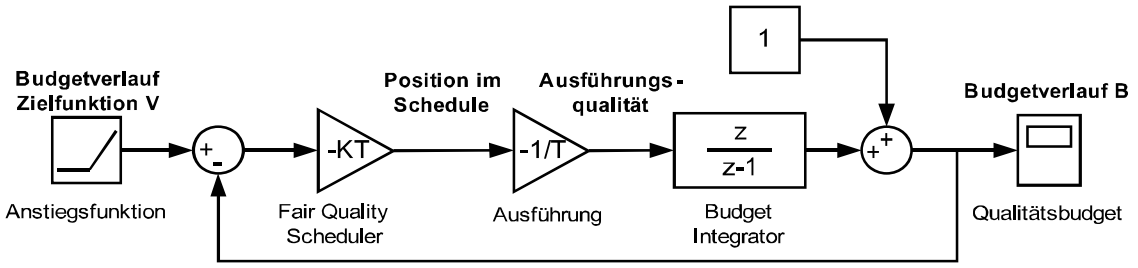


Abbildung 4.20: Simulink Regelkreismodell *von einem ausgewählten* kooperativen, periodischen Prozess

gleichung für einen Prozess p_i bestimmt sich entsprechend zu

$$P(z) = \frac{KT_iz}{T_i(z-1)+KT_iz} \tag{4.8}$$

$P(z)$ wird gemäß der Eigenschaften aus Abschnitt 3.4.3 analysiert. In Tabelle 4.3 werden die Eigenschaften Stabilität, Genauigkeit und Schnelligkeit bestimmt. Das

Eigenschaft	**Wert**	**Beschreibung**
Stabilität	Pol bei $\frac{1}{K+1}$	Für $K > 0$ ist $P(z)$ BIBO stabil
Genauigkeit	$P(1) = 1$ ist erfüllt	$P(z)$ wird vorgegebenen Zielwert ohne stationären Regelungsfehler erreichen.
Schnelligkeit	$w \approx \frac{\ln 0.02}{\ln \frac{1}{K_{\max}+1}}$ Wegen $0 \leq n \leq T_i$ in $Q_i^P(n)$ folgt, $0 \leq K \leq 1$. Es ist $K_{\max} = 1$ und $w \approx 6$.	$P(z)$ hat nach $w \approx 6$ Zeitschritten den Zielwert zu 98% erreicht. Dies gilt nur für Eingangssignale, die mindestens 6 Zeitschritte konstant bleiben.

Tabelle 4.3: Eigenschaften der Regelung kooperativer, periodischer Prozesse

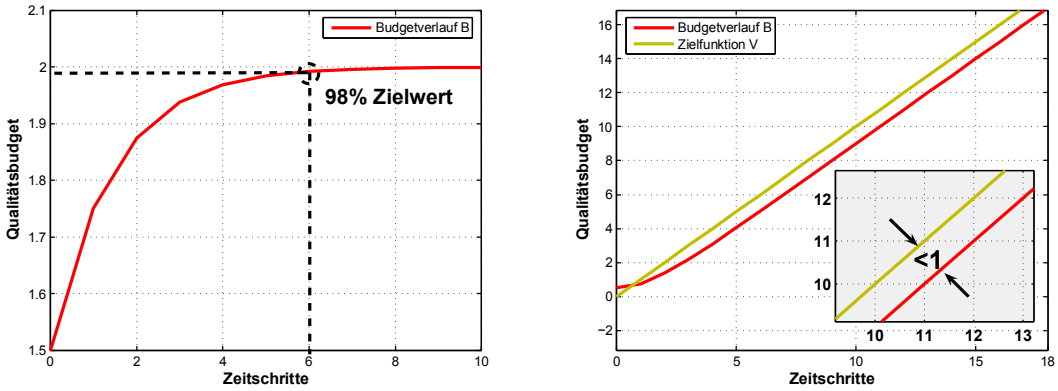


Abbildung 4.21: Links: Antwortverhalten auf Treppenfunktion. Rechts: Antwortverhalten auf lineare Funktion $V(w)$ als Eingangssignal. Eingebettete Vergrößerung: Abstand $|V(w) - B(w)| < 1$ wie in Theorem 4.1 gezeigt.

Verhalten von $P(z)$ wurde in Matlab Simulink simuliert. Die Ergebnisse sind in Abbildung 4.21 gezeigt. Sie bestätigen die vorhergesagten Eigenschaften aus der Analyse des Regelkreises $P(z)$. Die Abbildungen 4.22 und 4.23 zeigen die Messergebnisse der FQS Implementierung aus der Diplomarbeit von Andrea Sayer [78]. Abbildung 4.22 zeigt das Entstehen der Kooperation durch die Zusammenführung der Qualitätsbudgets von fünf Prozessen. Nachdem das vorgegebene Budget aller Prozesse aufgebraucht ist, werden alle Budgetvorgaben $B_i(0)$ auf ihren originären Wert zurückgesetzt. Der Einfluss von Störungen, zum Beispiel die plötzliche Än-

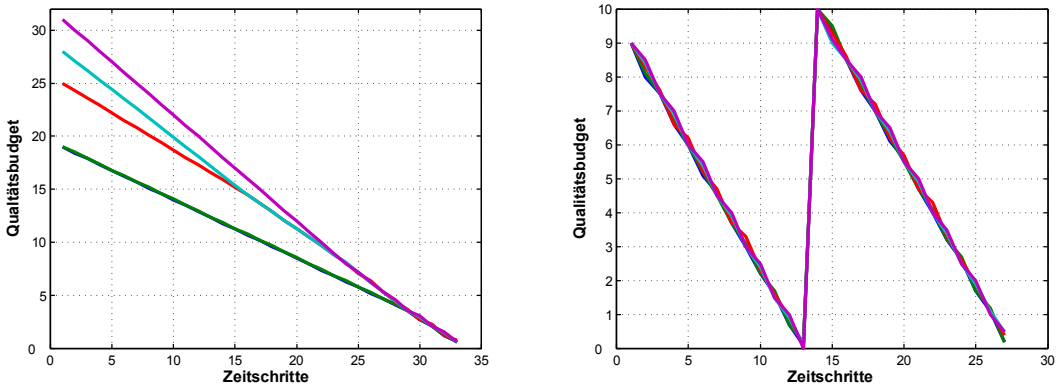


Abbildung 4.22: Links: Budgetverlauf bei Ausbildung der Kooperation und bei kooperativen Prozessverhalten. Bis zum Zeitschritt 29 sind Prozesse priorisiert - danach ist Kooperationssziel erreicht. Rechts: Kooperation über längere Zeiträume

derung der Ausführungsdauer eines Prozesses, wird in Abbildung 4.23 gezeigt. Bei andauernder, konstanter Störung gleichen sich alle Prozesse einer neuen Zielfunktion an. Das Theorem 4.1 zeigt, dass die Verläufe der Qualitätsbudgets zu einer gemeinsamen linearen Verlaufsfunktion konvergieren. Die Parameter der linearen Verlaufsfunktion sind unbekannt. Die Eigenschaften der geregelten Ausführung von Prozessen aus Tabelle 4.3 zeigen, dass ein vorgegebener Budgetverlauf in Form einer Zielfunktion $V_i(w)$ durch den Regler respektive Scheduler erreicht werden kann. Dabei ist es nicht notwendig, dass die Zielfunktion durchweg bekannt ist oder sogar konstant bleibt. Der Scheduler passt die Ausführung durch eine entsprechende Ein-

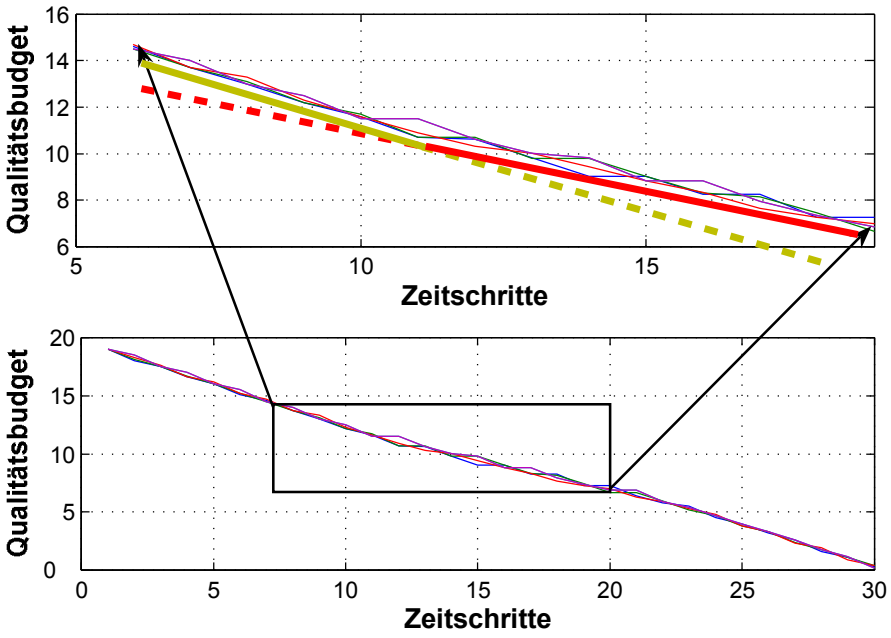


Abbildung 4.23: Änderung der Ausführungsdauer eines Prozesses im Zeitschritt 11. Der vergrößerte Ausschnitt zeigt einen flacheren Budgetverlaufs als Ergebnis der Störung. Der Verlauf gleicht sich einer neuen Zielfunktion an.

planung der Prozesse an. Beide Resultate kombiniert führen direkt zu dem Schluss, dass die Prozessorganisation das Kooperationsziel $V_1(w) = \ldots = V_n(w)$ erreicht.

4.4.7 Prozesspriorisierung und Kooperation

Der Kooperationsmechanismus durch den FQS Scheduler erlaubt es, flexibel und feingranular den Übergang zwischen spezifischer Priorisierung und Prozesskooperation zu gestalten. Für die Dauer von v Zeitschritten priorisiert FQS ein oder mehrere Prozesse *permanent* gegenüber anderen Prozessen. Ein Anwendungsentwickler hat somit auch die Möglichkeit über eine geeignete Wahl der Qualitätsbudgets von Prozessen eine feste Priorisierungsstrategie zu realisieren.

O.b.d.A. sind $B_i(w) > B_j(w)$ die Qualitätsbudgets zweier Prozesse im gleichen Servicegraphen: p_i wird gegenüber p_j solange priorisiert bis der Schnittpunkt $B_i(w) - Q_i^P(w) = B_j(w) - Q_j^P(w)$ beider Kurvenverläufe erreicht ist. Im Verlauf von v Zeitschritten wird die initiale Budgetdifferenz zwischen p_i und p_j ausgeglichen. Es ist

$$\Delta B_{ij}(0) = \sum_{w=0}^{v} \left(Q_i^P(w) - Q_j^P(w) \right) \tag{4.9}$$

Die Terme der Ausführungsqualitäten Q^P sind im allgemeinen vor der Ausführung nicht bekannt. In Sonderfällen ist es jedoch möglich, den Zeitpunkt zwischen Priorisierung und Kooperation zu bestimmen.

Es sei p_i der Prozess mit dem größten Budget $B_i(0) = \max_{p_k} B_k(0)$. T_i ist die Periode des Servicegraphen von p_i und p_j. Im Zeitraum $w \in [0; v]$ gibt es kein Budget $B_n(w)$ mit $B_i(w) > B_n(w) > B_j(w)$. Es ist dann $Q_i^P(w) = 1$ und $Q_j^P(w) = 1 - \frac{D_{ij}}{T_i}$. D_{ij} ist

die Aktivierungszeitverzögerung von p_j verursacht durch die Ausführungsdauer von p_i. Eingesetzt in Gleichung 4.9 und umgestellt ist

$$v = \Delta B_{ij}(0)\frac{T_i}{D_{ij}}$$

der Priorisierungszeitraum von FQS bis kooperatives Prozessverhalten erreicht wird.

4.4.8 Arbeitsbereich der Kooperation

Kooperation funktioniert nur im begrenzten Rahmen. Sind die Budgetunterschiede zu Beginn $\Delta B(w), w = 0$ zu groß, dann kann der Scheduler die Budgets in den $w = \frac{\min_{p_i} B_i(0)}{\min_{p_i} Q^P}$ Zeitschritten, die dem Prozess mit dem kleinsten Budget maximal zur Verfügung steht, nicht zusammenführen. Das System operiert dann außerhalb des Arbeitsbereiches des Schedulers. Das System bleibt im Arbeitsbereich, wenn

$$m_{\min} = \frac{\min_{p_i} B_i(0)}{\min_{p_i} Q^P} \geq m_{\max} = \max_{p_i} B_i(0)$$

erfüllt ist. Mit $B_{\min} = \min_{p_i} B_i(0)$, $\Delta B_{\text{minimax}} = \max_{p_i} B_i(0) - \min_{p_i} B_i(0)$ und $Q^P_{\min} = \min_{p_i} Q^P$ kann der Arbeitsbereich gemäß der maximalen Budgetdifferenz bestimmt werden

$$\frac{1 - Q^P_{\min}}{Q^P_{\min}} B_{\min} \geq \Delta B_{\text{minimax}} \tag{4.10}$$

Gleichung 4.10 beschreibt die hinreichende Bedingung, dass der Arbeitsbereich des Schedulers nicht verlassen wird. Kooperatives Verhalten kann für alle Prozesse gewährleistet werden.

4.4.9 Prozesskollaboration

Die Kollaboration gemäß Definition 3.3 bezeichnet die organisierte Zusammenarbeit. Kollaborationsziel ist es, dass kein Prozess sein Qualitätsbudget vor allen aufgebraucht hat.

Die Zieldefinition der Kooperation aus Abschnitt 4.4.6 scheint das Kollaborationsziel zu implizieren. Der Unterschied besteht darin, dass das Kooperationsziel nur dann gültig ist, solange der Arbeitsbereich des Schedulers eingehalten wird. Wenn die Bedingung aus Gleichung 4.10 nicht erfüllt wird, dann verschiebt der Kollaborationsmechanismus das System wieder in den Arbeitsbereich des Schedulers. Erreicht wird die Verschiebung durch einen *Budgettransfer* von Prozessen mit maximalem Budget zu Prozessen, deren Budget fast aufgebraucht ist. Über diesen Mechanismus werden die unabhängigen Prozesse miteinander gekoppelt.

Die Prozesskollaboration bestimmt das Qualitätsbudget, das für die Kooperation zur Verfügung steht. Sie ist der Kooperation vorgeschaltet und erhält als Rückkopplungsinformation die Budgets aller Prozesse. Kollaboration und Kooperation arbeiten in einer *Kaskade* wie in Abbildung 4.24 dargestellt. Der Kollaborationsmechanismus ist ein Zwei-Punkt-Regler MIMO System. Die Eingangssignale bestehen aus den Qualitätsbudgets der Prozesse. Das Regelungsgesetz der Kollaboration ist

$$u(w) = \begin{cases} 1, & \text{wenn } \min_{p_i} B_i(w) \leq 1 \wedge \max_{p_i} B_i(w) > 1 + \min_{p_i} B_i(w) \\ 0, & \text{sonst} \end{cases} \tag{4.11}$$

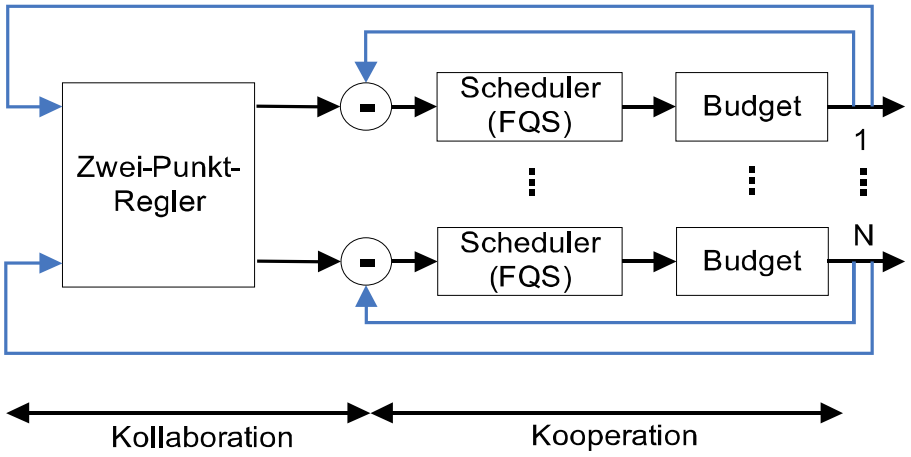


Abbildung 4.24: Kaskade von Kollaboration und Kooperation

Der Ausgang des Reglers ist die monoton steigende Folge der Qualitätsbudgets mit

$$\forall r, s \in \mathcal{P}:$$
$$0 < \underset{p_i}{\operatorname{argmin}}\, B_i(w) + u(w) \leq \ldots \leq B_r \leq \ldots \leq B_s \leq \ldots \leq \underset{p_i}{\operatorname{argmax}}\, B_i(w) - u(w)$$
$$\text{und} \sum_{p_i} B_i(w) = \text{ konst.}$$

Kooperatives und kollaboratives Prozessverhalten wird in Abbildung 4.25 anhand des zeitlichen Verlaufs von Qualitätsbudgets gezeigt. Kooperatives Verhalten von vier Prozessen besteht ab Zeitschritt 15. Kollaboration tritt ab Zeitschritt 21 ein und führt schlussendlich *alle* Qualitätsbudgets zusammen. Kollaborative Prozesse

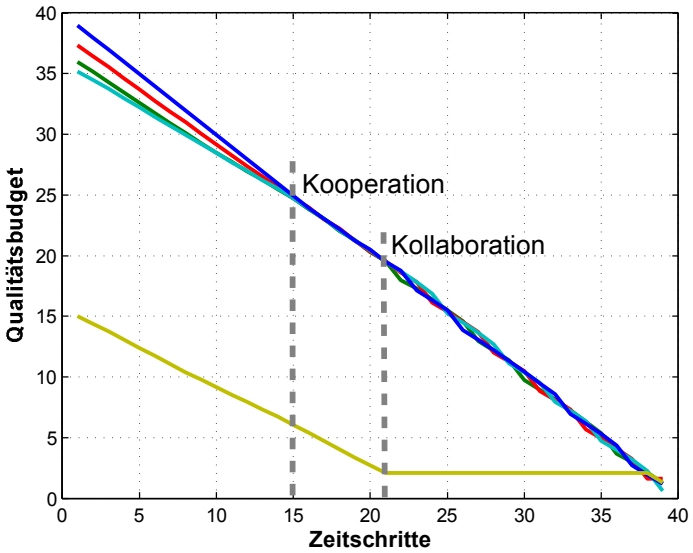


Abbildung 4.25: Zeitlicher Verlauf von Qualitätsbudgets bei kooperativer und kollaborativer Organisation periodischer Prozesse

koppeln ihre Budgets in einer Nullsumme, d.h., die Summe der Qualitätsbudgets ist vor und nach Ausführung von Gleichung 4.11 konstant. Der Transfer findet nur

zwischen dem kleinsten und größten Budget statt. Alle anderen Qualitätsbudgets bleiben unverändert. Die Kaskade erlaubt es, das MIMO System der Kollaboration transparent mit dem SISO System der Kooperation zu verbinden. Der Arbeitsbereich des Kooperationsmechanismus wird nicht verlassen. Kooperatives Prozessverhalten wird zeitlich maximal nutzbar gemacht.

4.4.10 FQS - Fair Quality Scheduler

Die kooperative und kollaborative Prozessorganisation periodischer Prozesse ist im Fair Quality Scheduler (FQS) realisiert. FQS ist eine Lösung des in Abschnitt 4.3.2.1 formulierten Problems des Scheduling serviceorientierter Prozesse. Bei Initialisierung wird der Servicegraph aufgestellt und in einen Baum transformiert. Durch Tiefensuche werden Prozesspfade identifiziert. Aufeinanderfolgende Services in den Pfaden repräsentieren eine datenorientierte Reihenfolge. Prozesspfade werden über die Periode der Wurzel in zeitlicher Aktivierungsreihenfolge organisiert.

Mittels des aktuellen Qualitätsbudgets bestimmt FQS gemäß der datenorientierten Strategie $\mathcal{F}$ an den Verzweigungspunkten im Servicegraphen den nächsten auszuführenden Prozesspfad. In der kooperativen Prozessorganisation übernimmt FQS die Funktion einen P-Reglers. Die Prozessreihenfolge verstärkt respektive mindert die Ausführungsqualität der Prozesse. Vorgeschaltet ist ein Zwei-Punkt-Regelmechanismus für kollaboratives Prozessverhalten. Dieses Instrument stellt den Arbeitsbereich des P-Regelungsverhaltens von FQS sicher.

Nach der Ausführung ermittelt FQS bereits ausgeführte Teilpfade und berechnet die Ausführungsqualitäten. Jede begonnene Prozessausführung ist mit einer Ausführungsqualität respektive Kosten verbunden. Bricht im Zeitschritt w ein Prozess p_i aus internen Gründen ab oder wird die Datenechtzeit verletzt, so ist $Q_i^P(w) = -1$. FQS stellt sicher, dass kein Priorisierungsvorteil durch Abbruch erzeugt werden kann.

FQS ist ein nicht-präemptiver, dynamischer und online-fähiger Scheduler. Neu in den Servicegraphen eingeordnete Services können in den Prozesspfaden instantan berücksichtigt werden. Die Abbildung 4.26 zeigt die Anordnung der Komponenten der FQS Architektur. FQS wird unterstützt durch das vom Autor der vorliegenden Arbeit in [52] vorgeschlagene Verfahren zur Jitterkorrektur. Jitter verändert die Aktivierungszeitpunkte periodischer Prozesse. Jitter tritt zufällig auf, ist aber auf einen maximalen Wert begrenzt. Es entsteht eine Periodenabweichung als Folge von ungleichmäßigen Aktivierungszeitpunkten. Die Jitterkorrektur modifiziert die nächste Aktivierungszeit des Wurzelservice zu $a_{k,n} = a_{k,n-1} + T_k + Jit_{k,n-1}$, wobei $Jit_{k,n-1}$ die Abweichung von der Aktivierungszeit der $n-1$-ten Jobausführung von s_k bezeichnet. Dadurch kann ein sich langsam ändernder Jitter deutlich verringert werden. Das folgende Theorem trifft eine Aussage über die Grenzen der entstehenden Periodenabweichung bei Anwendung der Jitterkorrektur.

Theorem 4.3. *Die Jitterkorrektur* $a_{k,n} = a_{k,n-1}+T_k+Jit_{k,n-1}$ *begrenzt die Änderung der Periode* T_k *auf* $\pm \max_n Jit_{k,n}$.

Der Beweis zu Theorem 4.3 ist im Anhang D aufgeführt. Bei unkorrigiertem Verhalten kann im schlechtesten Fall die Periodenabweichung $\pm 2 \max_n Jit_{k,n}$ betragen. Die

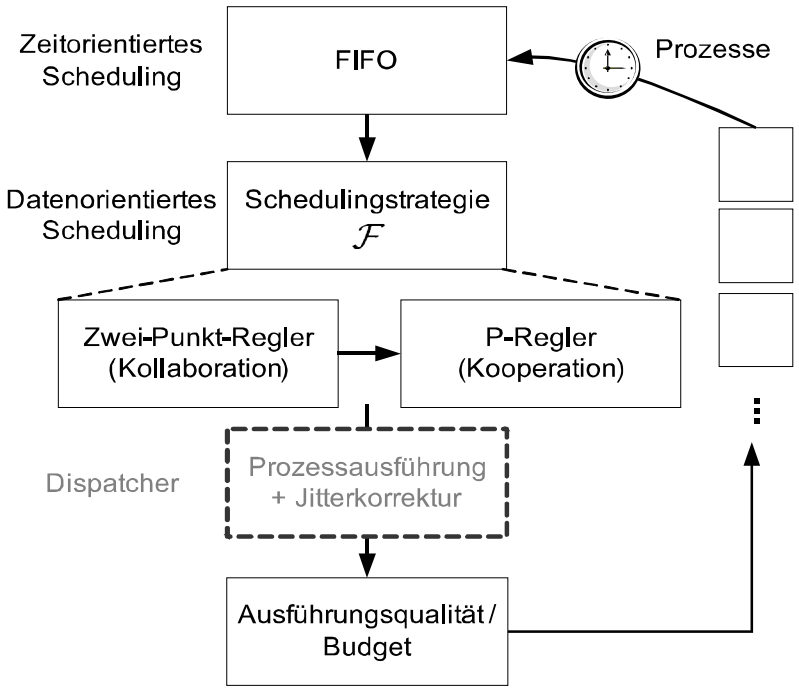


Abbildung 4.26: Komponenten der FQS Architektur zur kooperativen und kollaborativen Organisation periodischer Prozesse. Der ausgegraute Dispatcher ist nicht Teil von FQS.

Jitterkorrektur verringert mit geringen Auswirkungen auf die Periodenlänge deutlich den Jitter. Es folgt direkt aus den jeweiligen Definitionen, dass Periodenqualität des Wurzelservice und Ausführungsqualitäten von Prozesspfaden verbessert werden. Der Dispatcher implementiert das Verfahren.

4.4.11 Alternative Schedulingverfahren

Neben FQS werden weitere Schedulingverfahren zur Anwendung im Servicegraphen untersucht. Die Verfahren fokussieren dabei auf das Erreichen einer hohen Ausführungsqualität. Die folgenden Verfahren sind Gegenstand der Untersuchungen:

Verfahren mit Vorwissen über die Ausführungsdauer der Services

Shortest Total Delay First (STDF) Nach Korollar 4.1 ist die Prozessqualität nur von der Aktivierungsverzögerung entlang des Prozesspfades abhängig. STDF wählt in jedem Schedulingschritt denjenigen Pfad aus, der alle anderen Pfade geringstmöglich verzögert. STDF ist ein Greedy-Algorithmus.

Shortest Path First (SPF) Sortiert die Prozesspfade nach der Summe der Ausführungsdauern der Services auf dem Pfad. Eine alternative Abwandlung berücksichtigt die Ausführungsdauern von noch nicht ausgeführten Services an Verzweigungen im Graph. Das aktuelle Laufzeitverhalten kann besser berücksichtigt werden.

Shortest Path First mit Berücksichtigung von Teilpfaden Diese Variante priorisiert Prozesspfade mit großem Anteil an gemeinsamen Teilpfaden. Ein hoher Grad zeitnaher Wiederverwendung von Zwischenergebnissen wird erreichet. Es wird die Qualitätsminderung von Abschnitten, in die bereits erhebliche Ressourcen investiert wurden, begrenzt.

Verfahren mit Nutzung von Qualitätsinformationen

Qualitätsinformation aus der vorherigen Ausführung wird verwendet, um eine Ausführungsordnung herzustellen. Informationen über die Ausführungsdauer von Services *vor* ihrer Ausführung sind nicht erforderlich.

Highest Quality First (HQF) Prozesspfade, die zuvor eine hohe Qualität erreicht haben, werden bei der kommenden Ausführung bevorzugt.

Least Quality First (LQF) Um Fairness zwischen den Prozessen zu erreichen, werden Prozesspfade mit niedriger Ausführungsqualität bevorzugt. Kein Prozesspfad soll permanent hohe Qualitätsminderungen erleiden.

Leichtgewichtige, anwendungsspezifische Verfahren

Einsatz findet diese Klasse von Verfahren, wenn minimaler Implementierungsaufwand gefordert ist oder spezielle Anwendungsanforderungen vorliegen. Es ist detailliertes Vorwissen des Anwendungsentwicklers notwendig, welches in die Schedulingentscheidung direkt einfließt. Es findet keine Rückkopplung von Ausführungsinformationen statt.

First-In-First-Out (FIFO) plant die Prozesspfade lediglich nach Aktivierungszeit der periodischen Wurzel ein. Es entsteht eine zufällige Ausführungsreihenfolge bei Prozessen mit ein- und derselben Wurzel.

Anwendungsspezifische Prioritätsvergabe Eine manuelle und statische Priorisierung der Prozesspfade durch den Applikationsentwickler wird vorgenommen. Die Reihenfolge ist unveränderlich. Insbesondere können Anpassungen aufgrund neuer Prozessergebnisse nicht vorgenommen werden.

4.4.12 Vergleichende Betrachtungen

Die im vorigen Abschnitt aufgezählten Verfahren wurden in der vom Autor betreuten Diplomarbeit von Andrea Sayer [78] untersucht und miteinander verglichen. Die Tabelle 4.4 fasst die Ergebnisse zusammen und ordnet das neue FQS Verfahren ein.

4.5 Aperiodische Prozesse

Aperiodische Prozesse bedienen die Ereignisverarbeitung in eingebetteten, ubiquitären Rechnersystemen. Dieser Abschnitt stellt ein kooperatives Verfahren der aperiodischen Prozessorganisation vor. Kooperation ermöglicht zeitnahe Reaktionen auf Ereignisse und dass Ressourcen, insbesondere Speicher und Prozessorauslastung, effizienter zwischen den Prozessen verwaltet werden. Ein Vergleich mit der Verarbeitung in dem Betriebssystem TinyOS [13] und anderen bekannten Verfahren bestätigt die Anwendbarkeit in ubiquitären Rechnersystemen.

4.5.1 Motivierendes Beispiel

Zur Illustration der aperiodischen Prozessorganisation in ubiquitären Rechnersystemen soll das folgende Beispiel dienen. Ein elektronisches Siegel (eSeal) [7] überwacht Temperatur und Verschlusssicherheit von verderblichen Lebensmitteln. Die entsprechenden Informationen eines Temperatur- und eines Lichtsensors werden periodisch

Verfahren	Qualität	Aufwand	Bewertung
Verfahren mit Vorwissen über die Ausführungsdauer der Services			
Shortest Total Delay First (STDF)	i.a. sehr hohe Qualität, aber nicht optimal, da STDF greedy ist. Illustratives Gegenbeispiel in [78].	extrem hoher Aufwand bei Berechnung und Bestimmung der Ausführungsdauer	ungeeignet, da Ressourcen und Vorwissen i.a. auf ubiquitären Rechnersystemen nicht verfügbar
Shortest Path First (SPF)	oftmals sehr hohe und optimale Qualitäten erreichbar, dauerhafte Qualitätsminderung langlaufender Prozesse möglich	hoher Aufwand bei Bestimmung der Ausführungsdauer, Umsortierung der Warteschlange nach jeder Ausführung	ungeeignet, da Vorwissen i.a. auf ubiquitären Rechnersystemen nicht verfügbar
SPF mit Berücksichtigung von Teilpfaden	Verschlechterung unter starken Schwankungen der Ausführungsdauer	wie SPF	wie SPF
Verfahren mit Nutzung von Qualitätsinformationen			
Highest Quality First (HQF)	zuerst gewählte Prozessreihenfolge bleibt bestehen und legt Qualität fest, Änderung nur bei Prozessabbruch	Moderat, nach Berechnung der Qualitäten sehr einfache Sortierung	Für Extremsituationen: statisches Verhalten bei Laufzeitschwankungen, Re-Scheduling nur bei Prozessabbruch
Least Quality First (LQF)	alternierende Qualitäten durch Vertauschung des ersten und letzten Prozesspfades (Anhang E)	Moderat, nach Berechnung der Qualitäten sehr einfache Sortierung	Anwendung bei Gefahr der Verhungerung von Prozessen
Fair Quality Scheduler (FQS)	ausgeglichene Ausführungsqualität aller Prozesse	Moderat, nach Berechnung der Qualitäten sehr einfache Sortierung	sehr gute Eignung für unbekanntes Ausführungsverhalten und Umgebungen
Leichtgewichtige, anwendungsspezifische Verfahren			
First-In-First-Out (FIFO)	Qualität ist abhängig von Aktivierungszeit, dauerhafte Qualitätsminderungen möglich	extrem geringer Implementierungsaufwand	Einsetzbar, wenn Ausführungsdauer aller Prozesse sehr kurz im Vgl. zu Periode ist
Prioritätsvergabe	hohe Qualität für detailliert bekannte Anwendungen und Ausführungsumgebungen	sehr gering, statisches Verfahren	keine Eignung für dynamisches Anwendungsverhalten und unbekannte Umgebungen

Tabelle 4.4: Vergleich der Schedulingverfahren

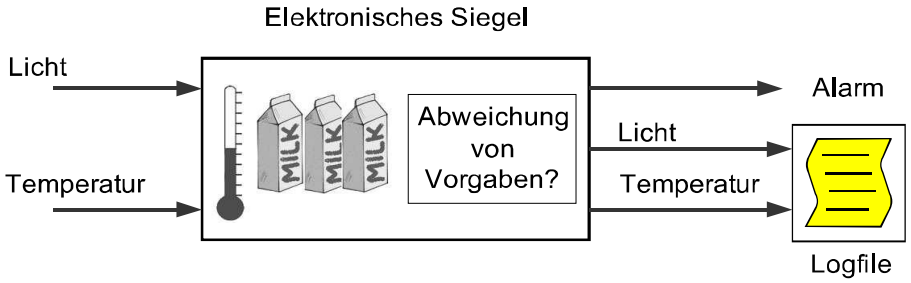


Abbildung 4.27: Aperiodische Prozessverarbeitung eines elektronischen Siegels

ausgelesen. Bei Abweichung von der Norm wird ein Alarm ausgelöst und ein Logfileeintrag geschrieben. Die Abbildung 4.27 illustriert das Beispiel. In dem Beispiel kann nicht vorhergesehen werden, durch welches Sensorereignis wann eine Abweichung festgestellt wird. Die Verarbeitung der Ausnahmen *beider* Prozesse und das Auslösen des Alarms muss durchgeführt werden.

4.5.2 Problembeschreibung

Als aperiodisch werden Ereignisse bezeichnet, die nicht regelmäßig eintreffen. Ihr Auftreten ist zufällig und kann nicht mit einer konstanten Periode beschrieben werden. Typische Prozesse mit aperiodischen Ereignissen sind asynchrone Kommunikationsprozesse, die an Schnittstellen des Mikroprozessors mit Peripheriekomponenten wie Sensoren und Aktuatoren stattfinden. Unterbrechungen, engl. Interrupts, informieren asynchrone Kommunikationsprozesse über ankommende Daten. Die Behandlung aperiodischer Ereignisse ist für eingebettete, ubiquitäre Rechnersysteme besonders interessant. Ereignisse wie sich verändernde Umgebungsbedingungen oder die Detektion neuer Kommunikationspartner starten bisher inaktive Applikationsprozesse. Ergebnisse der Datenvorverarbeitung von sensorischen Peripheriekomponenten, zum Beispiel Licht-, Temperatur- oder Bewegungsschwellwertdetektoren, können mittels Ereignissen zeitnah an die Prozessverarbeitung kommuniziert werden. Es werden folgende Grundannahmen vorausgesetzt:

- Ereignisse treten mit einer *unvorhersehbaren, zeitveränderlichen* Rate auf.
- Es können verschiedene Ereignistypen unterschieden werden, die durch dedizierte Prozesse verarbeitet werden.
- Für die Ereignisverarbeitung ist eine maximale Ausführungsdauer (WCET) bekannt sowie Speicher- und Rechenressourcen vorgegeben.

Aperiodische Serververfahren wie in Abschnitt 4.1.5 vorgestellt, haben zum Ziel die Antwortzeit zu minimieren und Echtzeitgarantien für die Ereignisverarbeitung zu geben. Daher wird eine Admission vorgenommen, so dass nur die Prozesse ausgeführt werden, die vorgegebene Ressourcenkapazitäten nicht überschreiten. Als Folge kann es passieren, dass bestimmte Ereignistypen gar nicht verarbeitet werden. Für ubiquitäre Rechnersysteme ist es wichtig die Verarbeitung nicht auszulassen, um auf Ereignisse zeitnah reagieren zu können. Es leitet sich die zentrale Problemformulierung der aperiodischen Prozessorganisation ab.

Problem 4.3 (Aperiodische Prozessorganisation)**.** *Unter unvorhersehbar auftretenden Ereignisabfolgen bestimme die Prozessausführung so, dass eine hohe Anzahl von Ereignissen von* ***jedem einzelnen*** *für einen Ereignistyp dedizierten aperiodischen Prozess verarbeitet wird. Dabei soll unter vorgegebenen Speicher- und Rechenzeitressourcen Überlast vermieden werden und ubiquitäre Rechnersysteme kontrollierbar bleiben.*

Für die Realisierung ist es notwendig, die zu verarbeitende Menge der Ereignisse zu begrenzen. Es folgt der Ansatz, dass nicht die Verarbeitung, sondern die zu verarbeitende Ereignisdatenmenge angepasst wird. *Faire Gestaltung* dieser Aufgabe bei Vermeidung von Überlast und unter vorgegebenen Speicher- und Rechenzeitressourcen stehen im Vordergrund.

4.5.3 Entwurf der aperiodischen Prozessverarbeitung

Aperiodische Prozesse sind in zwei Teile untergliedert.

Ereignis-Stub: Tritt ein Ereignis auf, so wird ein dafür dedizierter Stub aktiviert, der nur die Ereignisdaten in einen für diesen Typ vorgesehenen Puffer mit fester Länge kopiert. Der Ereignis-Stub kann andere Prozesse unterbrechen. Er hat eine extrem kurze Laufzeit, ist unabhängig von anderen Prozessen und nutzt einmalige Hardwareressourcen exklusiv.

Server: Datenechtzeitfähige, periodische Prozesse, die den gesamten Puffer an Ereignisdaten in einem Stück verarbeiten. Für jeden Ereignistyp gibt es einen Serverprozess. Die maximale Ausführungsdauer (WCET) ist bekannt. Der Verhältnis aus WCET und Periode ist die Serverauslastung.

Auch die periodischen Prozesse aus Abschnitt 4.4 können Ereignisse emittieren, indem sie direkt den entsprechenden Ereignis-Stub aufrufen. Über datenechtzeitfähige Serverprozesse zur periodischen Verarbeitung von Ereignissen ordnet sich die aperiodische Prozessverarbeitung in das periodische Laufzeitsystem ein. Abbildung 4.28 skizziert den Aufbau der aperiodischen Prozessverarbeitung.

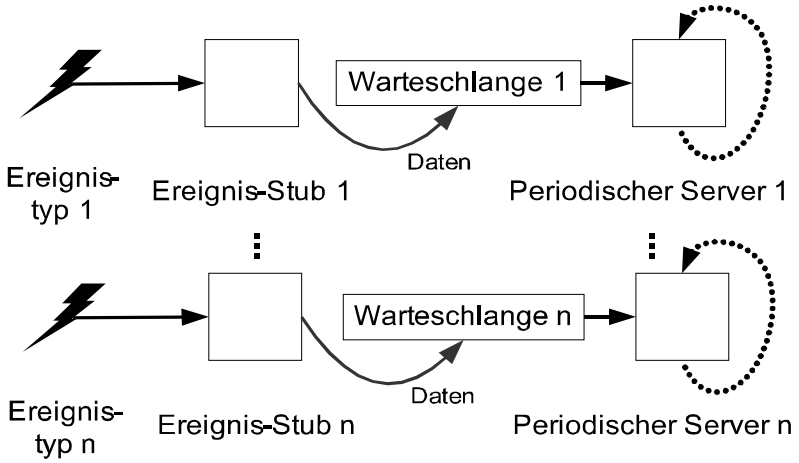


Abbildung 4.28: Aperiodische Prozessverarbeitung

Es folgt die mathematische Beschreibung des Entwurfs. Es sei $\mathcal{P}^{\mathrm{aP}}$ die Menge der aperiodischen Prozesse eines ubiquitären Rechnersystems. Für jeden Ereignistyp haben die Puffer die gleiche, feste Länge, d.h. $\forall i, j \in \mathcal{P}^{\mathrm{aP}} : l_i = l_j = \text{konst.}$. Mit jedem Puffer ist ein Serverprozess mit Periode T_i und Ausführungsdauer C_i assoziiert. Bei Ausführung des Servers werden alle Daten konsumiert. Anschließend ist der Puffer leer.

Es soll keine Admission der Serverprozesse erfolgen, sondern die zu verarbeitende Ereignisdatenmenge angepasst werden. Dies kommt in der Verlustwahrscheinlichkeit eines Ereignispuffers zum Ausdruck. Die Verlustwahrscheinlichkeit ist die Wahrscheinlichkeit, dass ein oder mehrere Ereignisse nicht mehr in den vorgesehenen Puffer aufgenommen werden können. Der Puffer bietet keine ausreichende Kapazität mehr an. Eine Verlustwahrscheinlichkeit von 1% bedeutet, dass im Mittel von 100 Füllvorgängen, es einmal zu einer Situation kommt, in der mehr Ereignisse in einer bestimmten Zeitspanne auftreten, als in den Puffer aufgenommen werden können.

Unter Berücksichtigung der Grundannahmen aus Abschnitt 4.5.2 wird die Verlustwahrscheinlichkeit modelliert. Die Ereignisse treten poissonverteilt mit Rate λ auf. Die Wahrscheinlichkeit, dass genau m Ereignisse innerhalb des Zeitintervall w mit der Länge einer Serverperiode $T = [wT; (w+1)T]$ auftreten, ist $P_\lambda(N(t+T)-N(t) = m) = \frac{(\lambda T)^m}{m!} e^{-\lambda T}$. Die Verlustwahrscheinlichkeit für einen Puffer der Größe l berechnet sich gemäß der Verteilungsfunktion zu

$$\begin{aligned} P(N(t+T) - N(t) \geq l) &= 1 - P(N(t+T) - N(t) \leq l) \\ &= 1 - e^{-\lambda T} \sum_{m=0}^{l} \frac{(\lambda T)^m}{m!} \end{aligned} \tag{4.12}$$

Innerhalb eines Zeitintervalls w wird die Ankunftsrate λ als konstant betrachtet. Sie kann sich jedoch von Intervall zu Intervall verändern. Die Rate λ ist nicht kontrollierbar. Die Serverperiode T ist jedoch beeinflussbar. Beide Größen sind spezifisch für einen Ereignistyp und werden im folgenden mit $\lambda_i(w)$ und $T_i(w)$ bezeichnet.

4.5.4 Systemmodell aperiodischer Prozesse

Der Entwurf erlaubt es, das Budget/Kosten Modell auf die aperiodische Prozessorganisation anzuwenden. Im unmittelbar ersten Schritt wird die Methode zur Übertragung des Budget/Kosten Modells auf spezifische Prozessklassen aus Abschnitt 3.4.4 bemüht. Es sind Budget, Kosten, Ziel zu identifizieren, die in Tabelle 4.5 zusammengefasst sind.

Budget	Verlustwahrscheinlichkeit
Kosten	Periodische Ausführung eines Serverprozess
Ziel	Alle Serverprozesse verursachen die gleiche Verlustwahrscheinlichkeit.

Tabelle 4.5: Entsprechungen des Budget/Kosten Modells für aperiodische Prozesse

Daraus wird die folgende Regelungsaufgabe abgeleitet.

Regelungsziel Erreichen einer gegebenen Verlustwahrscheinlichkeit

Stellgröße Serverperiode zur Verarbeitung von aperiodischen Ereignissen

Regelstrecke Systemgleichung 4.12 zur Berechnung der Verlustwahrscheinlichkeit

(nicht kontrollierbare) Störgröße Ankunftsrate λ der Ereignisse, messbar

Im Budget/Kosten-Modell wird die Verlustwahrscheinlichkeit durch Messung des Systemausgangs der Regelstrecke erfasst. Dies ist während der Laufzeit sehr schwierig, da über sehr lange Zeiträume gemessen werden muss, um eine zuverlässige Aussage zu erhalten. Die Systemgleichung 4.12 ist ein statisches System. Die Nachstellung der Serverperiode geschieht daher nicht über den Systemausgang, sondern wird direkt hergeleitet. Es ist

$$P^{\text{verlust}}_{\lambda_i(w)}(T_i(w)) = W = \text{konst.} \quad \text{gdw. } \lambda_i(w)T_i(w) = R = \text{konst.} \tag{4.13}$$

das Systemmodell für den aperiodischen Prozess i. $\lambda_i(w)$ ist der Erwartungswert der Poissonverteilung. Er kann durch Auszählen innerhalb einer Serverperiode ermittelt werden. Die Serverperiode ergibt sich sofort aus

$$T_i(w) = \frac{R}{\lambda_i(w)} \tag{4.14}$$

Diese Technik wird auch als Steuerung, engl. *feedforward control*, bezeichnet. Abbildung 4.29 illustriert das Steuerungsdiagramm. Zu beachten ist, dass der Systemausgang nicht verwendet wird. Sowohl die Systemgleichung 4.12 wie auch das Steue-

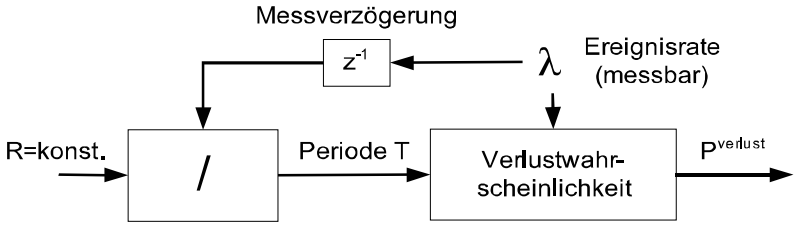


Abbildung 4.29: Blockdiagramm der aperiodischen Prozesssteuerung

rungsglied in Gleichung 4.14 für die Periodenberechnung beschreiben statische Systeme. Die Analyse hinsichtlich kooperativen Prozessverhaltens ist Gegenstand des folgenden Abschnitts.

4.5.5 Prozesskooperation

Aperiodische Prozesse verarbeiten unabhängig voneinander Ereignisse. Gemäß der Problemformulierung der aperiodischen Prozessorganisation soll jeder einzelne Prozess die gleiche Verlustwahrscheinlichkeit erreichen. Nach Definition 3.2 entspricht dieses zielgerichtete Zusammenwirken einer Prozesskooperation.

Für aperiodische Prozesse existiert zusätzlich die Bedingung, dass eine vorgegebene Systemlast U_b zur Verarbeitung der Ereignisse nicht überschritten werden darf, um Überlast zu vermeiden. Die Prozesskooperation muss daher das Ziel gegen die Systemlast U_b ausbalancieren. Es wird die Kooperation über die folgenden Teilziele formuliert.

Teilziel 1 Alle aperiodischen Prozesse einer Menge $\mathcal{M} \subset \mathcal{P}^{\mathrm{aP}}$ sollen die gleiche Verlustwahrscheinlichkeit aufweisen.

Teilziel 2 Alle aperiodischen Prozesse $p \notin \mathcal{M}$ sollen ebenfalls die gleiche Verlustwahrscheinlichkeit aufweisen.

Bedingung 1 Für die Kardinalität von $\mathcal{M}$ gilt: $1 \leq |\mathcal{M}| < |\mathcal{P}^{\mathrm{aP}}|$.

Bedingung 2 Die gesamte Systemauslastung durch die aperiodische Verarbeitung soll U_b nicht überschreiten.

Die Prozesskooperation verändert die Serverperioden aller aperiodischen Prozesse, um dieses Ziel zu erreichen. Es wird nach folgendem Schema vorgegangen:

Ausgangssituation Es sind alle Serverperioden so bestimmt, dass das Ziel gleicher Verlustwahrscheinlichkeiten erreicht ist.

Kooperation gemäß Teilziel 1 Es ist o.B.d.A. $\lambda_i(w+1) > \lambda_i(w)$, d.h. die Ereignisrate für den Ereignistyp i erhöht sich.

- Daraufhin muss $T_i(w+1)$ verkleinert werden, zu $T_i(w+1) = \frac{R}{\lambda_i(w+1)}$. Damit ist $P^{\mathrm{verlust}}_{\lambda_i(w+1)}(T_i(w+1)) = P^{\mathrm{verlust}}_{\lambda_i(w)}(T_i(w))$.
- Mit verkleinerter Serverperiode steigt die Serverauslastung $U_i(w) = \frac{C_i}{T_i(w+1)} > U_i(w+1) = \frac{C_i}{T_i(w)}$, so dass $\sum_i U_i(w+1) > U_b$.

Kooperation gemäß Teilziel 2 Die Idee ist, dass alle anderen aperiodischen Prozesse ihre Serverperioden verlangsamen, um die erhöhte Serverauslastung von i zu kompensieren.

- Es werden zwei Mengen von Prozessen $\boldsymbol{I}$, $\boldsymbol{J}$ mit $\boldsymbol{I} \cup \boldsymbol{J} = \mathcal{P}^{\mathrm{aP}}$ und $\boldsymbol{I} \cap \boldsymbol{J} = \emptyset$ definiert. In $\boldsymbol{I}$ sind die Prozesse aus Teilziel 1 enthalten. Durch die Steuerung gemäß Gl. 4.13 sind die Verlustwahrscheinlichkeiten der Prozesse in $\boldsymbol{I}$ konstant geblieben. Die Prozesse aus $\boldsymbol{J}$ haben unveränderte Ereignisraten λ_j und sollen durch Periodenverlangsamung die erhöhte Serverauslastung der $\boldsymbol{I}$ kompensieren. Die Serverauslastung durch die Prozesse $\boldsymbol{I}$ muss kleiner sein als die gegebene U_b. Die Zugehörigkeiten zu den beiden Mengen kann in jeder einzelnen Serverperiode neu bestimmt werden.
- Kooperation passt die Serverperioden der Prozesse aus $\boldsymbol{J}$ wie folgt an

$$T_j(w+1) = \frac{\sum_{j \in \boldsymbol{J}} C_j \lambda_j(w)}{RU_b - \sum_{i \in \boldsymbol{I}} C_i \lambda_i(w+1)} T_j(w) \tag{4.15}$$

  Die Herleitung ist in Anhang F gegeben.

Die Herleitung von Gleichung 4.15 zeigt, dass Überlast vermieden wird. Je nach aktueller Rate $\lambda_j(w)$ muss die Serverperiode $T_j(w)$ jeweils angepasst werden. Dabei werden die Verlustwahrscheinlichkeiten der Prozesse in $\boldsymbol{J}$ größer. Für alle $j \in \boldsymbol{J}$ wird gefordert, dass die aperiodische Prozessorganisation fair ist. Dies ist Gegenstand des folgenden Theorems.

Theorem 4.4. *Alle Prozesse $j \in \mathcal{J}$, deren Serverperiode gemäß Gleichung 4.15 verändert wurde, haben gleiche Verlustwahrscheinlichkeiten.*

Beweis. Vorausgesetzt wird, dass die Prozesse $j \in \mathcal{J}$, also die Prozesse, die die schnelleren Raten der I kompensieren, vor der Anpassung gleiche Verlustwahrscheinlichkeiten hatten, d.h.

$$\lambda_j T_j(w) = \lambda_{j+1} T_{j+1}(w) \tag{4.16}$$

Es ist zu zeigen, dass diese Beziehung auch für die angepassten Perioden $T_j(w+1)$ gilt. Gemäß der Prozesskooperation sind die λ_j unverändert.

$$\begin{aligned} \lambda_j T_j(w+1) &= \lambda_{j+1} T_{j+1}(w+1) \\ \lambda_j T_j(w) &= \lambda_{j+1} T_{j+1}(w) \qquad \text{m.Gl. 4.15} \end{aligned}$$

Dies entspricht der Voraussetzung. Die Anpassung der Prozesse j verändert die Beziehung aus Gleichung 4.16 nicht und es folgt sofort die Behauptung. □

Das Verhalten der kooperativen Prozesssteuerung ist in der Abbildung 4.30 illustriert. Es werden für 10 Prozesse zufällig Ereignisraten gewählt. Die Ereignisrate für Typ 1 vergrößert sich. Um die Verlustwahrscheinlichkeit für 1 zu halten, müssen alle anderen langsamer werden, d.h. ihr Serverperiode vergrößern. Es ist $p_1 \in I$ und $p_2 \ldots p_{10} \in \mathcal{J}$. Die Ergebnisse in Abbildung 4.30 (rechts) zeigen die Änderungen der Serverperioden (unterer Abschnitt) und die Verlustwahrscheinlichkeiten der einzelnen Prozesse (oberer Abschnitt). Für $p_1 \in I$ bleibt $P_{\lambda_1}^{\text{verlust}}$ vor und nach der Ratenänderung konstant. Wie zuvor bewiesen sind auch für $p_2 \ldots p_{10}$ die Verlustwahrscheinlichkeiten gleich.

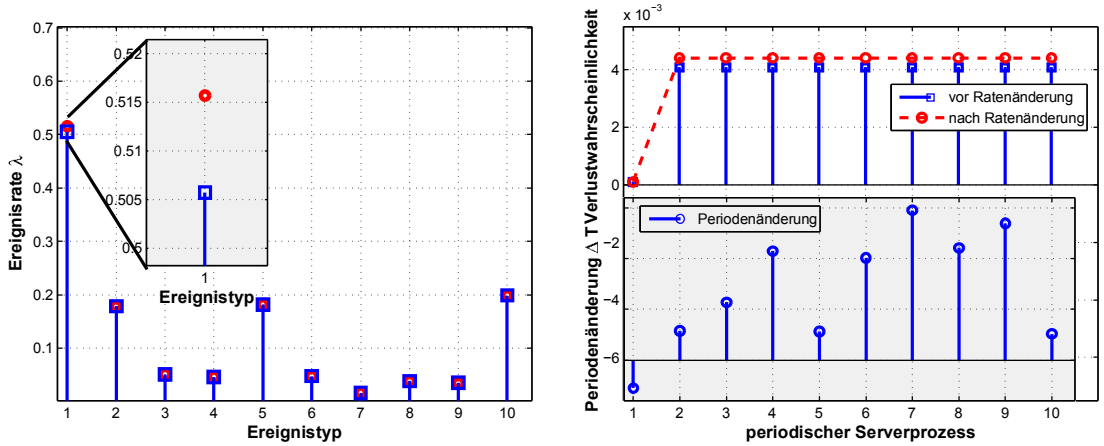


Abbildung 4.30: Kooperation aperiodischer Prozesse. Links: Raten der Ereignistypen für $p_1 \ldots p_{10}$. Die Rate für p_1 wird erhöht. Rechts: (unterer Abschn.) Periodenänderungen der Serverprozesse, p_1 wird beschleunigt. (oberer Abschn.) Verlustwahrscheinlichkeiten sind gleich für $p_2 \ldots p_{10}$ und bleiben konstant für p_1.

Als Resultat ist festzuhalten, dass alle Teilziele der Prozesskooperation unter den geforderten Bedingungen erreicht wurden. Die kooperative, aperiodische Prozessorganisation ermöglicht eine Ereignisbehandlung in unbekannten Einsatzszenarien. Der Kooperationsmechanismus stellt die Ausführung so ein, dass eine faire Behandlung

jedes einzelnen aperiodischen Prozesses erreicht wird und das Rechnersystem unter Ressourcenbeschränkungen kontrollierbar bleibt. Kooperation erfüllt die Problemstellung der aperiodischen Prozessorganisation aus Abschnitt 4.5.2.

Für alle Prozesse $p \in I$ erlaubt Kooperation eine feingranulare Priorisierung für besonders kleine Verlustwahrscheinlichkeiten. Kommunikationsprozesse mit Peripheriekomponenten oder Prozesse, die kurzzeitig eine Häufung von Ereignissen verarbeiten, sind typische Beispiele. Durch das kooperative Zusammenwirken von Prozessen der Menge $\mathcal{J}$ kann die Priorisierung getragen werden.

4.5.6 Analyse der Steuerbarkeit

Die Serverperiodenanpassung in Gleichung 4.15 der Prozesskooperation, verwendet zur Berechnung von $T_j(w+1)$ gemessene Informationen über $\lambda_i(w+1)$. Dies verletzt die Kausalität des Systems. Wenn $\lambda_i(w+2) = \lambda_i(w+1)$ gegeben ist, d.h. die Ereignisrate über mindestens zwei Serverperioden konstant bleibt, dann ist die Gleichung 4.15 in dieser Form auch für die Serverperiode $T_j(w+2)$ anwendbar. In Gleichung 4.15 wird dann $T_j(w+1)$ durch $T_j(w+2)$ ersetzt und Kausalität ist hergestellt. Diese Betrachtungen stellen die Frage nach der Steuerbarkeit des Systems.

Steuerbarkeit hinsichtlich eines vorgegebenen Ziels ist gegeben, wenn die Erfassung und Kompensation einer Störung vor Veränderung der Störgröße geleistet werden kann. Im aperiodischen Systemmodell aus Gleichung 4.13 muss die Störgröße λ erfassbar sein. Dies kann durch Auszählen innerhalb einer Serverperiode geleistet werden. Da die Erfassung der Ankunftsrate λ über einen bestimmten Zeitraum geschieht, und erst danach die Periode angepasst wird, ist die Steuerung *bandbreitenbegrenzt*. Um aperiodische Prozesse erfolgreich steuern zu können, muss das Samplingtheorem erfüllt sein. Es wird das Frequenzverhalten des gesteuerten Systems in Abbildung 4.29 untersucht: Der Systemausgang ist ein bandbreitenbegrenztes Signal B. Er gibt an wie schnell sich das System über die Zeit verändert. Die Bandbreite entspricht der Veränderung der Ankunftsrate λ. Das Steuerungsglied tastet λ mit der Frequenz $f = \frac{1}{T}$ ab. Das Steuerungssignal wirkt mit der gleichen Frequenz f auf das System. Um die Veränderung von λ *vollständig* zu erfassen und damit das System steuerbar zu gestalten, muss das Samplingtheorem $B \leq \frac{f}{2}$ erfüllt sein. Das heißt: Die Änderungsgeschwindigkeit der Ankunftsrate darf höchstens halb so schnell sein wie das Steuerungssignal. Oder anders ausgedrückt: Über den Zeitraum von mindestens zwei Serverperioden darf sich die Ankunftsrate nicht ändern.

Es ist zu erwähnen, dass bis zur vollständigen Erfassung der aktuellen Rate $\lambda(w)$ die vorgegebene Verlustwahrscheinlichkeit nicht garantiert werden kann. Während der transienten Phase der Steuerung ist die Verlustwahrscheinlichkeit $P^{\text{verlust}}_{\lambda(w)}(T(w-1))$. Es kann eine maximale Abweichung (engl. overshoot) $\hat{P}$ der Verlustwahrscheinlichkeit auftreten. Mit dem bekannten Steuerungsglied $R = \lambda(w)T(w) = \text{konst.}$ und der maximalen Steuerungsabweichung $\hat{R} = \lambda(w)T(w-1)$ ist $\hat{P} = -e^{-\hat{R}}\frac{\hat{R}^l}{l!} + e^{-R}\frac{R^l}{l!}$.

Mit den vorangegangenen Betrachtungen können Aussagen zu den Eigenschaften Stabilität, Schnelligkeit, Genauigkeit und Overshoot getroffen werden. Sie sind in Tabelle 4.6 zusammengefasst. Der Aufwand der Steuerung ist ebenfalls für ubiquitäre Rechnersysteme geeignet. Das Steuerungsglied in Gleichung 4.14 führt lediglich eine Division mit einer Konstanten aus. Der Skalierungsfaktor in Gleichung 4.15, der $T_j(w)$ auf $T_j(w+1)$ abbildet, ist konstant für *alle* Prozesse. Er muss nur bei Änderung

Eigenschaft	Systemverhalten	Interpretation
Stabilität	stabil, wenn mindestens $\lambda(w) = \lambda(w+1) =$ konst.	Rate der Ereignisse konstant über einen Zeitraum von mindestens zwei Serverperioden
Schnelligkeit	wenn System stabil, $P_{I/\mathcal{J}}^{\text{verlust}} =$ konst. nach $2T$	Für ein stabiles System ist gleiche Verlustwahrscheinlichkeit für Prozesse jeweils in I und $\mathcal{J}$ nach frühestens zwei Serverperioden gegeben.
Genauigkeit	wenn System stabil, dann Zielwert nach $2T$ erreicht	Statische Systeme erreichen Zielwert immer exakt.
max. Abweichung	System transient für $1T$, Zielwert wird um bis zu $\hat{P} = -e^{-\hat{R}}\frac{\hat{R}^l}{l!} + e^{-R}\frac{R^l}{l!}$ überschritten	Keine Garantie der geg. Verlustwahrscheinlichkeit für die Dauer einer Serverperiode bei Änderung der Ereignisrate.

Tabelle 4.6: Eigenschaften der aperiodischen Prozessorganisation

eines λ_i neu berechnet werden und kann dann für alle T_j angewendet werden. Der Aufwand ist linear.

4.5.7 M/D/k/k Warteschlangenmodell

Die Datenverarbeitung auf ubiquitären Rechnersystemen soll zeitnah erfolgen. Zeitliche Aussagen, zum Beispiel die mittlere Verweildauer von Ereignisdaten im Puffer, sind von großem Interesse. Dies wird mit einem M/D/k/k Warteschlangenmodell analysiert. Dabei werden Aufträge in eine Warteschlange eingeführt und von Bedienstationen verarbeitet. Die Bezeichnung leitet sich von der Kendall Notation [79] ab und bedeutet: M - Markovscher Ankunftsprozess (hier: ein Poissonprozess), D - deterministischer Bedienprozess (hier:periodisch), Anzahl der Bedienprozesse k, Kapazität der Schlange ist k. Das Warteschlangenmodell ist in Abbildung 4.31 abgebildet.

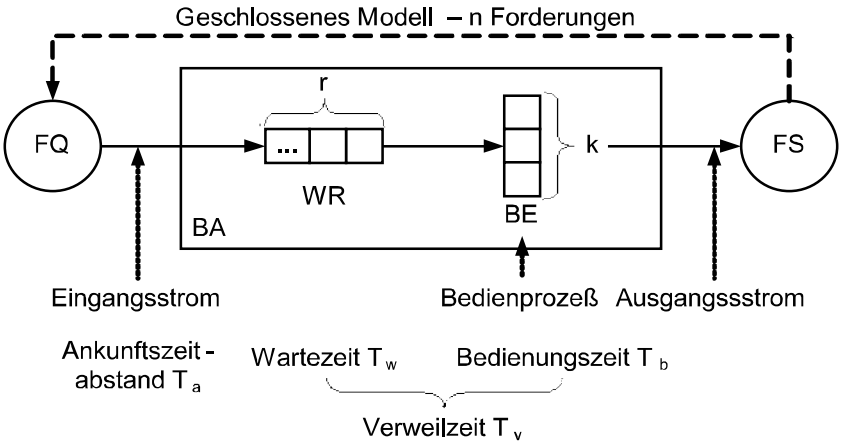


Abbildung 4.31: Warteschlangenmodell: BA:Bedienungsanlage, WR: Warteraum, BE: Bedienungseinrichtung, BK: Bedienungskanal, FQ: Forderungsquelle, FS: Forderungssenke (Quelle: [80]). Im M/D/k/k Modell ist die Kapazität des Warteraums $r = k$.

Es ist zu beachten, dass bei Ausführung des Servers alle Ereignisdaten im Puffer *in einem Stück* verarbeitet werden. Der Puffer wird bei Start des Servers vollständig entleert. Ereignisse, die zwischenzeitlich auflaufen, werden in der nächsten Periode verarbeitet. Dies wird durch k Bedienprozesse für eine Schlange mit der Kapazität k zum Ausdruck gebracht. Bisher wurde die Länge des Puffers mit l bezeichnet. Aufgrund der Kendall Notation wird implizit $k \equiv l$ angenommen. Für das M/D/k/k Modell der aperiodischen Prozessorganisation werden in Tabelle 4.7 die Bewertungsgrößen nach [80] aufgeführt. O.B.d.A. wird ein beliebiger aperiodischer Service herausgegriffen. Auf die Indizierung wird verzichtet. Die Bewertungsgrößen machen zeitliche Aussagen über die Ereignisse im Puffer. Dies erlaubt dem Entwickler die Leistungsfähigkeit kooperativer, aperiodischer Prozesse für den spezifischen Anwendungsfall einzuschätzen. Getrennt von Tabelle 4.7 aufgeführt ist die Formel 4.17 zur

Bewertungsgröße	**Wert**
Ankunftsrate	λ
Mittlerer Ankunftszeitabstand	$E[T_a] = \frac{1}{\lambda}$
Mittlere Bedienzeit	$E[T_b] = T$, T ist Serverperiode
Bedienrate	$\mu = \frac{1}{T}$
Ausführungszeit des Bedienprozess (Server)	C
Verkehrswert	$\rho = \frac{\lambda}{\mu} = \lambda T$
Erwartungswert der Warteschlangenlänge	$E[N_w] = \lambda T$
Maximal Wartezeit	$\max T_w = \frac{1}{\mu} = T$
Maximale Verweilzeit	$\max T_v = \frac{1}{\mu} + C = T + C$
Mittlere Wartezeit	siehe Formel 4.17

Tabelle 4.7: Bewertungsgrößen des M/D/k/k Modells der aperiodischen Prozessorganisation

Berechnung der mittleren Wartezeit von Ereignissen in einem endlichen Ereignispuffer. Die Herleitung ist im Anhang G gegeben.

$$E[T_w] = \frac{(\lceil \lambda T \rceil - \lambda T) \sum_{i=1}^{\lfloor \lambda T \rfloor} \frac{1}{\lambda} i + (\lambda T - \lfloor \lambda T \rfloor) \sum_{i=1}^{\lceil \lambda T \rceil} \frac{1}{\lambda} i}{\lambda T} \tag{4.17}$$

4.5.8 Speichereffizienz

Bisher wurden nur die Verlustwahrscheinlichkeit, also das Auftreten eines Pufferüberlaufs, diskutiert. Ein kleine Verlustwahrscheinlichkeit ist gewünscht. Die Prozesskooperation kann sicher stellen, dass eine vorgegebene (kleine) Verlustwahrscheinlichkeit für einige Prozesse eingehalten und Überlast generell vermieden wird. Aber neben einer hohen Serverauslastung als Folge, bleibt auf der anderen Seite auch Pufferkapazität ungenutzt. In diesem Abschnitt wird die Speicherausnutzung der Ereignispuffer analysiert. Dies erlaubt dem Entwickler die Speicherressourcen eingebetteter ubiquitärer Rechnersysteme gegen die Qualitätsanforderungen der Ereignisverarbeitung abzuwägen.

Als Metrik wird die Speichernutzungseffizienz angeführt. Sie wird aus dem Verhältnis der erwarteten Ereignisanzahl λT in einer Serverperiode und der Pufferlänge l gebildet. Die Abbildung 4.32 zeigt den Zusammenhang zwischen gewählter Verlustwahrscheinlichkeit für einen Ereignispuffer und der Speichereffizienz. Die Abbildung 4.32

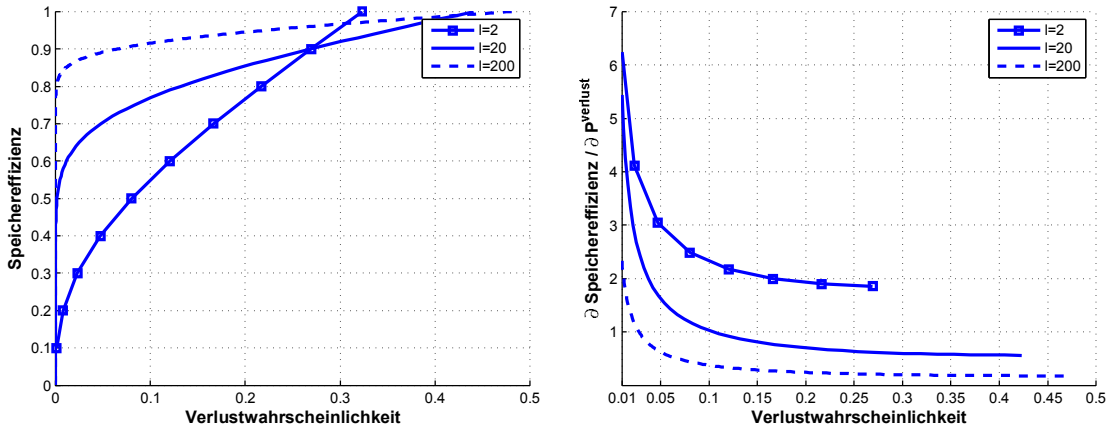


Abbildung 4.32: Speichereffizienz für drei aperiodische Prozesse mit $\lambda = 0.2$ und Pufferkapazitäten von $l = 2, 20, 200$. Links: Speichereffizienz $\lambda T/l$, Rechts:Änderung der Speichereffizienz in Abhängigkeit von P^{verlust}. ∂Speichereffizient $\to \infty$ für sehr kleine P^{verlust}.

zeigt, dass die Speichereffizienz dramatisch sinkt, wenn die Verlustwahrscheinlichkeit klein wird. Umgekehrt kann eine sehr gute Speichernutzung erzielt werden, wenn die Verlustwahrscheinlichkeit relativ hoch angesetzt wird. Überraschend ist, dass die Speichereffizienz gesteigert wird, wenn für kleine Verlustwahrscheinlichkeiten mehr Speicher investiert wird. Es erlaubt langsamere Serverperioden, was Gelegenheit gibt, Puffer stärker zu befüllen. Bei noch höheren Verlustwahrscheinlichkeiten kehrt sich der Effekt um und kleinere Puffer werden vorteilhafter.

Ein zweiter Effekt ist, dass bei größerem Speicheraufwand und kleiner werdender Verlustwahrscheinlichkeit die Effizienzänderung *extrem* groß wird. Prozesse, die kooperieren, um Überlastsituationen zu vermeiden, unterliegen schwankenden Verlustwahrscheinlichkeiten. Kleine Abweichungen von der vorgegebenen Verlustwahrscheinlichkeit können plötzlich eine massive Verschwendung von Pufferkapazitäten bedeuten. Die Abwägung von Pufferkapazität und Verlustwahrscheinlichkeit hängt von den vorhandenen Speicherressourcen ab, wie auch von der Anzahl und Ausführungsdauer der Prozesse und der Auslastungsgrenze für die Ereignisverarbeitung. Die letzteren Bedingungen haben insbesondere Einfluss auf Schwankungsbreite der Verlustwahrscheinlichkeit.

4.5.9 Praktisches Vorgehen

Beim Einsatz der kooperativen, aperiodischen Prozessorganisation sind sowohl Auslastungsbegrenzung wie auch Verlustwahrscheinlichkeit vom Entwickler in der Entwurfsphase festzulegen. Es sind zwei Vorgehensweisen anwendbar: auslastungsorientiert oder speicherorientiert. Beide Verfahren sind von Ereignisrate zur Laufzeit unabhängig. Kooperation passt die Prozessausführung entsprechend der aktuellen Bedingungen und unter Einhaltung der Entwurfsparameter an.

Auslastungsorientierter Entwurf

Es sind die Anzahl der aperiodischen Prozesse $N = |\mathcal{P}^{\text{aP}}|$, die Auslastungsgrenze U_b und die Verlustwahrscheinlichkeit $P_{\lambda_i}^{\text{verlust}}$ bekannt. Eine Ereignisrate λ_i, zum Beispiel abgeleitet aus Erfahrung oder Messung, wird angenommen. Gesucht ist die Größe l der Pufferspeicher.

1. **Verteilung der Auslastung U_b:** $\forall i \in \mathcal{P}^{\text{aP}} : \frac{U_b}{N} = \frac{C_i}{T_i}$ gdw. $T_i = \frac{C_i}{U_b} N$
2. **Bestimmung der Puffergröße l:** Finde durch Iteration das kleinste l mit $\forall i \in \mathcal{P}^{\text{aP}} : 1 - P_{\lambda_i T_i}(l) \leq P_{\lambda_i}^{\text{verlust}}$

Bemerkung: Es kann passieren, dass l sehr groß wird, wenn $P_{\lambda_i}^{\text{verlust}}$ sehr klein gewählt wurde.

Speicherorientierter Entwurf

Es sind die Anzahl der aperiodischen Prozesse $N = |\mathcal{P}^{\text{aP}}|$, die Größe des zur Verfügung stehenden Speichers L und die Verlustwahrscheinlichkeit $P_{\lambda_i}^{\text{verlust}}$ bekannt. Eine Ereignisrate λ_i, zum Beispiel abgeleitet aus Erfahrung oder Messung, wird angenommen. Gesucht ist die Auslastungsgrenze U_b.

1. **Verteilung des Speichers L:** Es ist $l = \frac{L}{N}$. Bestimme T_i mit $1 - P_{\lambda_i, l}(T_i) \leq P_{\lambda_i}^{\text{verlust}}$
2. **Bestimmung der Auslastungsgrenze U_b:** Berechne die notwendige Auslastung $U_b = \sum_i \frac{C_i}{T_i}$.

Bemerkung: Es kann passieren, dass $U_b > 1$ ist. Das Verfahren muss dann mit einer größeren Puffergröße wiederholt werden.

4.5.10 Vergleichende Betrachtungen

In diesem Abschnitt wird ein Vergleich mit dem ereignisgetriebenen Betriebssystem TinyOS geführt. Außerdem wird die kooperative aperiodische Prozessorganisation mit bekannten Verfahren gegenübergestellt. Dabei werden auch Aspekte des (störenden) Einflusses periodischer Prozesse auf die Serverprozesse betrachtet. Eine untere Schranke der garantierten Datenverarbeitung wird angegeben.

TinyOS

TinyOS [13] ist ein ein ereignisgetriebenes Betriebssystem für drahtlose Sensorknoten. Applikationen sind aus Komponenten aufgebaut und werden durch Benachrichtigung mit dedizierten Ereignissen zur Ausführung gebracht. Alle Komponenten laufen ununterbrechbar. Die Ausführungsreihenfolge ist gemäß einer FIFO Strategie organisiert. TinyOS verarbeitet sowohl periodische Prozesse, also regelmäßige Zeitgeberereignisse, wie auch aperiodische Ereignisse in einer einzigen FIFO Schlange. Jedes Ereignis wird einzeln behandelt. Im Vergleich dazu verarbeitet die gesteuerte, aperiodische Prozessorganisation Ereignisse am Stück. Für jeden Ereignistyp gibt es einen dedizierten Puffer. Tabelle 4.8 stellt die Organisationsformen einander gegenüber. Es wird die Ereignisverarbeitung in Bezug auf einen aperiodischen Ereignistyp untersucht. Es wird das Antwortzeitverhalten beider Ansätze verglichen.

$$N(C^{aP} + MC^P) \stackrel{?}{\geq} NC^{aP} + T^{aP}$$

$$NC^{aP} + NMC^P \stackrel{?}{\geq} NC^{aP} + T^{aP}$$

$$\text{Es folgt: } \exists N^* : \; N > N^* \Rightarrow NMC^P > T^{aP} \tag{4.18}$$

Gleichung 4.18 gibt die Bedingung an, unter der durchschn. Antwortzeit$^{\text{TinyOS}}$ > durchschn. Antwortzeit$^{\text{aperiodisch}}$ gilt. Es zeigt, dass die Aufteilung der Verarbeitung in dedizierte Puffer der Ereignistypen zu einer Verbesserung des Antwortzeitverhaltens gegenüber TinyOS führen kann.

	TinyOS	**aperiodische Prozessorganisation**
Grundlast durch periodische Prozesse	M [Ereignisse]	-
Durchschnittliche Ausführungsdauer der periodischen Ereignisse [Takte]	C^P	-
Durchschnittliche Verweilzeit periodischer Ereignisse in FIFO Schlange [Takte]	MC^P	-
Durchschnittliche Berechnungszeit für ein aperiodisches Ereignis [Takte]	C^{aP}	C^{aP}
Durchschnittliche Antwortzeit für ein aperiodisches Ereignis [Takte]	$C^{aP} + MC^P$	$\lambda C^{aP} + T^{aP}$
Durchschnittliche Antwortzeit für ein N Ereignisse [Takte]	$N(C^{aP} + MC^P)$	$NC^{aP} + T^{aP}$

Tabelle 4.8: Gegenüberstellung der Ereignisverarbeitung von TinyOS und der aperiodischen Prozessorganisation dieser Arbeit.

Aperiodische Server mit Echtzeitfähigkeiten

Buttazzo unterscheidet in [55] Serververfahren mit festen und dynamischen Prioritäten zur Behandlung von aperiodischen Ereignissen. Gemeinsames Ziel der Verfahren ist es, die Antwortzeit zu minimieren und Echtzeitgarantien für die Ereignisverarbeitung zu geben. Dabei werden nur die Serverprozesse ausgeführt, die die Garantien einhalten. Als Folge kann es passieren, dass bestimmte Ereignistypen gar nicht mehr verarbeitet werden.

In der kooperativen, aperiodischen Prozessorganisation dieser Arbeit soll *jeder einzelne Serverprozess* eine hohe Anzahl von für ihn dedizierten Ereignissen verarbeiten. Faire Gestaltung dieser Aufgabe bei Vermeidung von Überlast und unter vorgegebenen Speicher- und Rechenzeitressourcen stehen im Vordergrund. Alle Serverprozesse müssen datenechtzeitfähig sein, um kooperatives Verhalten zu ermöglichen. Andere periodische Prozesse können die Ausführung der Serverprozesse um bis zu $\max C_{p_m}^{\text{per.Proz.}} + \max C_{p_n}^{\text{aper.Proz.}}$ verzögern. Diese Daten können über mehrere Ausführungsperioden ermittelt und in der Periodenberechnung berücksichtigt werden.

Im Extremfall kann jedoch jeder periodische Prozess i alle anderen so verzögern, dass deren Prozessqualität $\forall k, k \neq i : Q^P(p_k) = 0$. Die Datenechtzeitfähigkeit periodischer Serverprozesse ist dadurch nicht mehr erfüllt. Gleiche Verlustwahrscheinlichkeiten können nicht mehr garantiert werden und kooperatives Verhalten wird nicht erreicht. Die Prozessorganisation ubiquitärer Systeme degradiert zu einer FIFO Schlange. Kooperation und Kollaboration der periodischen Prozessorganisation garantieren jedoch, dass kein Prozess permanent ausgelassen wird. Alle Prozesse werden fair behandelt mit einer gemeinsamen Ausführungsperiode ausgeführt. Sie beträgt im Durchschnitt für alle $N = |\mathcal{P}|$ Prozesse

$$T = \frac{1}{N} \sum_{i=1}^{N} T_i + \sum_{k \in \mathcal{P}, k \neq i} C_k \tag{4.19}$$

Dies ist die untere Schranke, engl. bottom line, der Antwortzeit des Rechnersystems bei der Datenverarbeitung, die Kooperation und Kollaboration für alle periodischen

und aperiodischen Prozesse garantieren kann. Die Verlustwahrscheinlichkeiten der aperiodischen Prozesse ergeben sich gemäß Gleichung 4.12 für diese Serverperiode. Es wird nicht übersteuert, da die Steuerung in Gleichung 4.14 den Erwartungswert λ_i und nicht den tatsächlichen Verlust bestimmt.

4.6 Implementierung Particle OS

Particle OS ist ein Betriebssystem für die Particle Computer Plattform [19]. Es implementiert den Entwurf der Laufzeitumgebung aus Abschnitt 3.5.3 und stellt zusätzliche Entwicklungsunterstützung für ubiquitäre Anwendungen bereit. Particle OS unterstützt serviceorientierte, periodische Prozesse und Prozessgraphen und implementiert das geregelte Kooperations- und Kollaborationsverhalten für eine faire Prozessorganisation.

4.6.1 Architektur und Komponenten

Die Architektur des Betriebssystem ist in Abbildung 4.33 dargestellt. Particle OS

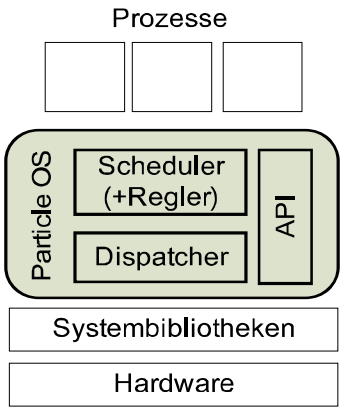


Abbildung 4.33: Systemarchitektur des Particle OS

ist für 1-Prozessor Systeme mit sehr beschränkten Speicher und Rechenressourcen entwickelt worden. Die Aufgaben sind Prozessor- und Prozessmanagement. Die Laufzeitumgebung mit Scheduler, Regler und Dispatcher führt die Prozesse aus. Interprozesskommunikation wird durch gemeinsame Speicherbereiche realisiert, auf die nicht-präemptive Prozesse nacheinander zugreifen.

Prozesse sind nicht Teil des Particle OS, sondern anwendungsspezifische Teile. Prozesse können auf Funktionen der Systembibliotheken zugreifen, wenn sie in der Laufzeitumgebung des Particle OS ausgeführt werden. Die Bibliotheken stellen eine einfache Schnittstelle zu den Plattformfunktionen bereit.

Prozesse, Services und OS sind in der Programmiersprache C geschrieben und werden durch den Compiler in ausführbaren Code für den PIC18F6720 Mikrocontroller der Particle Plattform übersetzt. In dieser nativen Implementierung sind Laufzeitumgebung und Prozesse sehr eng gebunden. Die Tabelle 4.9 erklärt Komponenten und Funktionalitäten des Particle OS. Weitere Aufgaben eines Betriebssystems sind Speicher- und Gerätemanagement. Speichermanagement wird durch Compiler realisiert. Der Compiler analysiert die Systemsoftware und Prozesse und alloziert den

Komponente	Funktion
Scheduler	Es wird ein prioritätsbasiertes Prozessscheduling unterstützt. Die Schedulingkomponente ist austauschbar und erlaubt den Einsatz von echtzeitfähigen Schedulingstrategien.
Regler	Für Kooperation und Kollaboration wird FQS als Scheduler eingesetzt. FQS arbeitet prozessbezogen wie ein P-Regler (Abschnitt 4.4.6). Auch der Zwei-Punkt-Regelmechanismus für Kollaboration ist im Scheduler implementiert.
Dispatcher	Periodisch werden die Prozesse durch den Dispatcher ausgeführt. Die Verwaltung der Systemzeit treibt das gesamte Laufzeitsystem voran.
API	Die Schnittstelle zur Anwendungsprogrammierung stellt Funktionen zur Erzeugung von Services, Prozessen und dem Bekanntmachen der Prozesse im Laufzeitsystem bereit. Strukturen zur Verwaltung der Service- und Prozessparameter werden angeboten. Scheduler und Dispatcher verwenden das API, um Prozesse zu adressieren und deren Laufzeitparameter zu verändern.

Tabelle 4.9: Komponenten des Particle OS

Speicher statisch. Durch Optimierung können Speicherbereiche mehrfach verwendet werden. Gerätemanagement zur Verwendung peripherer Komponenten wie Sensoren und der Kommunikationseinheit wird durch Treibersoftware in den darunterliegenden Systembibliotheken realisiert.

4.6.2 Java Virtuelle Maschine

Ubiquitäre Rechnersysteme sind nicht auf eine unveränderliche Menge von Prozessen festgelegt. Services und Prozesse sollen austauschbar sein. In der nativen Implementierung sind Particle OS und Services eng gebunden. Der Austausch von übersetzten Programmteilen zur Laufzeit erfordert relokierbaren Binärcode, der auf der Plattform zur Laufzeit durch einen Linker neu gebunden werden muss. Realisierte Verfahren sind in [81] und [82] beschrieben.

Für diese Arbeit wurde eine Java virtuelle Maschine gewählt, die *kompakten* Bytecode ausführt. Die Java virtuelle Maschine wurde im Rahmen der Diplomarbeit von Andreas Arnold [83] für die Particle Computer Plattform entwickelt. Services sind einzelne Java Klassen, die zur Laufzeit per drahtloser Kommunikation in den Speicher des Particle Computer geladen werden. Das Particle OS Laufzeitsystem vermag die Java-Services in gleicher Weise zu behandeln wie native Services. Zu jedem Java Service existiert ein sogenannter Spiegel-Service oder Stellvertreter, der durch das Particle OS API verwaltet wird. Dies ermöglicht die nahtlose Einbindung von Java-Prozessen in das native Laufzeitsystem. Die Abbildung 4.34 illustriert die Verbindung. Der Java Service überträgt mittels der Funktion `register()` Konfigurationsparameter und eine seine eigene Referenz auf den Spiegel-Service. Bei Ausführung ruft der Dispatcher dann durch die `execute()` Funktion die virtuelle Maschine mit dieser Referenz auf und die Servicefunktion startet. Sowohl native Services wie

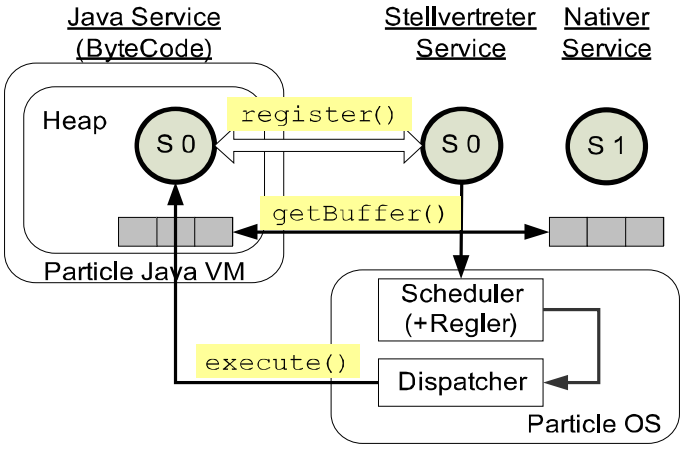


Abbildung 4.34: Verbindung der Java virtuellen Maschine mit dem Particle OS

auch Java Services können auf die jeweils anderen Puffer verlinken. Dafür existiert im Particle OS API eine `getBuffer()` Funktion. Durch kopieren überträgt sie den Pufferinhalt des nativen Service auf den Heap der Java virtuellen Maschine. Der Java Service kann im Anschluss auf die Daten zugreifen. Das Speichermanagement erfolgt durch dynamische Allokation durch die virtuelle Maschine.

4.6.3 Entwicklungsunterstützung

Die Erstellung von und die Arbeit mit serviceorientierten Prozessen für eingebettete, ubiquitäre Rechnersysteme wird durch eine Werkzeugkette für Desktop-Rechner unterstützt. Servicegraphen werden durch das Graphenwerkzeug Graphviz [84] modelliert. Die Werkzeugkette erlaubt die Entwicklung in beide Richtungen (engl. roundtrip engineering). Es können aus der graphischen Repräsentation Codegerüste erzeugt werden. Umgekehrt kann aus einem laufenden Prozesssystem heraus ein Servicegraph ausgelesen, angezeigt, modifiziert und wieder dem System übergeben werden. Alle Werkzeuge wurden in der vom Autor betreuten Diplomarbeit von Andrea Sayer [78] implementiert und mit der Particle Computer Plattform getestet. Der Entwickler wird in drei Phasen unterstützt:

Entwurf nativer Services und Prozesse: Mit dem Graphviz Werkzeug werden Services und Prozesse in einem Servicegraphen entworfen und mit Periode und Qualitätsbudget parametrisiert. Aus der Beschreibung werden C Codegerüste für die Services generiert. Unter Zuhilfenahme des Particle OS API wird Code zum Erstellen und Bekanntmachen der Prozesse im Laufzeitsystem erzeugt. Services und Prozesse werden zusammen mit dem Particle OS und den Systembibliotheken kompiliert und mittels Flash-Programmierer auf den Particle Computer transferiert. Die Prozesse aus dieser Phase sind zur Laufzeit unveränderlich. Beispiele sind Sensor-Sampling und Kommunikationsprozesse.

Veränderung der Prozesskonfiguration zur Laufzeit: Die aktuelle Konfiguration eines Particle Computer wird ausgelesen und der Servicegraph mittels des Graphviz Werkzeuges angezeigt. Neue Services und Prozesse können eingebracht werden. Optional werden gemäß Abschnitt 4.3.2.2 Servicegraphen transformiert, um die Datenverarbeitung mit mehreren Quellen zu ermöglichen. Neue und veränderte Services und Prozesse werden zur Laufzeit per

drahtloser Kommunikation an den Particle Computer übertragen. In dieser Phase werden nur Java Services unterstützt. Particle OS ermöglicht die einheitliche Integration mit bestehenden nativen Services und Prozessen.

Simulation: Anstelle der Particle Computer Hardware kann eine Systemsimulation eingesetzt werden. Diese Phase ermöglicht die Erfassung von Leistungsdaten und unterstützt die Suche nach Programmfehlern. In der Diplomarbeit von Yusuf Iskenderoglu [85] wurde ein zyklengenauer PIC Hardwareemulator vorgestellt. Der Emulator wurde in die Werkzeugkette integriert und simuliert das gesamte Particle OS einschließlich eingebrachter Prozesse.

Die Abbildung 4.35 zeigt alle Werkzeuge im Überblick. Die Werkzeuge in Interface generieren Code, transformieren Graphen und bereiten die Darstellung auf. Der ServiceManager kommuniziert mit dem Particle Computer und liest den aktuellen Servicegraphen aus. PVMTool und PVMControl organisieren den Transfer von Java Services auf den Particle Computer.

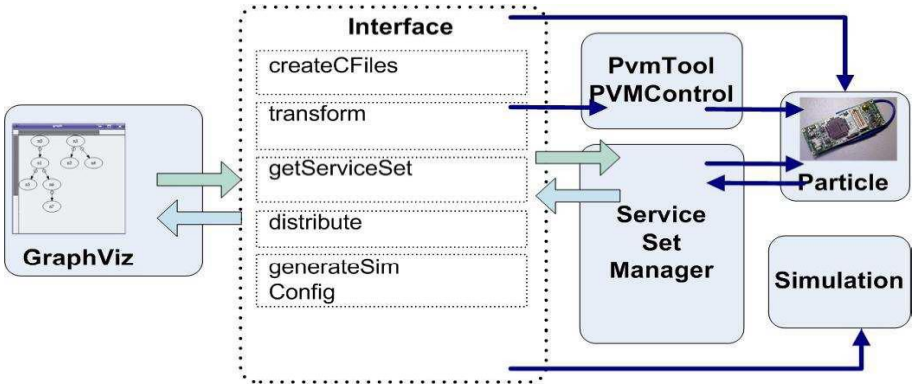


Abbildung 4.35: Werkzeuge für die Entwicklung von serviceorientierten Prozessen mit Particle OS (Quelle: [78]).

4.6.4 Kennzahlen

Die begrenzten Speicher- und Rechenressourcen sind die limitierenden Faktoren für Softwareentwicklungen auf eingebetteten, ubiquitären Rechnersystemen. Die Referenzplattform Particle Computer verfügt über 128 Kilobyte Programmspeicher und 4 Kilobyte Datenspeicher. Insbesondere der Speicherverbrauch und die Ausführung der Betriebssystemumgebung sind daher von großem Interesse und werden durch Kennzahlen erfasst.

Die Implementierung der Betriebssystemumgebung auf der Particle Computer Plattform umfasst die Systembibliotheken, als ParticleBaseSystem (PBS) bezeichnet, das Particle OS und die Java virtuelle Maschine ParticleVM. Die drei Bestandteile belegen mit 99, 8% den Programmspeicher quasi vollständig. Das Speichermanagement ist statisch und wird zur Kompilierzeit festgelegt. Es steht damit sehr wenig Speicher für das Anlegen und die Ausführung von Prozessen zur Verfügung. Die Betriebssystemumgebung ist auch ohne virtuelle Maschine voll funktionsfähig. Diese Kombination stellt wesentlich mehr Speicherressourcen für Prozesse zur Verfügung. Jedoch kann Particle OS dann lediglich unveränderliche, native Prozesse verwalten.

Bestandteile	**Codegröße** [bytes]	**Datengröße** [bytes]	**Eigenschaften**
ParticleBaseSystem (PBS)	25954 (19,8%)	697 (17,71%)	nur Plattformzugriff, keine Prozessunterstützung
PBS + ParticleVM	104770 (79,93%)	3334 (84,71%)	Austausch von Klassen zur Laufzeit, keine Prozessunterstützung
PBS + Particle OS	49112 (37,47%)	1353 (34,38%)	koop. und kollab. Prozessorganisation durch FQS, nur unveränderliche, native Prozesse möglich
PBS + Particle OS + VM	130815 (99,8%)	3620 (91,98%)	koop. und kollab. Prozessorganisation von nativen und Java Services durch FQS
Particle OS			
FQ Scheduler	6592 (5,03%)	8 (0,22%)	
Dispatcher	2656 (2,03%)	17 (0,47%)	
API	13910 (10,61%)	631 (15,98%)	

Tabelle 4.10: Speicherbedarf der Bestandteile der Betriebssystemumgebung. Der Großteil der Prozess- und Zeitgeberinformationen wird durch das API verwaltet, um Scheduler, Dispatcher und Regler leichtgewichtig und austauschbar zu halten.

Die Speicherbelegungen der Bestandteile der Betriebsumgebung sind für die aktuelle Implementierung auf der Particle Computer Plattform in der Tabelle 4.10 angegeben. Die Zahlen wurden in [78] ermittelt. Zusätzlich sind Eigenschaften der Prozessausführung für jede Kombination angegeben. Das Zusammenfassen von Services in Prozessen reduziert den Rechenzeitaufwand des Particle OS, da Betriebssystemfunktionen nur einmal pro Prozess aufgerufen werden. Tabelle 4.11 vergleicht den Rechenzeitaufwand für die Einplanung und Ausführung von Services und Prozessen. Im Fall der prozessorientierten Ausführung weist `dispatch()` durch Qualitätsberechnungen und Zugriffe auf die Prozessdatenstruktur eine größere Laufzeit auf. Die enorme Einsparung in `schedule()` führt letztendlich zur einer signifikaten Reduktion des Rechenaufwandes des Laufzeitsystems.

Java Bytecode reduziert die Größe von Programmen. Die kompaktere Darstellung erlaubt eine Speicherreduktion um Faktor 2 bis 3 im Vergleich zu nativen 8-bit Code [83]. Die ParticleVM ist jedoch durch Interpretation um Faktor 30 langsamer als die native Ausführung von Services. Der Entwickler muss Austauschbarkeit der Services und Prozesse und Geschwindigkeit für die jeweilige Applikation abwägen.

4.7 Anwendung AwarePen

Das Anwendungsbeispiel AwarePen illustriert, wie die kooperative und kollaborative Prozessorganisation die zeitnahe Verarbeitung aktueller Situationsinformationen,

Funktion	**Rechenzeit** [Zyklen]	
	Service	Prozess
`dispatch()`	2140	2472
`schedule()`(FIFO)	Service 1: 192	Prozess mit 5 Services: 504
	Service 2: 215	
	Service 3: 241	
	Service 4: 267	
	Service 5: 293	
	gesamt: 1208	gesamt: 504

Tabelle 4.11: Rechenzeitaufwand des Schedulers und Dispatchers bei Einplanung und Ausführung. Für den Prozess muss der Scheduler nur einmal aufgerufen werden. Der Aufwand zur Einplanung reduziert sich signifikant. (Quelle: [78])

sogenannte Kontexte, ermöglicht. Es wird gezeigt, dass Prozesse, die Informationen zum aktuellen Kontext liefern, durch die Prozessorganisation in Particle OS bevorzugt werden. Kooperation kann eine *kontextsensitive Prozesseinplanung* ermöglichen. Die Appliance AwarePen ist ein Stift mit einem integrierten, ubiquitären Rechnersystem. Er erfasst die Interaktion seines Benutzers und die Umgebungssituation. Diese Informationen unterstützen Anwendungen wie automatische Meeting-Annotationen und Steuerung von Geräten [77]. Periodisch werden Daten von zwei Beschleunigungssensoren am Stift akquiriert, in mehreren Prozessen gefiltert und zu Aktivitätskontexten wie SCHREIBEN, LIEGEN und SPIELEN verarbeitet. Erfassung der Sensorinformation und Bestimmung des aktuellen Aktivitätskontextes sollen zeitnah erfolgen. Vorangegangene Arbeiten [4][86][78] haben bereits die Prozesse zur Informationsverarbeitung definiert. Durch Filter werden Sensorinformationen der periodisch abgefragten Beschleunigungssensoren aufbereitet und verarbeitet. Die Abbildung 4.36 zeigt den Filtergraphen aus dem vorangegangenen Arbeiten und seine Umsetzung als Servicegraph für die Prozessausführung in Particle OS.

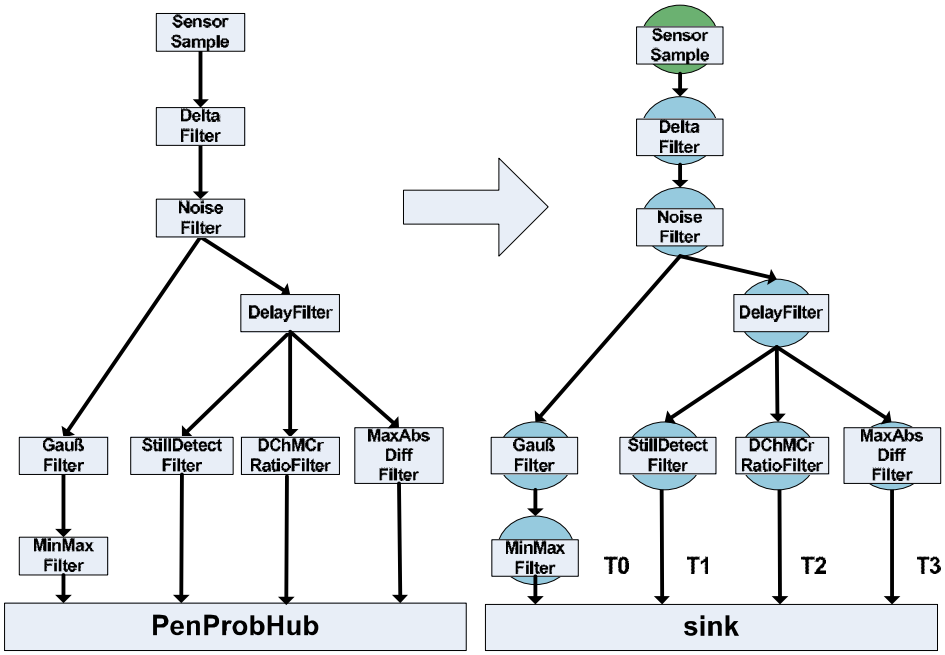


Abbildung 4.36: Filtergraph zur Datenverarbeitung des AwarePens und die Umsetzung in Prozesse des Particle OS. Die Prozesse $T0 \ldots T3$ entstehen. (Quelle:[78]).

Es können vier Prozesse $T0 \ldots T3$ identifiziert werden. Die Komponenten *PenProb-Hub* beziehungsweise *sink* markieren die für jeden Prozess separate Entscheidungslogik zur die Weitergabe des ermittelten Aktivitätskontextes an Anwendungen in der Umgebung des AwarePen. Der Prozess $T1$ ist für die Erkennung des Kontextes LIEGEN verantwortlich. $T0, T2$ und $T3$ für Erkennung von SCHREIBEN und SPIELEN. Durch den rückgekoppelten Regelungsmechanismus führt Particle OS die Prozesse zur Erfassung und Bereitstellung der Kontextinformationen abhängig von der Umgebungssituation aus. Abbildung 4.37 zeigt die Ablaufreihenfolge der Prozesse während der zwei Kontexte LIEGEN und SCHREIBEN. Im Kontext LIEGEN brechen

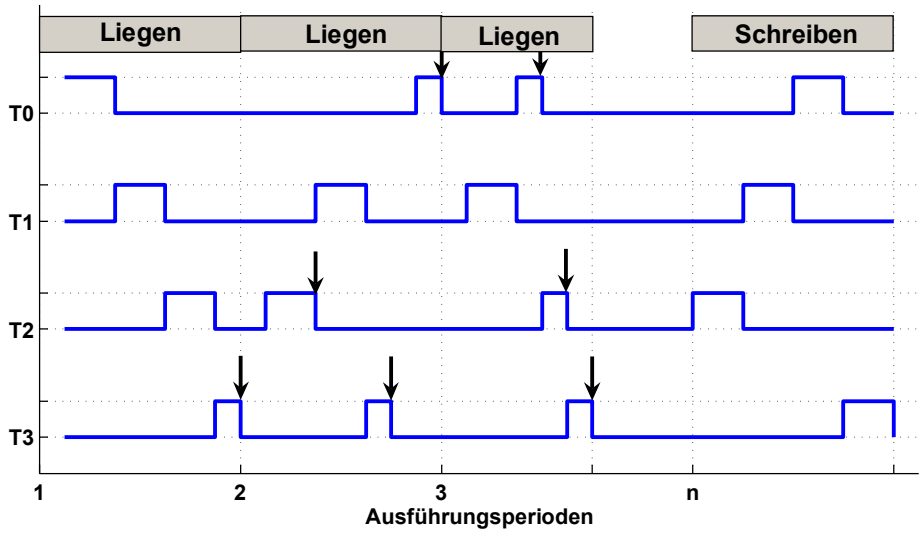


Abbildung 4.37: Ablaufreihenfolge der Prozesse während der Kontexte LIEGEN und SCHREIBEN. Pfeile markieren Abbrüche der Prozesse, die im jeweiligen Kontext nicht ausgeführt werden müssen. In der dritten Ausführungsperiode brechen alle Prozesse bis auf $T2$ ab, der daraufhin bevorzugt wird. In der n-ten Periode ist eine Reihenfolge für den Kontext SCHREIBEN angegeben. Abbildung ist zeitlich skaliert, um unterschiedliche Ausführungsdauern deutlicher darzustellen.

die Prozesse $T0, T2$ und $T3$ nach kurzer Laufzeit von sich aus ab, indem ein Service des Prozesses auf Basis der Ergebnisse des Vorgängerservice keine Ausgabedaten produziert. Dies ist vom Entwickler innerhalb eines Filterservice zu implementieren. Datenechtzeit ist gewährleistet, aber durch den Abbruch werden die Prozesse mit $Q_0^P = Q_2^P = Q_3^P = -1$ bewertet. $T1$ zur Erkennung des Kontextes LIEGEN ist erfolgreich. Er wird durch zuvor abgebrochene Prozesse um $D_{0,2,3}$ verzögert und erreicht $Q_1^P = 1 - \frac{D_{0,2,3}}{T}$.

Durch Prozessabbruch in Verbindung mit dem Kooperationsmechanismus in FQS kann eine Partitionierung der Schedulerwarteschlange entstehen. Es erfolgt eine Unterteilung in ausgeführte und abgebrochene Prozesse. Die erste Gruppe wird vom Scheduler bevorzugt und erhält immer höhere Priorität. Im Kontext LIEGEN bevorzugt die Prozessorganisation den Prozess $T1$. Er wird an ersten Stelle im Ablaufplan eingeplant. Abbildung 4.38 (links) zeigt die Warteschlange nachdem Kontext LIEGEN erkannt wurde. Der Kontext SCHREIBEN wird durch die Prozesse $T0, T2, T3$ erkannt. Diese Prozesse werden nicht abgebrochen. Da für die Erkennung des Kontextes LIEGEN nur noch ein weiterer Service ausgeführt werden muss, wird $T1$ ebenfalls

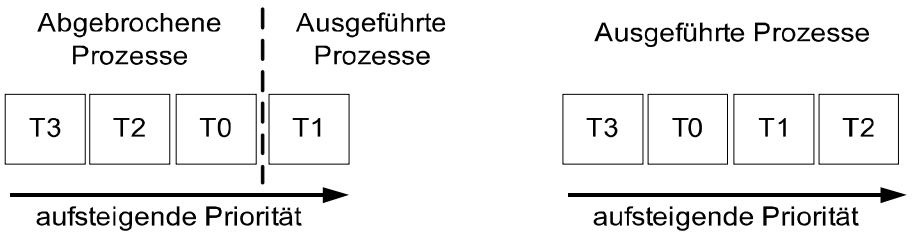


Abbildung 4.38: Warteschlange von FQS gemäß der Ausführungsreihenfolge in Periode 3 und n der Abbildung 4.37. Links: Paritionierung in ausgeführten Prozess $T1$ und abgebrochene $T0, T2, T3$ während des Kontext LIEGEN. Rechts: Eine mögliche Reihenfolge im Kontext SCHREIBEN. Es tritt keine Partitionierung auf.

nicht abgebrochen. In diesem Fall entsteht keine Partitionierung. Abbildung 4.38 (rechts) zeigt eine mögliche Prozessreihenfolge.

Es ist darauf hinzuweisen, dass das FQS Kooperationsziel $\forall i : \ B_i(w) = B_{i+1}(w)$ in jedem Fall erreicht wird. Im Fall von Partitionierung der Warteschlange bei der Erkennung des Kontextes LIEGEN ergeben sich die Budgetänderungen aller Prozesse zu $\Delta B_i(w) = B_i(w) - \left|Q_i^P(w)\right| = -1$. Deshalb ist die Partitionierung stabil bis zum nächsten Kontextwechsel. Ändert sich der Kontext und die Warteschlange ist nicht mehr partitioniert, dann benötigt FQS gemäß den Eigenschaften aus Tabelle 4.3 $w \approx 6$ Zeitschritte, um das Kooperationsziel zu erreichen. Der Kollaborationsmechanismus stellt sicher, dass der Regler für Kooperation im Arbeitsbereich bleibt.

Das Anwendungsbeispiel zeigt, dass Prozesse, die für den aktuellen Kontext Informationen liefern, durch die kooperative Prozessorganisation in FQS bevorzugt werden. Die AwarePen Appliance kann somit für Anwendungen in der Umgebung *Informationen mit hohem Grad an Aktualität* bereitstellen. Es wird die für ubiquitäre Rechnerumgebungen wichtige Eigenschaft der zeitnahen Datenverarbeitung realisiert. Die Information über die Warteschlangenpartitionierung kann zusätzlich verwendet werden, um Prozesse im Überlastfall oder zur Energieeinsparung gar nicht erst auszuführen.

4.8 Was wurde erreicht?

Dieses Kapitel zeigte wie das Konzept von Kooperation und Kollaboration durch rückgekoppelte Regelkreise aus Kapitel 3 erfolgreich in eine Laufzeitumgebung für periodische und aperiodische Prozesse auf ubiquitären Rechnersystemen umgesetzt werden kann.

Zentral in allen Betrachtungen ist die *zeitnahe* Informationsverarbeitung *aller* Prozesse auf ubiquitären Rechnersystemen. Zur Bewertung wurde ein neues Maß, die Datenechtzeit, in Abschnitt 4.4.3 eingeführt. Die Rückkopplung dieses Maßes erlaubt es, mit einem Minimum an Informationen über das Prozesssystem vorgegebene Ausführungsziele *garantiert* zu erreichen.

Im Fair Quality Scheduling (FQS) Verfahren aus Abschnitt 4.4.10 wurde Kooperation und Kollaboration zur Regelung der zeitnahen Ablauforganisation datenechtzeitfähiger, periodischer Prozesse eingesetzt. Dabei ist es nicht notwendig Prozessaufgaben oder Prozessausführungsdauern zu kennen. Die kooperative Organisation

aperiodischer Prozesse aus Abschnitt 4.5.5 stellt sicher, dass die Ereignisverarbeitung für alle Prozesse gleichartig garantiert ist.

Beide Verfahren wurden detailliert untersucht und ermöglichen Fairness in unbekannten und nicht vorhersagbaren Einsatzumgebungen ohne administrative Einflussnahme. Sie erweisen sich als sehr robust, d.h. können Kooperations- und Kollaborationsziele über ein breites Spektrum an Veränderungen garantieren. Im schlechtesten Fall kann nach Gleichung 4.19 eine Antwortzeit des Rechnersystems für die Datenverarbeitung für jeden Prozess garantiert werden.

Die Integration in das Betriebssystems Particle OS im Abschnitt 4.6 zeigt, dass sich Kooperation und Kollaboration effizient implementieren lassen. Im Abschnitt 4.7 konnte am konkreten Anwendungsbeispiel AwarePen gezeigt werden, dass Prozesse, die für den aktuellen Kontext Informationen liefern, durch kooperative Prozessorganisation in Particle OS priorisiert werden. Kooperation durch Regelung kann in diesem Fall eine kontextsensitive Prozesseinplanung ermöglichen. Somit werden den Anwendungen Informationen mit hohem Grad an Aktualität bereitgestellt.

5. Echtzeitprozesse

Echtzeitprozesse geben Garantien über die Beendigung von Prozessabläufen an vorgegebenen Zeitschranken. Für ubiquitäre Rechnersysteme, die sensorische Informationen in unbekannten Einsatzumgebungen verarbeiten, ist es im allgemeinen schwierig Zeitschranken der Verarbeitung anzugeben. Für sehr regelmäßig ablaufende Teilaufgaben, zum Beispiel Sensordatenakquise, sind periodische Echtzeitprozesse vorteilhaft. Bei optimaler Einplanung garantieren sie die Ausführung mit einer bekannten maximalen Verzögerung. Ziel diese Kapitels ist es, die Vorteile echtzeitfähiger Prozesse für ubiquitäre Rechnersysteme nutzbar zu machen.

Zentraler Aspekt der Prozessorganisation ist das Zusammenspiel echtzeitfähiger Prozessteile der Datenakquise mit den nicht-echtzeitfähigen Teilen der Datenverarbeitung. Kollaboration, d.h. die Kopplung beider Teile, für eine erfolgreiche Datenverarbeitung ist daher Schwerpunkt dieses Kapitels. Anwendung findet dabei die Theorie der kooperativen und kollaborativen Prozessorganisation ubiquitärer Systeme aus Kapitel 3. Die Verwendung von Feedback-Information in einem den Prozessen zugrundeliegenden Regelkreis ermöglicht das Nachstellen der Prozesskopplung und beschleunigt signifikant die Geschwindigkeit der gesamten Datenverarbeitung in unbekannten Einsatzszenarien.

Zu Beginn werden wichtige wissenschaftliche Ergebnisse zur Einplanbarkeit von Echtzeitprozessen angeführt. Eine Klassifikation bestehender Ansätze zeigt Unterscheidungsmerkmale zur Prozessorganisation eingebetteter, ubiquitärer Rechnersysteme auf. Ein einheitliches Systemmodell führt die Organisation von Echtzeit- und Nicht-Echtzeitprozesse zusammen. Es wird die Verbindung zum Budget/Kosten Regelkreismodell aus Abschnitt 3.4 etabliert, das die Schnittstelle zur Regelung des Ablaufverhaltens bildet, um die Prozesskopplung herzustellen. Die praktische Umsetzung erfolgt in Form von serviceorientierten Prozessen mit den Instrumenten des Particle OS. Die Appliance Remembrance Camera für sensorgesteuerte Fotoaufnahmen wird vorgestellt. Sie dient der Bewertung der Datenverarbeitung bei kollaborativem Verhalten von Echtzeit- und Nicht-Echtzeitprozessen in einem typischen Einsatzszenario ubiquitärer Rechnersysteme.

Dieses Kapitel betrachtet periodische Echtzeitprozesse, deren Aufgabenausführung auf das lokale Rechnersystem beschränkt ist und die nicht von anderen Rechnersystemen abhängig sind.

5.1 Motivierendes Beispiel

Die Remembrance Camera (RemCam) [87][10] ist eine von einer Person getragene sensorgesteuerte Kamera. Sie zeichnet automatisch Bilder von wichtigen Ereignissen und Situationen wie Arbeitsunterbrechungen, Meetings und bestimmten Bewegungsabläufen auf. In diesem elektronischen Logbuch können RemCam-Nutzer Ereignisse schnell auffinden und visuell rekapitulieren. Eine digitale Miniaturkamera ist mit einem Mikrofon und Bewegungs-, Licht- und Temperatursensoren ausgestattet. Sensordaten werden periodisch in Echtzeit akquiriert und zur Verarbeitung bereitgestellt. Die anschließende Situationserkennung sucht Übereinstimmungen mit gespeicherten Mustern und löst gegebenenfalls die Kamera aus. Die Abbildung 5.1 illustriert das Beispiel. Während die periodischen Echtzeittasks der Akquise regelmäßig ablaufen,

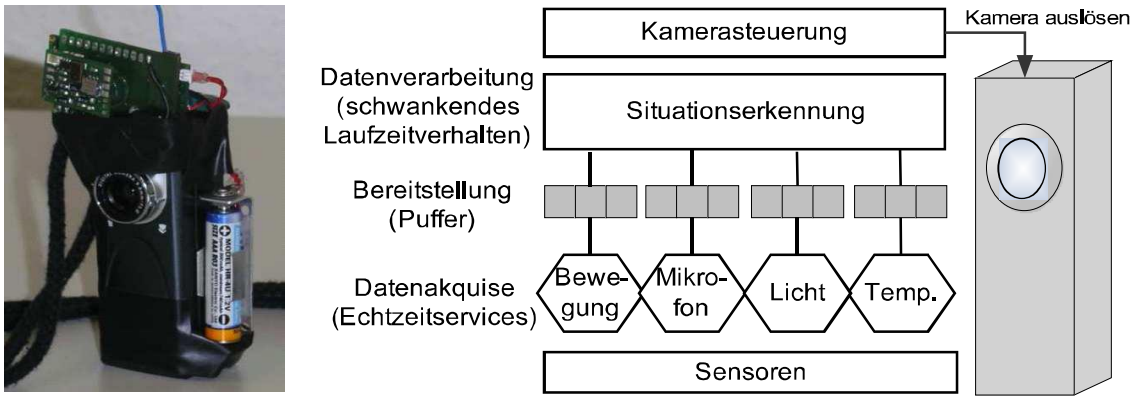


Abbildung 5.1: Links: Remembrance Camera (RemCam) bestehend aus einer digitalen Miniaturkamera und angebrachtem ubiquitären Rechner- und Sensorsystem. Rechts: Datenakquise und -verarbeitung auf der RemCam durch Echtzeit- und Nicht-Echtzeitprozesse

ist der Ablauf der Sensordatenverarbeitung schwankend und wird durch die zu erfassende Umgebungssituation bestimmt. Beide Prozessteile müssen gekoppelt werden, damit die Verarbeitung zeitnah zur Datenbereitstellung erfolgt. In unbekannten, sich schnell ändernden und mobilen Umgebungen ermöglicht es der RemCam Situationen *schneller* zu erkennen und zu reagieren.

5.2 Analyse und Einordnung ins wissenschaftliche Umfeld

Zentral für die Organisation von Echtzeitprozessen ist ein Scheduler, der die Einplanung Ausführungsreihenfolge übernimmt. Die Echtzeitbedingung fordert, dass ein Ablaufplan erstellt werden muss, bei dem keine vorgegebene Zeitschranke verletzt wird.

Durch Zeitschranken garantieren Echtzeitprozesse die Ausführung mit einer bekannten maximalen Verzögerung. In harten Echtzeitsystemen wird diese Garantie zu jedem Zeitpunkt erbracht. Schon bei Einplanung und bevor der erste Prozess startet

kann das Einhalten der Zeitschranken in solchen Systemen garantiert werden. In weichen Echtzeitsystemen ist erlaubt, die Zeitschranke bis zu einem bestimmten Grad zu überschreiten. Gegenstand des gesamten Kapitels sind harte Echtzeitsysteme.

Die Einplanung von Echtzeitprozessen wird durch eine im Scheduler implementierte Strategie beziehungsweise einen Algorithmus durchführt. Dabei ist man vor allem an *optimalen Algorithmen* interessiert. Ein Algorithmus ist optimal, wenn nur dann eine Zeitschranke verletzt wird, wenn es keinen anderen Algorithmus der selben Klasse gibt, der die Zeitschranke einhalten kann. Diese wichtige Eigenschaft besagt, dass ein optimaler Schedulingalgorithmus das beste Verfahren für eine ganze Klasse von Algorithmen ist.

Für die Beschreibung echtzeitfähiger periodischer Prozesse wird auf das eingeführte Taskmodell in Abschnitt 4.1.1 zurückgegriffen. Ein Task ist der Teil des Prozesses, der Echtzeiteigenschaften besitzt. Das ist zum Beispiel ein Service für die Sensordatenakquise, der in regelmäßigen Abständen von peripheren Sensorkomponenten Informationen anfordert und sie anderen, nicht-echtzeitfähigen, Prozessteilen zur Verfügung stellt.

5.2.1 Formalisierung und theoretische Grenzen der Einplanbarkeit

Im Taskmodell wird ein Echtzeittask wird durch das Tupel $\tau_i = \{P_i, C_i, T_i, d_i, r_i\}$ charakterisiert, wobei P_i die Funktion bezeichnet, die ausgeführt werden soll, C_i ist die Ausführungszeit des Task, T_i die Periode, d_i die Zeitschranke (engl. Deadline) und r_i bezeichnet die Ressourcen, die durch den Task belegt werden.

Zur Vereinfachung wird der Task mit der auszuführenden Funktion fest gebunden und damit P_i ausgelassen. Ressourcenkonflikte $r_i \cap r_j \neq \emptyset$ werden ausgeschlossen und die Zeitschranken mit $d_i = T_i$ festgelegt. Um Schwankungen der Ausführungsdauer zu berücksichtigen, wird die maximale Ausführungsdauer $C_i = \mathrm{WCET}_i$ angenommen. Der Mehraufwand des Schedulers für die Einplanung wird vernachlässigt. Die betrachteten ubiquitären Rechnersysteme für Echtzeittasks sind Uniprozessorsysteme.

Diese Vereinfachungen machen eine breite Untersuchung der theoretischen Grenzen der Einplanbarkeit möglich. Dem Scheduler müssen nur Ausführungsdauer C_i und Ausführungsperiode T_i bekannt sein.

Ob eine Taskmenge τ mit Echtzeittasks $\tau_i = \{C_i, d_i, T_i\}, d_i = T_i$ einplanbar ist, wird durch die erforderliche Prozessorauslastung $U = \sum_i \frac{C_i}{T_i}$ bestimmt. Das Echtzeitrechnersystem muss mindestens diese Leistung aufbringen, damit die Tasks mit den vorgegebenen Perioden ausgeführt werden können. Darüberhinaus muss das Echtzeitsystem noch Leistungsreserven für das Auftreten der maximalen Verzögerung eines Tasks aufbringen, um dessen Zeitschranke nicht zu verletzen. Es wird daher in Gleichung 5.1 eine Auslastungsbedingung für eine kleinste obere Grenze $B \leq 1$ formuliert. Die eingesetzte Schedulingstrategie bestimmt die Größe von B.

$$U = \begin{cases} \leq B, & \text{einplanbar} \\ > B, & \text{nicht einplanbar, wenn } U \leq B \text{ hinreichend } \textit{und} \text{ notwendig} \\ > 1, & \text{unmöglich für Uniprozessorsysteme} \end{cases} \tag{5.1}$$

Wenn die Bedingung $U \leq B$ nur hinreichend ist, kann die Schedulingstrategie für $U > B$ keine Aussage über die Einplanung unter Einhaltung aller Zeitschranken treffen. Das heißt aber nicht, dass in einem solchen Fall kein Ablaufplan mit Zeitschrankengarantien möglich wäre [88].

In Anlehnung an die Klassifizierung im Abschnitt 4.1.2 werden präemptive, nichtpräemptive, statische und dynamische Schedulingstrategien untersucht. Dies sind fundamentale Klassen. Jeder Scheduler eines Echtzeitsystems lässt sich in eine der Klassenkombination einordnen. In Tabelle 5.1 werden *optimale* Schedulingstrategien für jede Klassenkombination angegeben. Wenn nicht anders erwähnt, dann sind die Bedingungen für die Einplanung hinreichend *und* notwendig. Zwei Schedulingstrate-

	dynamisch	**statisch**
präemptiv	**Strategie:** EDF **Einplanbarkeit:** $U \leq 1$ [62] **Aufwand:** $O(n)$	**Strategie:** RM **Einplanbarkeit:** $U \leq n(2^{1/n} - 1)$ (nur hinreichend) [62] **Aufwand:** $O(1)$
nicht-präemptiv	**Strategie:** EDF **Einplanbarkeit:** zwei Bedingungen [72] 1. $U \leq 1$ 2. $\forall i, 1 < i \leq n; \forall L, T_1 < L < T_i : L \geq C_i + \sum_{j=1}^{i-1} \lfloor \frac{L-1}{T_j} \rfloor C_j$ **Aufwand:** pseudopolynomiell	**Strategie:** RM **Einplanbarkeit:** $U < \frac{1}{r},\ T_n = rT_1$ [89] **Aufwand:** $O(1)$

Tabelle 5.1: Optimale Schedulingstrategien und deren Auslastungsgrenzen nach Gleichung 5.1 für periodische Echtzeittasks.

gien decken alle fundamentalen Klassenkombinationen ab und sind außerdem auch optimal.

Earliest Deadline First (EDF): Es wird immer der Task bevorzugt, der die kleinste Zeitschranke aufweist. Diese geschieht zur Laufzeit - EDF ist eine dynamische Schedulingstrategie. Im präemptiven Fall wird der laufende Task unterbrochen, sobald ein anderer Task eine kleinere Zeitschranke aufweist. Im nicht-präemptiven Fall kann der Task nicht unterbrochen werden. Daher muss die maximale Verzögerung eines Task berücksichtigt werden. Letzteres ist auch der Grund für eine niedrigere Auslastungsgrenze der nicht-präemptiven EDF Strategie im Vergleich zur präemptiven.

Rate Monotonic (RM) Scheduling: Jedem Task wird aufgrund seiner Periode T respektive Rate $1/T$ eine Priorität zugeordnet. Je größer die Rate, desto höher die Taskpriorität. Hat ein Task mit höherer Priorität seine Periode erreicht, dann wird der laufende niedrigpriore Task unterbrochen. Im nicht-prämptiven Fall muss die maximale Verzögerung eines Tasks durch einen Task mit niedriger Priorität berücksichtigt werden. Die Prioritätszuordnung erfolgt zur Zeit des ersten Einbringen des Tasks ins Laufzeitsystem. Ein Task behält diese Priorität bei - RM ist eine statische Schedulingstrategie.

5.2.2 Laufzeitumgebungen mit Echtzeitfähigkeiten für ubiquitäre Rechnersysteme

Es werden bestehende Ansätze von Laufzeitumgebungen mit Echtzeitfähigkeiten für ihren Einsatz auf ubiquitären Rechnersystemen untersucht. Es wird ein Klassifikation in vier Kategorien vorgenommen.

- Prioritätsbasierte Betriebssystemkerne für eingebettete Applikationen
- Echtzeiterweiterungen von Betriebssystemen mit Zeitmultiplex
- Echtzeitbetriebssysteme aus der Forschung
- Echtzeitumgebungen für Mikrocontroller

Die ersten drei Klassen wurden für den Einsatz auf größeren einbetteten Systemen und Desktopsystemen entworfen. Sie werden nur kurz beleuchtet. Die letzte Kategorie fokussiert auf Plattformen, die auch im Bereich typischer ubiquitärer Appliances verwendet werden.

Prioritätsbasierte Betriebssystemkerne

Diese Klasse findet vor allem bei leistungsfähigen und schnell reagierenden Applikationen auf eingebetteten Systemen Anwendung. Es werden nur statische Prioritäten unterstützt. Für die Umsetzung von Echtzeitanwendungen müssen Zeitschranken zu festen Prioritäten zugeordnet werden. Übersteigt die Anzahl der Tasks die Anzahl der vom System festgelegten Prioritätsstufen, dann verkleinert sich Auslastungsgrenze B, unter der eine Einplanung von Echtzeittasks garantiert werden kann [90]. Für präemptives RM Scheduling mit theoretisch unendlich vielen Prioritätsstufen beträgt die Grenze $U \leq n(2^{1/n} - 1) \approx 0.69$. Bei 8 Prioritätsstufen reduziert sich diese Grenze auf $U \leq 0.26$. Die Folge ist eine ineffektive Nutzung von Rechenzeitressourcen. Dies ist für extrem ressourcenbeschränkte ubiquitäre Rechnersysteme ein Nachteil. Kommerzielle Betriebssysteme wie VxWorks [91] und ChrousOS [92] sind prioritätsbasierte Echzeitkerne. Typische Zielplattformen wie Intel 8088, x86, MIPS, PowerPC, SH-4, ARM, StrongARM, XScale sind leistungsfähige 16bit und 32bit Prozessoren. Sie entsprechen nicht den Plattformen für ubiquitäre Rechnersysteme wie sie in Abschnitt 2.2.3 dieser Arbeit charakterisiert wurden.

Echtzeiterweiterungen

Echtzeiterweiterungen finden sich vor allem in herkömmlichen Betriebssystemen mit Zeitmultiplexbetrieb. Beispiele sind RTLinux [93][94] und RT-MACH [95] für die Linux und MACH Betriebssysteme. Der bisherige Kernel setzt auf eine Echtzeiterweiterung auf, die garantierte Zeitschranken für Echtzeittasks geben kann, wie auch reguläre Anwendungen unterstützt. Es werden sowohl dynamische wie auch statische Echtzeitschedulingstrategien für präemptive Tasks eingesetzt. Nicht-präemptive Systemaufrufe, FIFO Warteschlangen und Synchronisierung zwischen Erweiterung und Kern reduzieren die Vorhersagbarkeit des Echtzeitverhaltens [55]. Typische Zielplattformen sind x86, PowerPC, ARM und MIPS Prozessoren. Echtzeiterweiterungen kommen im Verbund mit herkömmlichen Betriebssystem, die mehrere Megabyte Speicher benötigen. Damit sind sie für ressourcenbeschränkte ubiquitäre Rechnersystme wie sie in Abschnitt 2.2.3 dieser Arbeit charakterisiert wurden ungeeignet.

Echtzeitbetriebssysteme aus der Forschung

Die effiziente Behandlung von Echtzeittasks ist Schwerpunkt von Betriebssystemen aus der Forschung. Anwendungsdomänen sind verteilte und komplexe Regelungsaufgaben. Die Systeme erlauben vielfältige Möglichkeiten der Spezifikation von Zeitschranken und die Implementierung von zahlreichen Mechanismen, um im Voraus zu verifizieren, ob Anwendungsrestriktionen während der Laufzeit eingehalten werden können. Dynamische präemptive Schedulingstrategien erlauben eine maximale Nutzung der Ressourcen. Andere Strategien, zum Beispiel statische Strategien oder zur Behandlung von Ereignissen, können gewählt werden, und sogar im ARTS System zur Laufzeit ausgewechselt werden [96][97]. Systeme wie HARTIK [98] und Spring [99][100] führen Taskklassen für harte, weiche und nicht-echtzeitfähige Tasks ein. Dadurch werden unterschiedliche Ausführungsanforderungen getrennt und interferieren nicht. Der Spring Kernel unterstützt effizient sogar nicht-präemptive Echtzeitstrategien durch die Verwendung von Heuristiken. Spring reduziert den pseudopolynomiellen Aufwand des nicht-präemptiven EDF aus Tabelle 5.1 auf $O(n^2)$, durch backtracking sogar auf $O(kn)$ [55]. Jedoch können harte Echtzeitgarantien nur noch für eine Task-Untermenge gegeben werden. Ausführungs- und Ressourceneffizienz machen diese Gruppe für ubiquitäre Rechnersysteme interessant. Bisherige Zielplattformen wie Rechnercluster oder Einplatinenrechner auf Basis von M68k bieten für die teilweise komplexe Einplanung und Verwaltung jedoch Speicher- und Rechenressourcen auf, die für ubiquitäre Systeme nicht erfüllbar sind.

Echtzeitumgebungen für Mikrocontroller

Laufzeitumgebungen für Mikrocontroller berücksichtigen vor allem extrem beschränkte Speicher- und Rechenzeitressourcen. Dies gelingt mit statischen, auf ein Minimum reduzierten Datenstrukturen für Tasks, weitesgehenden Verzicht auf dynamische Speicherallokation, Begrenzung der einplanbaren Taskmenge, teilweiser Verzicht auf Synchronisationsmechanismen wie Semaphoren und Mutex, und Applikationsdesign mit festen, vordefinierten Schedulingpunkten. Untersucht wurde ein Auswahl von Betriebssystemen für Mikrocontroller und Sensorknotenplattformen in der vom Autor dieser Arbeit betreuten Seminararbeit von Alex Günter [101]. Sensorknoten weisen zusätzlich durch periphere Sensoren und Kommunikationsschnittstellen bereits die Komplexität ubiquitärer Rechnersysteme aus dieser Arbeit auf. Die Betrachtung der

Sensorknotenbetriebssysteme erlaubt also auch, Rückschlüsse auf die Einsatztauglichkeit von Echtzeitverfahren für ubiquitäre Rechnersysteme zu ziehen.

In Tabelle 5.2 wird eine zusammenfassende Analyse gegeben. Alle Beispielsysteme sind auf 8-bit Mikrocontrollern lauffähig. Der Speicherbedarf beträgt zwischen 360 Bytes bis 4 Kilobytes ROM für Kern und mehrere 100 Bytes RAM für die Taskinformationen und gegebenenfalls Stapelspeicher für zu sichernde Informationen bei einem Kontextwechsel. Es gelingt harte Echtzeitsysteme, sogar mit präemptiven

Name	**Schedulingstrategie**	**Eignung**
Mikrocontroller		
FreeRTOS [64], XMK [65], Pico]OS [102]	präemptives Scheduling mit festen Prioritäten, teilweise zusätzlich auch round-robin	(+) hartes Echtzeitscheduling möglich, Synchronisationsmechanismen vorhanden(−) schlechte Prozessorauslastung durch statisches Scheduling
PicOS [103]	kooperatives Multitasking mittels Co-Routinen	(+) speichereffizient durch globalen Stapelspeicher, (−) harte Echtzeit nicht garantierbar, keine Synchronisationsmechanismen
Sensorknoten		
SOS [104], Contiki [105], BTNut [106]	kooperatives Multitasking durch Co-Routinen oder ereignisbasiert, teilweise mit festen Prioritäten	(+) speichereffizient, reaktiv auf Ereignisbehandlung (−) harte Echtzeit nicht garantierbar
AmbientRT [63]	Earliest Deadline First with Deadline Inheritance (EDFI) [107]	(+) hartes Echtzeitzeitscheduling, effizient, hohe Auslastung erreichbar, EDFI löst Ressourcenkonflikte, garantiert Verklemmungsfreiheit (−) nicht verfügbar

Tabelle 5.2: Betriebssystemumgebungen für Mikrocontroller und Sensorknotenplattformen.

Tasks, auf extrem ressourcenbeschränkten Mikrocontrollern zu implementieren. Die Schedulingstrategien arbeiten mit festen Prioritäten, was zu einer Effizienzeinbuße führt. Der Einsatz von Echtzeitsystemen auf für ubiquitären Rechnersystemen geeigneten Sensorknotenplattformen ist gering. Bis auf AmbientRT bietet keines der untersuchten Systeme diese Eigenschaft an. Dem Autor sind auch darüber hinaus keine existierenden Sensorknotenplattformen bekannt, die hartes Echtzeitscheduling erlauben.

Die Gründe dafür liegen in der Anwendungsdomäne. Sensornetzwerkplattformen verarbeiten vorrangig aperiodische Ereignisse. Aufgrund der Energiespartechniken sind periodisch ablaufende Prozesse selten, extrem kurz und bedürfen keiner Echtzeitbehandlung. Applikationen auf eingebetteten ubiquitären Rechnersystemen verwenden Umgebungsinformationen, sogenannte Kontexte, als Eingabe. Änderungen der Umgebung sind nicht vorhersehbar, so dass sich keine Zeitschranken definieren lassen.

5.2.3 Echtzeitprozesse für ubiquitäre Rechnersysteme

Der Einsatz von harten Echtzeitprozessen hat für die Prozessorganisation große Vorteile. Die wichtigsten sind:

Ressourcennutzung: Echtzeitscheduler erzielen eine hocheffiziente Nutzung extrem beschränkter Rechen- und Speicherressourcen.

Determinismus: Vorhersagbarkeit des Systemverhaltens zur Laufzeit

Regelbarkeit: In einem deterministischen System können Einflussnahme und Wirkung berechnet werden.

Zeitnähe: Prozessausführung mit einer bekannten maximalen Verzögerung (Zeitschranke) erlaubt eine zeitnahe Datenverarbeitung.

Insbesondere der letzte Punkt trifft wieder die zentrale Anforderung an die Prozessorganisation eingebetteter, ubiquitärer Rechnersysteme aus Kapitel 2.

In unbekannten Einsatzszenarien hat ein Prozess p_i eines ubiquitären Rechnersystems ein unbekanntes Laufzeitverhalten und keine garantierte Zeitschranke [52].Die Vorteile von Echtzeitprozessen werden nutzbar, wenn Prozesse in zwei Teile mit verschiedenen Laufzeitverhalten zerlegt werden. Ein Echtzeitprozess p_i setzt sich aus einem nicht-präemptiven periodischen Echtzeittask τ_i^{rt} und einem nicht-echtzeitfähigen Prozessteil τ_i^{nrt} zur Datenverarbeitung zusammen. Es sei erwähnt, dass für den zusammengesetzten Prozess $p_i = \tau_i^{\mathrm{rt}} + \tau_i^{\mathrm{nrt}}$, wie anfangs erwähnt, kein Laufzeitverhalten und keine Zeitschranke garantiert werden kann.

Echtzeitprozesse können mit den Mitteln serviceorientierter Prozesse aus Abschnitt 4.3 formuliert werden. Im Folgenden wird der Zusammenhang erläutert.

- Echtzeittasks sind nicht-präemptive periodische Services, deren Ausführungsdauer durch eine vordefinierte Zeitschranke begrenzt ist. Es ist $\tau_i^{\mathrm{rt}} = s_i^{\mathrm{rt}}$ mit $\forall k: \ Pred_k(s_i^{\mathrm{rt}}) = \emptyset$.
- Nicht-echtzeitfähige Prozesse bestehen nur aus datenorientierten Services, wie sie bereits im Servicegraphen in Abschnitt 4.3.2 eingeführt wurden. Eine Ausnahme ist, dass ihre Ausführung von den Echtzeitservices unterbrochen werden kann. Es ist $\tau_i^{\mathrm{nrt}} = \left\{ s_j^{\mathrm{nrt}} | s_j \in p_i \wedge s_j \neq s_i^{\mathrm{rt}} \right\}$.
- Das Zusammenfügen beider über einen dazwischenliegenden Puffer bildet einen Servicegraphen gemäß der Definition 4.1. Prozesse p_i werden identifiziert als Pfade im Servicegraphen mit echtzeitfähigen Services s_i^{rt} als Wurzel und datenorientierten s_j^{nrt} entlang bis zum Blattknoten.

Die Abbildung 5.2 zeigt die Zerlegung. Periodische Echtzeitservices sind auf die Akquise von Sensorinformationen beschränkt. Es findet keine weitere Verarbeitung statt. Sie laufen ununterbrechbar ab. Die Ausführungsdauer kann ausreichend genau abgeschätzt werden, um Leerlaufzeiten zu minimieren. Die Echtzeitservices haben ein bekanntes Laufzeitverhalten C_i^{rt}. Über die Ausführungsperiode werden Zeitschranken mit $d_i^{\mathrm{rt}} = T_i$ definiert.

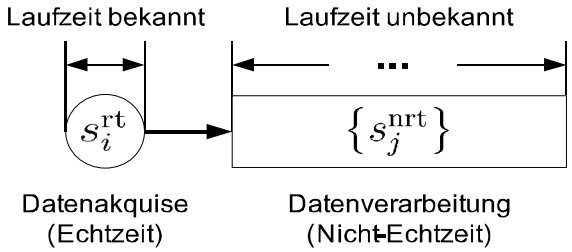


Abbildung 5.2: Zerlegung des Prozesses p_i in Echtzeitservice s_i^{rt} und nicht-echtzeitfähigem Prozessteil τ_i^{nrt}.

Nicht-echtzeitfähige Prozessteile sind datengetrieben, d.h. sie beginnen ihre Ausführung, sobald Eingangsdaten vorhanden sind. Sie zeigen ein unbekanntes, zeitveränderliches Laufzeitverhalten. Ihnen kann keine Zeitschranke zugeordnet werden. Der Ausdruck datengetrieben wird gegenüber dem Ausdruck datenorientiert bevorzugt, um die Richtung der Verbindung zwischen Echtzeitservices und den datenverarbeitenden Prozessteilen zu betonen.

Beide Prozessteile sind lose über einen Datenaustausch miteinander gekoppelt. In ubiquitären Rechnersystemen findet ein Mischbetrieb von echtzeitfähigen und nicht-echtzeitfähigen Prozessteilen statt.

5.2.4 Zusammenspiel mit nicht-echtzeitfähigen datenverarbeitenden Prozessteilen

Bisherige Systemansätze betrachten den Mischbetrieb mit nicht-echtzeitfähigen datenverarbeitenden Prozessteilen nicht. Sie werden im Hintergrund ausgeführt und beliebig von den echtzeitfähigen Prozessteilen unterbrochen. Da keine Zeitschranken für solche Teile vereinbart wurden, es wären sonst Echtzeittasks, werden nicht-echtzeitfähige Prozessteile vom Scheduler immer nur dann eingeplant, wenn keine anderen Tasks zur Ausführung bereit sind.

Dies hat Konsequenzen für die Koordination der Datenverarbeitung zwischen beiden Teilbereichen. Sollte es zu einem zu großen Datenaufkommen kommen, dann werden Daten an der Schnittstelle verworfen. Wenn dies nicht möglich ist, müssen die datenverarbeitenden Prozessteile ebenfalls Echtzeiteigenschaften aufweisen und eine Einplanung mit vergrößerter Taskmenge vorgenommen werden.

In ubiquitären Systemen sind beide Vorgehensweisen aus folgenden Gründen nicht erwünscht.

1. Das Verwerfen von Daten impliziert, dass der Aufwand der Datenakquise in Form von Rechenzeit und Energieverbrauch ungenutzt bleibt. Für ressourcenbeschränkte Geräte ist dies eine ineffektive Nutzung von Ressourcen, die vermieden werden sollte.

2. Die Ausführungsdauer datenverarbeitender Prozesse ist für ubiquitäre Systeme in unbekannten Einsatzszenarien schwierig. Eine obere Abschätzung führt zu Leerlaufzeiten im Ablaufplan. Die entstehenden Verzögerungen verhindern die zeitnahe Erfassung der Einsatzumgebung und darauffolgende Reaktion des Rechnersystems.

Im Unterschied zu bisherigen Echtzeitsystemen müssen ubiquitäre Rechnersysteme das datenorientierte Zusammenspiel von Echtzeit- und Nicht-Echtzeitprozessen berücksichtigen. Die Regelbarkeit erlaubt es, Einfluss auf die Zeitschranken der Echtzeittasks zu nehmen. Es kann sich zur Laufzeit zielgerichtet an das unbekannte Laufzeitverhalten von nicht-echtzeitfähigen Prozessteilen angepasst werden.

5.2.5 Fazit

Zusammenfassend ist festzuhalten:

- Echtzeitscheduling findet wenig Anwendung auf ubiquitären Rechnersystemen.
- Echtzeitprozesse sind aus periodischen Echtzeitservices und datengetriebenen Services zusammengesetzt.
- Periodische Echtzeitservices ermöglichen eine effizientere Nutzung von Rechenzeitressourcen.
- Für eine zeitnahe Datenverarbeitung müssen ubiquitäre Rechnersysteme das Zusammenspiel zwischen Echtzeitservices und datenorientierten Services bewerten und regeln.

Ziel dieses Kapitel ist es, die in Abschnitt 5.2.3 analysierten Vorteile echtzeitfähiger Prozesse für ubiquitäre Rechnersysteme nutzbar zu machen. Dazu wird letzteres Problem der obigen Liste aufgegriffen und ein kollaboratives Organisationsschema vorgeschlagen. Durch Rückkopplung und Regelung kann unter unbekannten Laufzeitverhalten der nicht-echtzeitfähigen Prozessteile eine zeitnahe Datenverarbeitung erreicht werden. Applikationen mit Echtzeitprozessen erreichen *höhere Reaktionsgeschwindigkeiten* und eine bestmögliche Ressourcennutzung.

5.3 Problembeschreibung

Ubiquitäre Rechnersysteme sollen eine zeitnahe Datenverarbeitung unter Verwendung von Echtzeitservices ermöglichen. Der folgende Abschnitt präzisiert die Problemstellung der Kopplung zwischen echtzeitfähigen und nicht-echtzeitfähigen Prozessteilen.

Ein Echtzeitprozess p_i aus dem Motivationbeispiel in Abschnitt 5.1 wird wie in Abschnitt 5.2.3 in zwei Teile zerlegt: einem nicht-präemptiven periodischen Echtzeitservice s_i^{rt} und einem datengetriebenen, nicht-echtzeitfähigen Prozessteil, der in dieser Betrachtung von einem datengetriebenen Service s_j^{nrt} gebildet wird. Die Tabelle 5.3 listet die Parameter beider Teile auf. Obwohl die datengetriebenen Prozesse keine Periode haben, kann man eine *scheinbare Periode* T^{nrt} definieren. Sie beschreibt die kleinste Zeitspanne bis alle datengetriebenen Nicht-Echtzeitprozesse mindestens einmal ausgeführt wurden.

Beide Prozessteile sind über einen gemeinsam genutzten Puffer *lose gekoppelt*. Der Echtzeitservice beschreibt den Puffer und der nicht-echtzeitfähige liest ihn aus. Bei diesem Zusammenspiel können zwei Problemfälle auftreten — Datenverlust und Leerlauf.

	Echtzeitservice s_i^{rt}		Datengetriebener Service s_j^{nrt}	
Ausführungsdauer	C_i^{rt}	fest und bekannt	C_j^{nrt}	unbekannt, zeitlich veränderlich
Periode	T_i^{rt}	Zeitschranke, bekannt	$T^{\mathrm{nrt}} = \sum_j C_j^{\mathrm{nrt}}$	*scheinbar*, unbekannt, zeitlich veränderlich

Tabelle 5.3: Parameter der Prozessteile eines Echtzeitprozesses auf ubiquitären Rechnersystemen

Datenverlust. Dieser Problemfall tritt bei Überlast auf, wenn die Ausführungsdauer C_j^{nrt} des datengetriebenen Service größer wird. Dadurch verzögert sich das Auslesen der Daten aus dem Puffer und die nächste Ausführung des Echtzeitservice überschreibt den Pufferinhalt bevor die Daten verarbeitet werden konnten. Die Datenverarbeitung ist *nicht zeitnah*. In Abbildung 5.3 wird ein langlaufender datengetriebener Service vom Echtzeitservice unterbrochen und verpasst schließlich die Datenverarbeitung. Der Problemfall wird vermieden, wenn die Bedingung

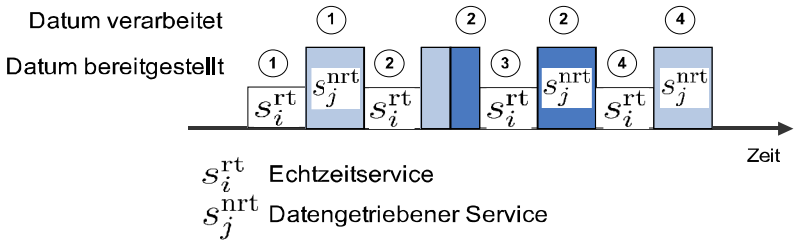


Abbildung 5.3: Datenverlust: Datum 3 wird nicht verarbeitet wegen der zu großen Ausführungsdauer C_j^{nrt}

$\forall i, j : \sum_j C_j^{\mathrm{nrt}} < T^{\mathrm{nrt}} \leq T_i^{\mathrm{rt}}$ erfüllt wird. Der nicht-echtzeitfähige Prozessteil muss die Datenverarbeitung beendet haben bevor der Echtzeitservice erneut ausgeführt wird.

Leerlauf. Wenn sich die Ausführungsdauer C_j^{nrt} des datengetriebenen Service verkleinert, dann wird auf den Puffer mehrmals zugegriffen bevor neue Daten durch den Echtzeitservice bereitgestellt werden. Die Datenverarbeitung ist *nicht effektiv*. Abbildung 5.4 illustriert die Situation.

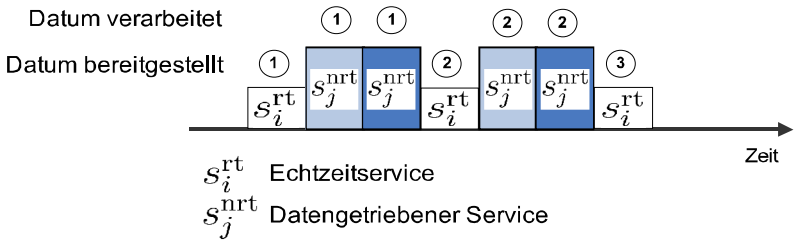


Abbildung 5.4: Leerlauf: Nach Datum 1 und 2 läuft der datengetriebene Service jeweils leer bis ein neues Datum vom Echtzeitservice bereitgestellt wird.

Der Problemfall wird vermieden, wenn die Bedingung $\forall i, j : T^{\text{nrt}} \geq T_i^{\text{rt}} > \sum_j C_j^{\text{nrt}}$ erfüllt ist. Der Echtzeittask muss die Daten bereitgestellt haben, bevor der datengetriebene Prozessteil sie verarbeitet.

Es leitet sich die zentrale Problemformulierung der Organisation von Echtzeitprozessen in ubiquitären Rechnersystemen ab.

Problem 5.1 (Echtzeitfähige Prozessorganisation)**.** *In unbekannten Einsatzszenarien mit unbekannten und zeitlich veränderlichen Ausführungsdauern C_j^{nrt} von nichtechtzeitfähigen Prozessteilen, bestimme die Prozessausführung so, dass die folgenden zwei Bedingungen erfüllt sind.*

1. **Vermeiden von Verlust und Leerlauf:** $\forall i, j : T_i^{rt} = T^{nrt} > \sum_j C_j^{nrt}$
2. **Erhaltung harter Echtzeit:** $\forall i$: s_i^{rt} *beendet seine Ausführung spätestens an der Zeitschranke* $d_i = T_i^{nrt}$

Je nach vorliegender Situation der datenverarbeitenden Prozessteile, muss die Kopplung zwischen Echtzeit- und Nicht-Echtzeitprozessen nachgestellt werden, um zeitnahe Datenverarbeitung ohne Verlust zu ermöglichen und Rechenzeitressourcen effektiv zu nutzen. In diesem Szenario der Datenverarbeitung sind beide Prozessteile voneinander abhängig. Sie müssen kollaborieren.

5.4 Serviceorientierter Entwurf

Der Entwurf beschreibt die funktionale Aufteilung, engl. *separation of concerns*, der Prozesse im Servicegraphen in zwei Schichten. Abbildung 5.5 illustriert den Servicegraphen bestehend aus Echtzeitprozessen. Zwischen den Services an den Schicht-

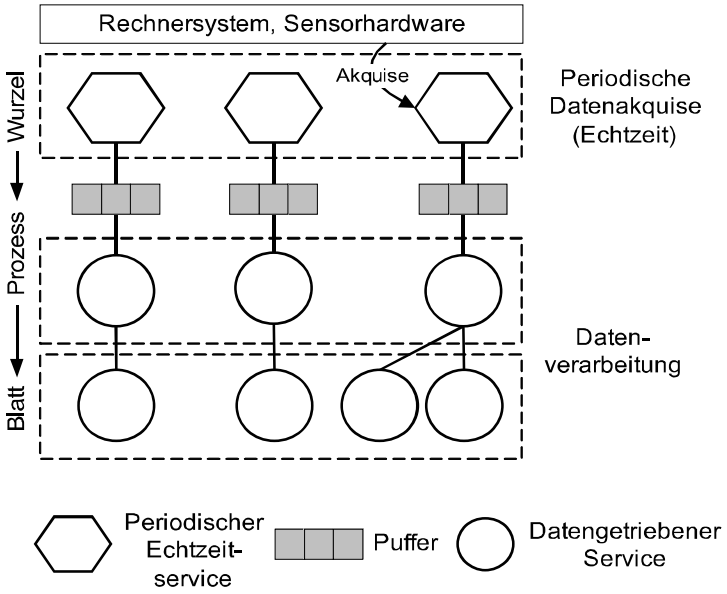


Abbildung 5.5: Echtzeitprozesse im Servicegraphen aufgeteilt in zwei Schichten. Die Wurzel ist ein nicht-präemptiver periodischer Echtzeitservice, zum Beispiel für die Akquise von Sensordaten. Die Datenverarbeitung erfolgt datengetrieben durch datenorientierte Nicht-Echzeitservices. Beide zusammen bilden einen Echtzeitprozess.

grenzen besteht eine über den Puffer realisierte mittelbare 1 : 1 Beziehung. Die Relation definiert sich wie folgt:

Definition 5.1 (Grenzpufferrelation). *Es ist* $\mathcal{I} \subseteq S \times B \times E$ *die Inzidenzrelation aus Definition 4.1. Es besteht eine mittelbare* 1 : 1 *Beziehung zwischen* s_i^{rt} *und* s_j^{nrt}*, wenn und nur wenn* b_i *Grenzpuffer ist. Es ist mit Teilbeziehung* $h \in \mathcal{H}$ *und* $\mathcal{I} = \mathcal{H}\mathcal{H}^T$

- $h_{ik} = 1$ *für die Inzidenzbeziehung* $s_i^{rt} \rightarrow b_k$
- $h_{kj} = -1$ *für Inzidenzbeziehung* $b_k \rightarrow s_j^{nrt}$ *mit* $\forall j_1, j_2 : b_k \rightarrow s_{j_1}^{nrt}$ *und* $b_k \rightarrow s_{j_2}^{nrt}$ *ist* $j_1 = j_2$*. D.h. eine Mehrfachsenke an* b_k *wird ausgeschlossen.*

5.4.1 Zeitnahe Datenverarbeitung mit serviceorientierten Echtzeitprozessen

Die Problemstellung für Echtzeitprozesse aus Abschnitt 5.3 wird im Systementwurf berücksichtigt. Zeitnahe Datenverarbeitung erfordert das Vermeiden von permanentem Datenverlust und Leerlauf. Diese Größen werden an den Puffern zwischen den Systemschichten gemessen und an das Laufzeitsystem zurückgeführt.

Datenverlust und Leerlauf werden durch einen 1-bit Zustandsautomaten an den Puffern gemessen. Das Bit wird in seinen Komplementärzustand versetzt, wenn der Echtzeitservice und der datengetriebene Service wechselseitig auf den Puffer zugreifen. Das Bit bleibt in seinem Zustand, wenn Echtzeitservice und datengetriebener Service wiederholt in einer nicht wechselseitigen Folge auf den Puffer zugreifen. In diesen Fällen wird jeweils ein Verlust- respektive Leerlaufzähler hochgezählt. Diese Zähler werden in die Ablaufplanung des Echtzeitschedulers im Laufzeitsystem zurückgeführt. Ein Regler im Laufzeitsystem passt die Echtzeitprozesse an, um permanenten Datenverlust und Leerlauf zu vermeiden. Abbildung 5.6 zeigt die Datenverarbeitung im Überblick.

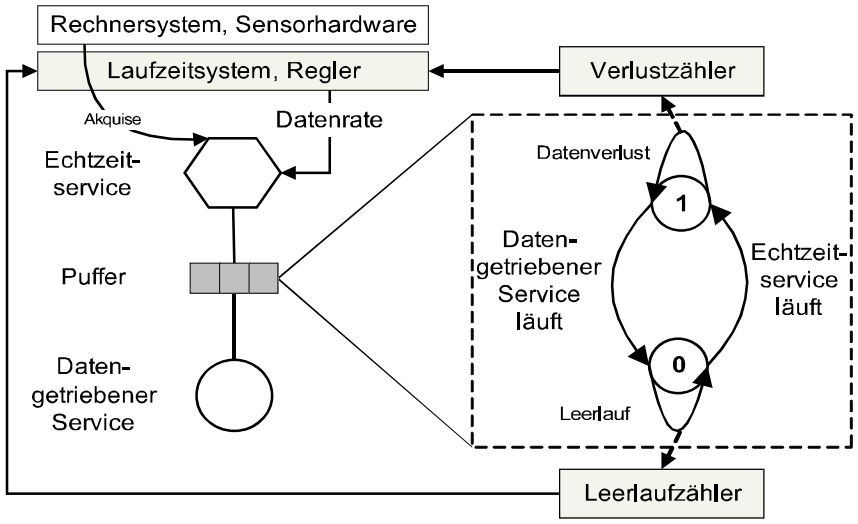


Abbildung 5.6: Datenverarbeitung am Puffer zwischen Echtzeit- und datengetriebenen Services. Die Datenverarbeitungsleistung wird mit Hilfe eines 1-bit Automaten gemessen. Leerlauf- und Verlustzähler werden ins Laufzeitsystem zurückgekoppelt und die Datenrate der Echtzeitprozesse angepasst.

Verlust- und Leerlaufzähler quantifizieren die Datenverarbeitungsleistung der Echtzeitprozesse, d.h. wieviel Sensorinformationen nicht verarbeitet werden oder noch zusätzlich verarbeitet werden können. Sie sind ein Maß für die Stärke der Kopplung der Systemschichten. Gemäß der Zähler wird durch einen Regler die Datenrate der Echtzeitservices für die Bereitstellung der Sensorinformationen so angepasst, dass Verlust und Leerlauf vermieden wird. Es ist dann die zeitnahe Datenverarbeitung erreicht und die Kopplung zwischen beiden Systemschichten ist maximal.

Die rückgekoppelte Anpassung der Datenrate von Echtzeitservices zum Erreichen einer vorgegebenen Datenverarbeitungsleistung wird als *Buffer Feedback Scheduling (BFS)* bezeichnet. Ziel von BFS ist die zeitnahe Datenverarbeitung von Echtzeitprozessen ohne Datenverlust und Leerlauf.

5.4.2 Implikationen für den Entwurf von Applikationen

Es werden wichtige Eigenschaften für Applikationen auf ubiquitären Rechnersysteme erreicht. Die Aufteilung in zwei Schichten trennt strikt zwischen verschiedenen Laufzeitverhalten. Nicht-präemptive periodische Echtzeitservices garantieren eine fortlaufende Bereitstellung von aktuellen Sensorinformationen. Das serviceorientierte Prozessmodell stellt eine datenbasierte Bindung zu den Echtzeitservices her, die es erlaubt, jegliche datenverarbeitenden Prozesse ohne Berücksichtigung des darunterliegenden Laufzeitverhaltens zu realisieren. Die Kopplung der Schichten durch das Laufzeitsystem gewährleistet eine effektive Datenverarbeitung in unbekannten sich ändernden Einsatzszenarien.

Diese gekoppelte Abstraktion entwickelt bisherige Entwurfskonzepte weiter, die entweder kein definiertes Zeitverhalten berücksichtigen können [13][104][68] oder das Zusammenspiel nicht diskutieren [65][103][102].

Der Systementwurf ist für Anwendungen auf ubiquitären Rechnersystemen geeignet, die folgende Eigenschaften aufweisen.

- Die gesamte Applikation läuft auf dem eingebetteten, ubiquitären Rechnersystem.
- Alle Eingabedaten, zum Beispiel Sensorinformationen, werden von ein oder mehreren, periodischen, nicht-unterbrechbaren Services zur Verfügung gestellt.
- Datengetriebene Prozesse implementieren alle weiteren Funktionen zur Ereignis- und Situationserkennung.

5.5 Systemmodell echtzeitfähiger Prozesse

Buffer Feedback Scheduling (BFS) - der Entwurf der zeitnahen Datenverarbeitung aus dem vorangegangenen Abschnitt 5.4.1 - wird in einen Regelkreis nach dem Budget/Kosten Modell für die Organisation von Echtzeitprozessen umgesetzt. Im unmittelbar ersten Schritt wird die Methode zur Übertragung des Budget/Kosten Modells auf spezifische Prozessklassen aus Abschnitt 3.4.4 bemüht. Es sind Budget, Kosten, Ziel für Echtzeitprozesse zu identifizieren, die in Tabelle 5.4 zusammengefasst sind. Vorteil der Echtzeitprozesse ist nach Abschnitt 5.2.3 Regelbarkeit. Sie erlaubt es, Ausführungsperioden der Echtzeitservices nachzustellen, um die Kopplung der Systemschichten zielgerichtet zu beeinflussen. Es wird die Regelungsaufgabe für BFS formuliert.

Budget	Datenverarbeitungsleistung
Kosten	Datenrate der Echtzeitservices
Ziel	Hohe Datenverarbeitungsleistung aller Echtzeitprozesse durch Kopplung zwischen Echtzeitservices und datengetriebenen Prozessteilen

Tabelle 5.4: Entsprechungen des Budget/Kosten Modells für Echtzeitprozesse

Regelungsziel: Alle Prozesse arbeiten ohne Datenverlust und Leerlauf.

Stellgröße: Ausführungsperioden der Echtzeitservices

Regelstrecke: Datenverarbeitungsleistung am Puffer zwischen den Services. Datenverlust/Leerlauf ist Rückkopplungsgröße.

Störgröße: Schwankungen der datenabhängigen Ausführungsdauer der datengetriebenen Prozessteile

Die Abbildung 5.7 illustriert die Einbettung der Echtzeitprozesse in den Regelkreis. Echtzeitservices stellen bei Ausführung Daten in den Puffern für die datengetriebe-

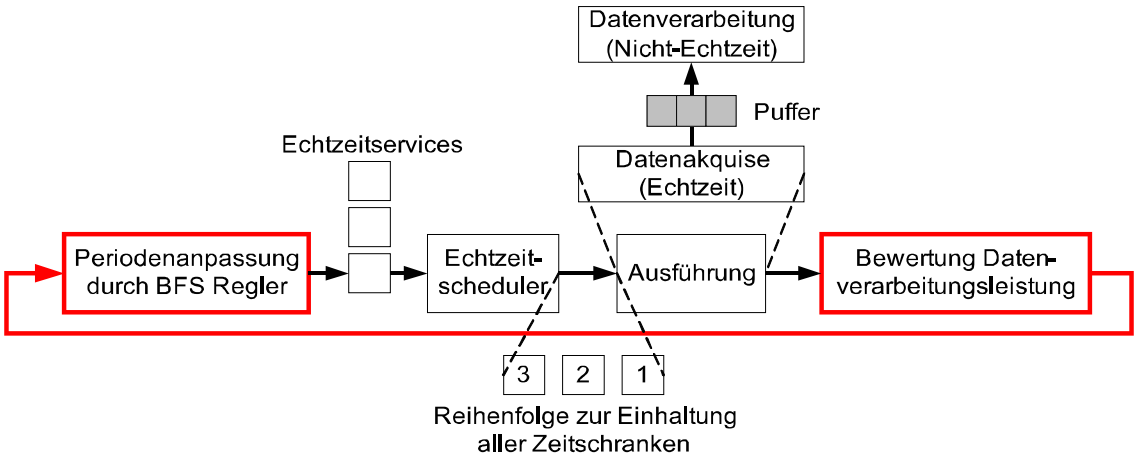


Abbildung 5.7: Einbettung der Echtzeitprozesse in einen Regelkreis

nen Services zur Verfügung. Die Datenverarbeitung erfolgt sofort, wenn keine Echtzeitservices ausgeführt werden. Durch Verlust- und Leerlaufzähler wird die Datenverarbeitungsleistung bewertet und mit dem Regelungsziel verglichen. Der BFS Regler stellt die Ausführungsperioden der Echtzeitservices nach, und der Echtzeitscheduler plant die Services in dieser Konfiguration erneut ein.

Es sind zwei Aufgaben zu erfüllen: (1) Modellierung der Ausführung und Regelung, (2) Beweis der Echtzeitfähigkeit bei geregelter Ausführung der Echtzeitservices.

5.5.1 Modellierung der Ausführung

Je nachdem wie stark die periodische Datenbereitstellung und die Verarbeitungsdauer der Services voneinander abweichen, sind ganzzahlige Datenverluste oder Leerläufe erst nach mehreren Ausführungsperioden zu beobachten. Nicht-ganzzahlige Verluste respektive Leerläufe sind in jeder Periode vorhanden und summieren sich auf. Die Ausführung lässt sich als dynamisches System modellieren.

Die Ausführung der Echtzeitprozesse im Zeitintervall w wird durch die Datenverarbeitungsleistung $P(w) = s(w) - j(w)$ bewertet. Es werden Echtzeitservices und datengetriebene Services in zwei Gruppen S^{rt} und S^{nrt} zusammengefasst. $s(w)$ ist die Anzahl der Ausführungen der Ablaufgruppe S^{rt} und $j(w)$ die analoge Entsprechung für S^{nrt} jeweils im Zeitintervall w. Im Fall von $P(w) = 0$ gibt es zu jeder Echtzeitserviceausführung eine entsprechende Ausführung der datengetriebenen Verarbeitung. $P(w) > 0$ bezeichnet einen Datenverlust, während $P(w) < 0$ Leerlauf angibt. Die Datenverarbeitungsleistung $P(w)$ spiegelt also sowohl den Verlustzähler wie auch den Leerlaufzähler des Entwurfs der Datenverarbeitung aus Abschnitt 5.4.1 wieder.

Mit Herleitung aus Anhang H.1 wird folgendes Systemmodell der Ausführung in Z Transformation etabliert.

$$G(z) = \frac{P(z)}{\Delta T(z)} = -K_f \frac{z}{z-1} \tag{5.2}$$

$G(z)$ - die Transferfunktion der Ausführung von Echtzeitprozessen - entspricht der Transferfunktion des Budgets im Budget/Kosten Modell. K_f symbolisiert die Systemfrequenz im Idealfall, d.h. dass die bereitgestellten Daten der Echtzeitservices von allen datengetrieben Services verarbeitet werden bevor sich die Echtzeitservices wiederholen. Der Wert ist $K_f = \frac{s(w)}{C^{\text{nrt}}(w)} = \frac{1}{C^{\text{nrt}}}$.

5.5.2 BFS Regelung

Die Regelung initialisiert die Ausführungsperioden der Echtzeitservices auf eine gemeinsame Periode $\forall i : T_i = \max_i T_i$. Diese Maßnahme ist notwendig, um alle Prozesse auf den gemeinsamen BFS Regler zu synchronisieren. Eine detaillierte Begründung ist in Anhang H.2 gegeben. In der anschließenden Arbeitsphase wird die gemeinsame Periode vom BFS Regler *gedehnt* oder *gestaucht*, um die Datenverarbeitungsleistung der Ausführung auf den gewünschten Wert einzustellen und Störungen durch das unbekannte Laufzeitverhalten der datengetriebenen Services zu kompensieren.

Der BFS Regler implementiert ein P Regelungsgesetz. Die Fehlerdifferenz zum Regelungsziel wird entsprechend verstärkt und als Periodenänderung an die Ausführung in Systemgleichung 5.2 weitergegeben. Die Transferfunktion des BFS Regler in Z Transformation ist

$$R(z) = \frac{\Delta T(z)}{E(z)} = K_m \tag{5.3}$$

Sie entspricht der Kostenmodellierung im Budget/Kosten Modell.

5.5.3 Transferfunktion des BFS Regelkreises

Der BFS Regelkreis hat eine weitere Besonderheit. Die im Systemmodell der Ausführung eingeführte Systemfrequenz K_f ist in der transienten Arbeitsphase der Regelung *nicht* konstant.

Es ist $K_f(w) = \frac{s(w)}{C^{\text{nrt}}(w)}$. Bei Vergrößerung von C^{nrt} unterbrechen die Echtzeitservices häufiger die datengetriebenen. Es kommt zu einem Anstieg von $s(w)$. Dadurch wird die scheinbare Periode T^{nrt} übermäßig gedehnt und die Datenverarbeitung verzögert. Dieses Verhalten wird durch Zerteilung von $K_f(w)$ modelliert. Es ist

$$K_f(w) = \gamma(w) K_f, \text{ mit } K_f \text{ ideale Systemfreq.} \tag{5.4}$$

$\gamma(w)$ ist ein *zeitveränderlicher* Verstärker im Regelkreis und bestimmt zur Laufzeit den Wert von $s(w)$ - die Anzahl der Ausführungen der Ablaufgruppe der Echtzeitservices bis die scheinbare Periode erreicht ist. T^{nrt} ist auch unbekannt und ergibt sich während dieser Messung. Die initiale Messverzögerung für $K_f(w)$ ist die längste. Es wird ein Verzögerungsglied $z^{-\eta}$ vor den BFS Regler eingeführt, mit dem diese Verzögerung geeignet berücksichtigt wird. Damit ergibt sich der BFS Regelkreis wie in Abbildung 5.8 dargestellt. Die Transferfunktion des geschlossenen BFS Regelkreises

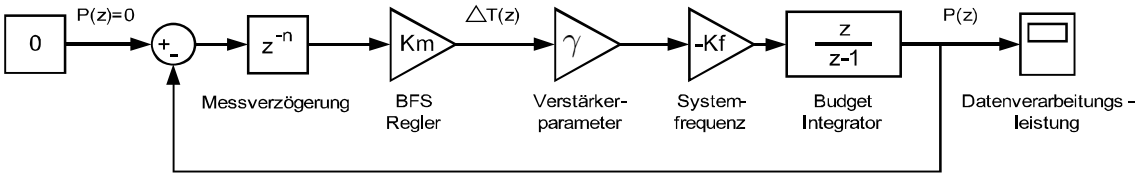


Abbildung 5.8: BFS Regelkreis als Simulink Modell

in Z Transformation ist

$$F(z) = \frac{-K_m K_f \gamma}{(z-1)z^{\eta-1} - K_m K_f \gamma} \tag{5.5}$$

Es ist zu bemerken, dass γ zeitveränderlich ist. Die Analyse des Regelkreises muss diesen Umstand berücksichtigen. Nach der geschlossenen Modellierung muss der zweite Teil, der Nachweis des Echtzeitverhaltens, erfolgen.

5.5.4 Schedulinganalyse

Echtzeitservices $s_i^{\text{rt}} = (C_i^{\text{rt}}, T_i^{\text{rt}})$ sind periodisch und nicht-unterbrechbar. Die Ausführung von s_i^{rt} muss vor der Zeitschranke $d_i = T_i^{\text{rt}}$ beendet sein. Die Regelung der Serviceperioden und die Reglerinitialisierung müssen diese Eigenschaften erhalten.

Die Änderung der Ausführungsperioden durch den Regler erlaubt nicht die Anwendung von statischen Schedulingstrategien. Echtzeitservices werden daher mit der nicht-präemptiven, dynamischen EDF Strategie eingeplant. Gemäß der Klassifizierung der Schedulingstratgien in Tabelle 5.1 liefert das Kriterium von Jeffay et al. [72] die notwendige und hinreichende Bedingung für die Einhaltung aller Zeitschranken. Die Betrachtungen in Abschnitt 5.4 liefern die Begründung, daß dieses Kriterium auch für Echtzeitservices angewendet werden kann. Alle Resultate werden daher mit dem Begriff Service anstelle von Task formuliert. Wenn es eindeutig aus dem Text hervorgeht, wird auf die Kennzeichnung von Echtzeitservices zur Unterscheidung von datengetriebenen Services verzichtet.

Theorem 5.1 (Kriterium von Jeffay et al.)**.** *Es sei* $S = \{(C_1, T_1), \ldots, (C_n, T_n)\}$ *eine Menge von periodischen, nicht-unterbrechbaren Echtzeitservices in Reihenfolge nicht-abnehmender Perioden, d.h.* $T_i \leq T_j$ *für* $i < j$*. Die Menge ist einplanbar unter Einhaltung aller Zeitschranken* $d_i = T_i$*, gdw. folgende zwei Bedingungen erfüllt sind:*

(1) $\sum_i \frac{C_i}{T_i} \leq 1$

(2) $\forall i, 1 < i \leq n; \forall L, T_1 < L < T_i : L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_j} \right\rfloor C_j$

Die Bedingungen sind notwendig und hinreichend.

Die Bedingung (1) fordert, dass der Prozessor nicht überlastet wird. Die Ungleichung in Bedingung (2) bildet die kleinste untere Schranke der von S erzeugten Prozessorauslastung, die eine Einplanung im Intervall L ermöglicht.

Die Änderung der Ausführungsperioden durch den BFS-Regler muss die Echtzeiteigenschaften erhalten. Dies kommt im folgenden Korollar zum Ausdruck.

Korollar 5.1. *Ist die Menge der Echtzeitservices $S = \{(C_i, T_i)\}$ einplanbar gemäß Theorem 5.1 ist, dann ist die neue Menge $S^* = \{(C_i, T_i + k)\}$, in der jede Periode um die Konstante $k > 0$ vergrößert wurde, ebenfalls unter Einhaltung aller Zeitschranken einplanbar.*

Es ist sofort ersichtlich und bedarf keines weiteren Beweises, dass Korollar 5.1 auch für $k < 0$ erfüllt ist, so lange $T_i = T_i^{\min}$ die kleinste Periode, für die im Anwendungsfalls die Gültigkeit des Kriteriums aus Theorem 5.1 gezeigt wurde, nicht unterschritten wird. Der BFS-Regler ist bei $T_i^{\min}$ gesättigt. Das Ermitteln von $T_i^{\min}$ ist mindestens pseudopolynomiell [72]. Eine praktische Alternative ist es, eine gemeinsame Ausführungsperiode der Echtzeitservices zu definieren. Es ist $\forall i : T_i^{\min} = T^{\min} = \sum_i C_i^{\mathrm{rt}} + \min_j \sum_j C_j^{\mathrm{nrt}}$, die kleinste gemeinsame Periode, in der Echtzeitservices und datengetriebene Services ausgeführt werden können.

Die Regelung des Systems initialisiert die Ausführungsperioden der Echtzeitservices auf eine gemeinsame Periode $\forall i : T_i = \max_i T_i$. Das Korollar 5.2 bringt die Erhaltung der Echtzeiteinplanung für diesen Fall zum Ausdruck.

Korollar 5.2. *Ist die Menge der Echtzeitservices $S = \{(C_i, T_i)\}$ einplanbar gemäß Theorem 5.1 ist, dann ist die neue Menge $S^{*,max} = \{(C_i, T^{max})\}$ mit $T^{max} = \max_i T_i$ ebenfalls unter Einhaltung aller Zeitschranken einplanbar.*

Die Beweise der Korollare sind im Anhang I enthalten. Die Anpassung der Ausführungsperioden der Echtzeitservices durch die Regelung erhält die Echtzeiteigenschaften. Die zweite Bedingung der Problemstellung aus Abschnitt 5.3 ist erfüllt.

5.6 Prozesskollaboration

Echtzeitservices und datengetriebene Prozessteile sind gekoppelt und somit voneinander abhängig. Ziel ist, permanenten Datenverlust und Leerlauf zu vermeiden. Nach Definition 3.3 entspricht dieses zielgerichtete Zusammenwirken einer Prozesskollaboration. Der BFS Regler bestimmt die Stärke der Kollaboration unter Berücksichtigung des rückgeführten Ausführungsverhalten.

Dieser Abschnitt prüft, ob das Kollaborationsziel erreicht wird. Dazu wird der Regelkreis aus Abbildung 5.8 und unter Verwendung der Transferfunktion aus Gleichung 5.5 analysiert. Für die Transferfunktion $F(z)$ aus Gleichung 5.5 werden wichtige Eigenschaften wie Stabilität, Schnelligkeit und Genauigkeit bestimmt.

5.6.1 Stabilität

Stabilität bestimmt, ob die Systemausgabe, hier die Datenverarbeitungsleistung, einen konstanten Wert erreichen kann. Das Kriterium ist, dass alle Polstellen der Transferfunktion im Einheitskreis der komplexen Zahlenebene liegen müssen. Im

ersten Schritt wird der Einfluss des zeitveränderlichen Verstärkerparameters γ der Systemfrequenz $K_f(w)$ des Regelkreises untersucht.

Bei Änderung von C^{nrt} erfolgt die Messung von $K_f(w)$. In der ersten Messphase wird der BFS Regler verzögert, d.h. es erfolgt noch keine Reaktion, bis die Phase abgeschlossen ist. In dieser Zeit werden zwei Beobachtungen gemacht.

1. γ verändert sich langsamer als der Systemausgang.
2. Es gibt ein $\gamma_{\max}$, welches zu Beginn einer Systemanregung maximal ist.

Es werden weiterhin feste maximale und minimale Grenzen für C^{nrt} vereinbart, d.h. $C^{\text{nrt}}_{\min} \leq C^{\text{nrt}} \leq C^{\text{nrt}}_{\max}$. Es sei $K_f = \frac{1}{C^{\text{nrt}}_{\min}}$ die feste Bezugssystemfrequenz. γ verstärkt K_f in den Grenzen

$$\frac{1}{C^{\text{nrt}}_{\max}} \overbrace{<}^{\gamma<1} K_f = \frac{1}{C^{\text{nrt}}_{\min}} \overbrace{<}^{\gamma>1} \frac{\max s(w)}{C^{\text{nrt}}_{\max}}$$

mit $\max s(w) = \left\lceil \frac{C^{\text{nrt}}_{\max}}{C^{\text{nrt}}_{\min}} \right\rceil$ als die größte Anzahl an Ausführungen der Ablaufgruppe der Echtzeitservices, die bei größter Änderung der Ausführungsdauer der datengetriebenen Services auftritt. γ ist auf den Bereich

$$\frac{C^{\text{nrt}}_{\min}}{C^{\text{nrt}}_{\max}} \leq \gamma \leq \max s(w) \frac{C^{\text{nrt}}_{\min}}{C^{\text{nrt}}_{\max}}$$

festgelegt. γ ist endlich und verändert sich langsamer als das System. Es ändert die Stabilität des Systems nicht. Für die Stabilitätsanalyse wird das größte $\gamma = \gamma_{\max}$ angenommen. Es ändert die Reglerausgabe am stärksten. Trotz der Annahme konkreter Werte für $C^{\text{nrt}}_{\min}$ und $C^{\text{nrt}}_{\max}$ ist die Kenntnis der aktuellen Ausführungsdauer C^{nrt} der datengetriebenen Services für die Stabilitätsanalyse nicht notwendig.

Das Messverzögerungsglied $z^{-\eta}$ des Regelkreises aus Abbildung 5.8 führt zu einem System höherer Ordnung. Es treten komplexe Polstellen auf. Der Systemausgang kann Überschwingverhalten zeigen. Die Datenverarbeitungsleistung schwingt um den Zielwert. Die Analyse der Polstellen geschieht mit dem numerischen Wurzelortskurvenverfahren (engl. root locus) [50]. Das Verfahren durchläuft eine Serie von Verstärkungen K_m und bestimmt die zugehörigen Polstellen. Es bilden sich Trajektorien der Polstellen aus, die auf Einheitskreislage getestet werden können. Abbildung 5.9 zeigt die Polstellen für zwei verschiedene Verzögerungen η. Für $\eta > 0$ verlaufen die Trajektorien nur abschnittsweise innerhalb des Einheitskreises. Abhängig von den konkreten Werten von K_m und $\gamma_{\max}$ ist das System $F(z)$ bis zu bestimmten Verzögerungen η stabil. Abbildung 5.10 illustriert den Zusammenhang. Bei der Reglersynthese liest man ein geeignetes K_m aus Abbildung 5.10 ab, damit $F(z)$ unter angenommenen η stabil ist. Je größer die Beträge $|K_m|$ der Reglerverstärkung des BFS Reglers, desto kleiner ist der Bereich für γ, damit $F(z)$ auch bei hohen Messverzögerungen noch stabil ist. Der Wert von η für ein konkretes System hängt direkt von den Ausführungsdauern der Services ab. Für praktische Anwendungen sollte mindestens $\eta = N^{\text{rt}} \max s(w)$ (N^{rt} ist Anzahl der Echtzeitservices) gewählt werden, um die initiale Messverzögerung abzubilden. $\gamma_{\max}$ leitet sich direkt aus den Annahmen der Grenzen der Ausführungsdauer her. Pessimistische Grenzen sind zu bevorzugen.

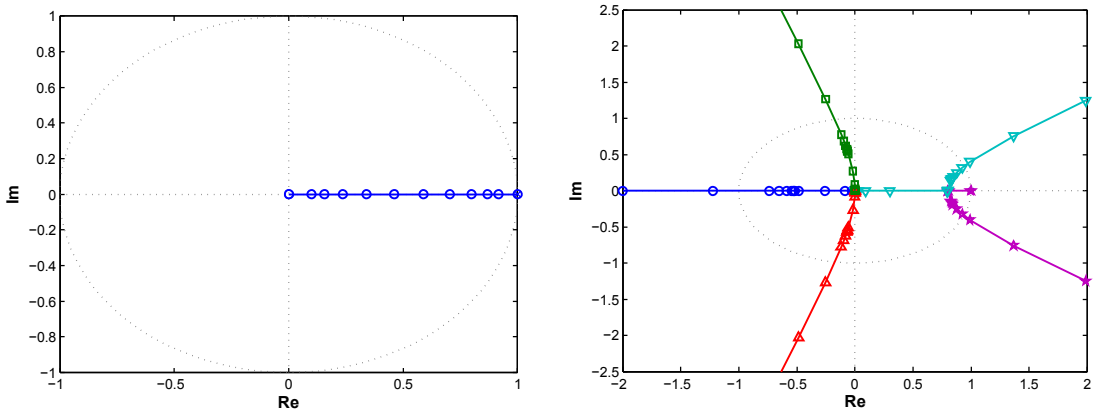


Abbildung 5.9: Wurzelortskurve für BFS Regelkreis $F(z)$. Links: Messverzögerung $\eta = 0$. Alle Polstellen sind im Einheitskreis. $F(z)$ ist stabil. Rechts: Messverzögerung $\eta = 5$. Abschnitte von Trajektorien mit komplexen Polstellen befinden sich außerhalb des Einheitskreises. Nur für bestimmte K_m sind alle Pole im Einheitskreis.

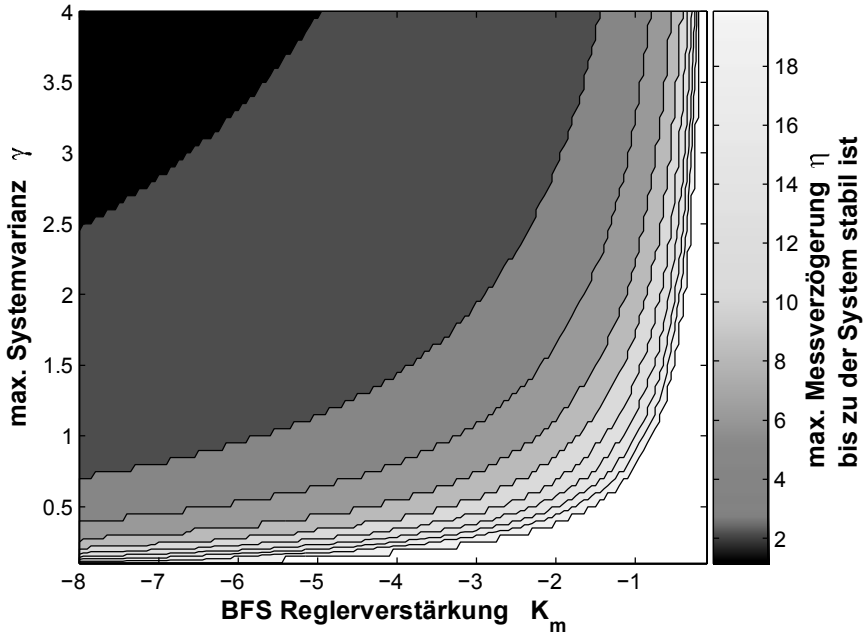


Abbildung 5.10: Stabilitätsanalyse des BFS Regelkreises in Abhängigkeit von K_m und $\gamma_{\max}$. Die Grauschattierung der Flächen und die Skala rechts zeigen die maximale Verzögerung bis $\eta = 20$ an, unter der $F(z)$ stabil ist.

Die Abbildung 5.11 zeigt den Verlauf der Serviceperioden und des Systemausgangs bei Anstieg der C^{nrt} (=Störung) im BFS Regelkreis aus Abbildung 5.8. Die Perioden werden gedehnt, so dass das Kollaborationsziel $P(w) = 0$ erreicht wird. Für stabiles Regelungsverhalten wurden K_m, γ und $\eta > 0$ entsprechend gewählt. Deutlich zu

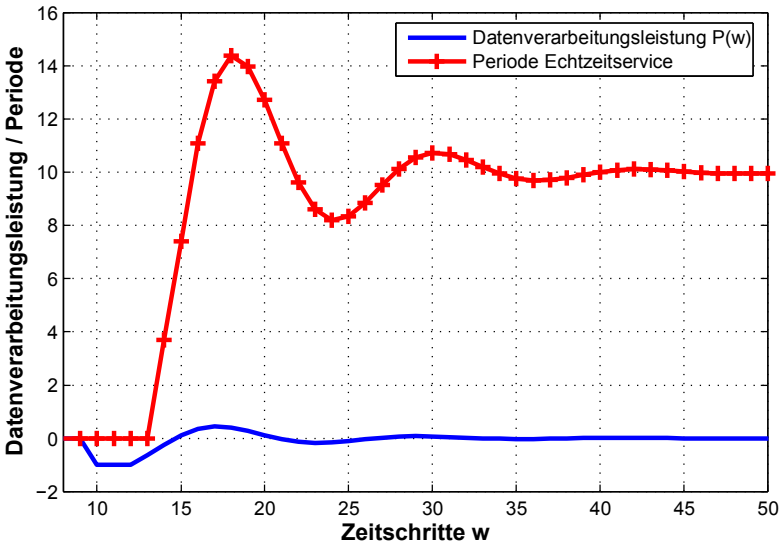


Abbildung 5.11: Verlauf der Serviceperioden bei Störung durch veränderte C^{nrt}. Die Störung wird durch eine vergrößerte Periode kompensiert. $P(w) = 0$ wird nach $w = 47$ Zeitschritten erreicht, d.h. Kopplung der Datenverarbeitung durch Kollaboration.

erkennen ist das Überschwingverhalten der Perioden und $P(w)$ in den ersten Schritten. Die Oszillation verursacht abwechselnd Verlust und Leerlauf. Nach Zeitschritt $w = 47$ ist die neue Periode stabilisiert und Datenverlust und Leerlauf treten praktisch nicht mehr auf. Echtzeitservices und datengetriebene Services sind dann über die Datenverarbeitung aneinander gekoppelt. Die Systemmodellierung setzt $U = 1$, die maximale Prozessorauslastung, voraus. Ist $F(z)$ stabil, dann wird die gekoppelte Datenverarbeitung und $U = 1$ garantiert erreicht. Die Ressourcennutzung ist maximal effektiv.

5.6.2 Schnelligkeit

Große Beträge $|K_m|$ können das System derart beschleunigen, dass recht früh der Zielwert erreicht wird. Auftretende komplexe Polstellen verursachen allerdings ein Überschwingverhalten, das in Abhängigkeit von K_m und η unterschiedlich stark ausgeprägt ist. Abbildung 5.12 (links) illustriert den Zusammenhang.

Überschwingverhalten. Kleine Beträge $|K_m|$ erlauben größeren Spielraum in den Parametern γ und η, insbesondere, wenn die Abschätzung sehr pessimistisch ausfällt. Für $|K_m| < 1$ besteht selbst für große η praktisch kein Überschwingverhalten. Große $|K_m| >> 1$ Beträge verursachen große Überschwinger mit kurzfristig extremen Datenverlusten *und* Leerlaufzeiten. Parameterunsicherheiten in γ und η müssen gegen das Überschwingverhalten aus Abbildung 5.12 (links) abgewogen werden.

Schnelligkeit. Für Beträge $|K_m| < 1$ nähert sich $P(w)$ sehr langsam dem Zielwert und es treten über längere Phasen *kontinuierlich* Verluste *oder* Leerläufe auf.

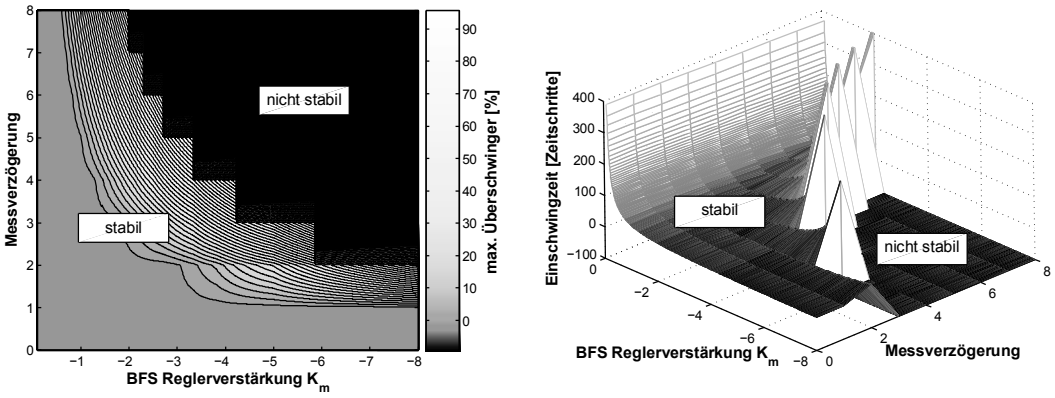


Abbildung 5.12: Links: maximales Überschwingverhaltens von $F(z)$ für steigende $|K_m|$. Rechts: Einschwingzeit für stabile $F(z)$ bis zum Erreichen des Zielwertes (2% Umgebung). Kürzere Zeiten für $|K_m| > 1$, aber mit extremen Überschwingern. Regionen, mit nicht stabilem Systemverhalten sind gekennzeichnet.

Allerdings wird mehrmals und *früher* der Zielwert $P(w) = 0$ erreicht. Die Abbildung 5.12 (rechts) zeigt die Einschwingzeiten zusätzlich in Abhängigkeit der Messverzögerung η. Kurzfristige extreme Abweichungen vom Zielwert müssen gegenüber kürzeren Einschwingzeiten abgewogen werden.

Für große $|K_m|$ besteht erhöhte Gefahr der Instabilität. Beide Graphen in der Abbildung 5.12 zeigen ein schmales und schmaler werdendes Band zwischen der K_m Achse und der Instabilitätsregion.

5.6.3 Genauigkeit

Die Untersuchung der Genauigkeit beschränkt sich auf $F(z)$ mit den Parametern $\{(K_m, \gamma)\}$ für die $F(z)$ nach vorangegangener Analyse stabil ist. Es ist $F(1) = \frac{-K_m K_f \gamma}{0 - K_m K_f \gamma} = 1$, d.h. es wird ein vorgegebener Zielwert ohne stationären Regelungsfehler erreicht.

5.6.4 Ergebnisse

Die folgende Tabelle 5.5 fasst die Ergebnisse der vorausgegangenen Abschnitte zusammen. Als zentrales Ergebnis bleibt festzuhalten, dass das Kollaborationsziel $P(w) = 0$ bei unbekanntem Laufzeitverhalten der datengetriebenen Prozessteile erreicht wird. Da der BFS Regler die Services synchronisiert ist $T_i^{\mathrm{rt}} = T^{\mathrm{nrt}}$. BFS ist eine Lösung des Problems der Prozessorganisation von Echtzeitprozessen aus Abschnitt 5.3. BFS ist ein kollaboratives Verfahren der Prozessorganisation.

5.7 Simulationsstudien

Dieser Abschnitt vergleicht das theoretisch vorhergesagte Verhalten der BFS geregelten kollaborativen Prozessorganisation mit den Ergebnissen einer Systemsimulationstudie [87]. Letztere wurden mit dem Ptolemy II Rahmenwerk [108] erzielt. Außerdem wird kollaboratives Prozessverhalten mit dem ungeregelten Prozessverhalten verglichen. Dazu wird ein Lastsprung - eine plötzliche Änderung der Ausführungsdauer C_j^{nrt} der datengetriebenen Services - simuliert. Untersucht wird die Systemantwort des BFS geregelten wie auch des ungeregelten Systems für eine Vergrößerung

Eigenschaft	Systemverhalten	Interpretation
Stabilität	stabil für K_m, die numerisch aus Wurzelortskurve gewonnen wurden. Messverzögerung η muss bekannt sein.	Kollaboration erreicht feste Datenverarbeitungsleistung der Prozesse. Ressourcennutzung ist maximal effektiv ($U = 1$).
Schnelligkeit	kleine Einschwingzeiten für $\|K_m\| > 1$. Gefahr der Instabilität.	kürzere Einschwingzeiten müssen gegen extremes Überschwingverhalten und die Gefahr der Instabilität abgewogen werden.
Genauigkeit	wenn System stabil, wird $P(w) = 0$ ohne stationären Regelungsfehler erreicht.	permanenter Datenverlust und Leerlauf werden vermieden.
Überschwingen	extreme Überschwinger für $\|K_m\| >> 1$. Gefahr der Instabilität.	Parameterunsicherheiten in γ und η müssen gegen kurzfristig extreme Abweichungen und Gefahr der Instabilität abgewogen werden.

Tabelle 5.5: Eigenschaften der kollaborativen Prozessorganisation von Echtzeitprozessen.

und Verkleinerung der Ausführungsdauer im Sprung. Die Ergebnisse werden mit Hilfe von zwei Metriken verglichen. Es sind $N^{\text{verlust}}(t)$, $N^{\text{leerlauf}}(t)$ die Verlust- und Leerlaufzähler im Zeitpunkt t wie in Abschnitt 5.4.1 eingeführt. $n^{\text{serv.}\cdot}(t)$ ist die akkumulierte Anzahl der Ausführungen der Echtzeitservices. Die Metriken sind dann die folgenden akkumulierten Raten

$$r^{\text{verlust}}(t) = \frac{\sum_t N^{\text{verlust}}(t)}{\sum_t n^{\text{serv.}\cdot}(t)} \quad \text{(akkumulierte Verlustrate)}$$

$$r^{\text{leerlauf}}(t) = \frac{\sum_t N^{\text{leerlauf}}(t) \cdot C_j^{\text{nrt}}(t)}{t} \quad \text{(akkumulierte Leerlaufrate)}$$

Zu Beginn der Untersuchungen werden Perioden der Echtzeitservices auf $T = 0.1$ gesetzt. In einer zuvor ausgeführten Schedulinganalyse wurde festgestellt, dass sie alle unter ihren Zeitschranken einplanbar sind. In beiden Abbildungen 5.13 und 5.14 tritt der Lastsprung im Zeitpunkt 2 auf. Die Ausführungsdauer der datengetriebenen Services C_j^{nrt} wird von 0.05 auf 0.3 Zeiteinheiten erhöht. Nach dem Lastsprung erhöht sich die Verlustrate. Der BFS Regler verändert die Perioden der Echtzeitservices, um die Verlustrate zu kompensieren. Ein detaillierter Vergleich mit dem ungeregelten Verhalten ist in Tabelle 5.6 angegeben. Die Abbildungen 5.15 und 5.16 zeigen das Verhalten bei einem umgekehrten Lastsprung. Es wird die Ausführungsdauer der datengetriebenen Services C_j^{nrt} wird von 0.3 auf 0.05 Zeiteinheiten verkleinert. Wiederum tritt der Lastsprung im Zeitpunkt 2 auf. Die beim Lastsprung frei werdenden Rechenzeitressourcen verursachen einen Leerlauf der datengetriebenen Services. Der BFS Regler stellt die Perioden entsprechend nach, um den Leerlauf zu kompensieren. Tabelle 5.7 vergleicht im Detail ungeregeltes und BFS geregeltes Prozessverhalten hinsichtlich der Leerlaufrate.

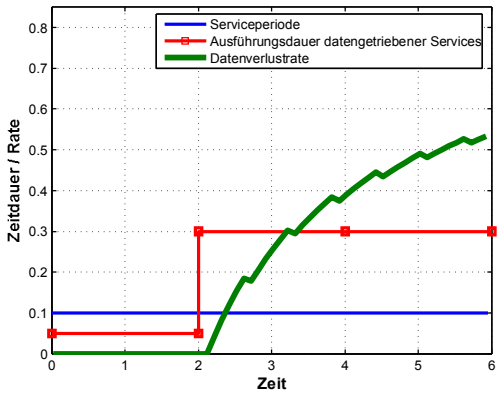


Abbildung 5.13: Ungeregeltes Lastsprungverhalten für einen Anstieg von C_j^{nrt}. Ausführungsperioden der Echtzeitservices bleiben konstant und die Verlustrate steigt.

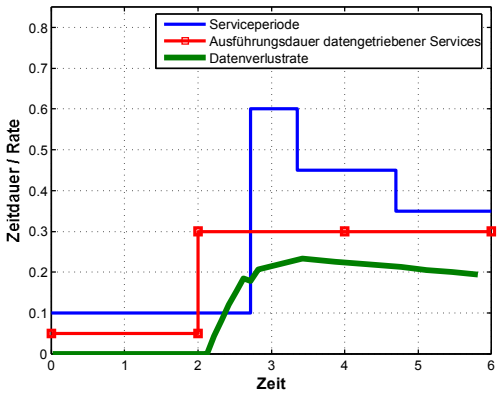


Abbildung 5.14: BFS geregeltes Lastsprungverhalten für einen Anstieg von C_j^{nrt}. Ausführungsperioden der Echtzeitservices werden gedehnt und die Verlustrate fällt.

	ungeregelt	**BFS geregelt**	**Erläuterung**
Perioden der Echtzeitservices [Zeiteinheiten]			
$t = 0$	0.1	0.1	Ausgangssituation
$t = 2$	0.1	0.1	Lastsprung setzt ein, Perioden noch konstant
$t = 2.8$	0.1	0.6	verzögerte Periodendehnung, max. Überschwingverhalten der BFS Regelung
$t = 4.7$	0.1	0.35	Periodendehnung abgeschlossen
Verlustrate $r^{\text{verlust}}(t)$			
$t = 0$	0	0	Ausgangssituation
$t = 2$	> 0	> 0	Lastsprung setzt ein, Verlustrate steigt
$2 < t < 3.4$	$\Delta r^{\text{verlust}}(t) > \Delta r^{\text{verlust, BFS}}(t)$		BFS beginnt verzögert zu regeln und bremst Verlustrate
$t = 3.4$	0.31	0.24	BFS erreicht max. Verlustrate, ungeregelter Fall steigt weiter
$t > 3.4$	0.83	$\to 0$	BFS regelt Verlustrate gegen 0, ungeregelter Fall wird sättigt

Tabelle 5.6: Vergleich des ungeregelten mit dem BFS geregelten Systemverhaltens bei einem ansteigenden Lastsprung

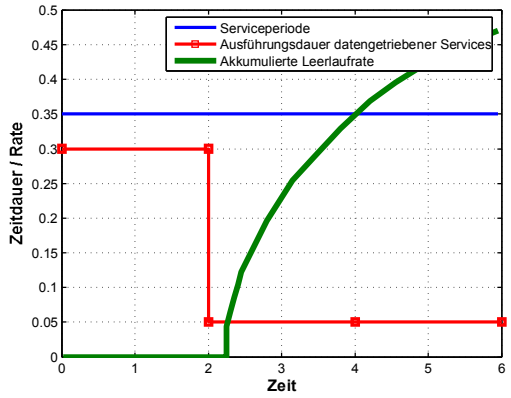


Abbildung 5.15: Ungeregeltes Lastsprungverhalten für eine Verkleinerung von C_j^{nrt}. Ausführungsperioden der Echtzeitservices bleiben konstant und die Leerlaufrate steigt.

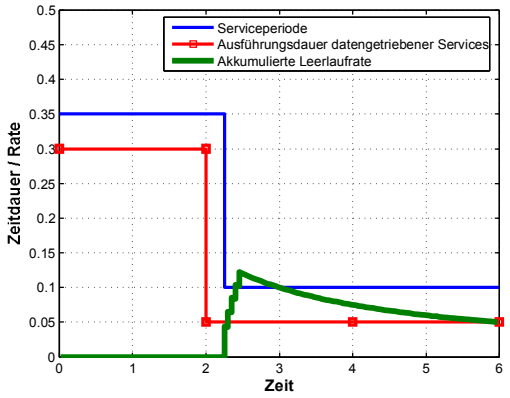


Abbildung 5.16: BFS geregeltes Lastsprungverhalten für eine Verkleinerung von C_j^{nrt}. Ausführungsperioden der Echtzeitservices werden gestaucht und die Leerlaufrate fällt.

	ungeregelt	**BFS geregelt**	**Erläuterung**
Perioden der Echtzeitservices [Zeiteinheiten]			
$t = 0$	0.35	0.35	Ausgangssituation
$t = 2$	0.35	0.35	Lastsprung setzt ein, Perioden noch konst.
$t = 2.25$	0.35	0.1	verzögerte Periodenstauchung, Periode nach unten begrenzt $T^{\min} = 0.1$
$t = 2.45$	0.35	0.1	Periodenstauchung abgeschlossen
Leerlaufrate $r^{\text{leerlauf}}(t)$			
$t = 0$	0	0	Ausgangssituation
$t = 2$	> 0	> 0	Lastsprung setzt ein, Leerlaufrate steigt
$2 < t < 2.45$	> 0	> 0	BFS beginnt verzögert zu regeln ($t = 2.25$), Effekt durch Akkumulation nicht sichtbar
$t = 2.45$	0.12	0.12	BFS erreicht max. Leerlaufrate, ungeregelter Fall steigt weiter
$t > 2.45$	0.71	$\rightarrow 0$	Regeleffekt wird durch Akkumulation erst jetzt sichtbar, BFS regelt Leerlaufrate gegen 0, ungeregelter Fall wird gesättigt

Tabelle 5.7: Vergleich des ungeregelten mit dem BFS geregelten Systemverhaltens bei einem fallenden Lastsprung

Die Ergebnisse der Simulationsstudien bestätigen die theoretischen Vorhersagen der kollaborativen Prozessorganisation aus Abschnitt 5.6. Das Schwingverhalten ist jedoch weniger stark ausgeprägt. Die Perioden in der Simulation wurden durch Messung erfasst. Die auftretende Messverzögerung verursacht ein gröberes Bild des Periodenverlaufs, des Schwingverhaltens und der Raten. Alle erzielten Ergebnisse belegen, dass die kollaborative Prozessorganisation Leerlauf und Datenverlust kompensiert. Die kollaborativ organisierte Datenverarbeitung übertrifft damit in allen Fällen die Leistung der ungeregelten Prozessorganisation. Es wird bestätigt, dass Kollaboration das Problem der Organisation von Echtzeitprozessen aus Abschnitt 5.3 löst.

5.8 Implementierung

Die Implementierung erfolgte mittels serviceorientierten Prozessen im Laufzeitsystem des Particle OS aus Abschnitt 4.6. Die Implementierungsplattform war die Particle Computer Plattform [19]. Die modulare Struktur der Betriebssystemumgebung erlaubt es als Scheduler einen dynamischen, nicht-präemptiven EDF Scheduler einzusetzen. Es ist ein optimaler Schedulingalgorithmus (siehe Tabelle 5.1), der eine höhere Systemauslastung als statische Verfahren erreichen kann. Außerdem sind Ressourcenkonflikte zwischen den Echtzeitservices ausgeschlossen und eine effektive Speichernutzung möglich, da das Speichern und Wiederherstellen des Prozesskontextes entfällt.

EDF plant die Echtzeitservices ein. Der Dispatcher bringt sie zur Ausführung. Ist die Warteschlange des Schedulers leer, dann bringt der Dispatcher die datengetriebenen Services zur Ausführung. Es gibt *keinen* speziellen Hintergrundprozess, der vom Scheduler eingeplant wird.

Ein Puffer an der Systemgrenzschicht dient für den Datenaustausch zwischen Echtzeit- und datengetriebenen Services. Direkt am Puffer werden die 1-bit Automaten für die Erfassung von Verlusten und Leerlaufzeiten verwaltet. Ein Mutexmechanismus für die Grenzpuffer zum Ausschluss des gleichzeitigen Schreib-/Lesezugriffs wurde in [109] vorgestellt.

Der BFS Regler misst nachdem alle Services mindestens einmal ausgeführt wurden die Systemfrequenz $K_f(w)$ und bestimmt das aktuelle $\gamma(w)$. Verlust- und Leerlaufzähler werden zur Datenverarbeitungsleistung $P(w)$ kombiniert. Der Vergleich mit dem Kollaborationsziel $P(w) = 0$ liefert die Fehlerdifferenz, die durch BFS Reglerverstärkung und $\gamma(w)$ die Periodenänderung der Echtzeitservices bestimmt. Im nächsten Durchgang werden die Echtzeitservices mit dieser Periode ausgeführt.

Die Tabellen 5.8 und 5.9 geben den Speicher- und Rechenaufwand der Laufzeitkomponenten des BFS Kollaborationsmechanismus wieder. Der Dispatcher ist unverändert zum Particle OS aus Abschnitt 4.6. Der BFS Regler implementiert auch das Messsystem zur Ermittlung von $K_f(w)$. Er verwendet das Zeitformat des Dispatchers. Dies vergrößert sowohl den Speicheraufwand wie auch den Rechenaufwand für Dehnung und Stauchung. Spezialisierte Datenformate für den Regler könnten den Aufwand signifikant senken. Stauchung der Ausführungsperioden ist etwas aufwendiger, da der Regler nach unten gesättigt ist. Es muss zusätzlich gegen die Sättigungsgrenze $T^{\min}$ des BFS Reglers verglichen werden.

Komponente	Speicheraufwand ROM [kB]	RAM [kB]
EDF Scheduler	5.556 (4.2%)	0.161 (4,0%)
BFS Regler	1.870 (1.4%)	0.021 (0.5%)
Summe	7.426 (5.6%)	0.182 (4.5%)

Tabelle 5.8: Speicheraufwand der BFS Laufzeitkomponenten

Funktion	Zyklen
Scheduler	
Einplanung, 4 Serv.	1903
Zeitschrankenvgl.	421
BFS Regler	
Stauchung, 5 Serv.	7371
Dehnung, 5 Serv.	6481

Tabelle 5.9: Rechenaufwand ausgewählter Funktionen von BFS auf der Particle Computer Plattform

5.9 Entwicklungsunterstützung

Der Systementwuf aus Abschnitt 5.4 ist in zwei Schichten unterteilt. Die untere Schicht besteht aus den Echtzeitservices, die als Datenlieferanten für die datengetriebenen Verarbeitungsprozesse der höheren Schicht arbeiten. Die Entwicklung eines schichtenüberspannenden Prozesses aus Echtzeit- und Nicht-Echtzeitservices erfolgt mit den entsprechenden Werkzeugen des Particle OS aus Abschnitt 4.6.3. Ergänzt wird die Werkzeugkette um die Schedulinganalyse und Reglersynthese. Die Schritte sind in Abbildung 5.17 dargestellt. Die Einplanung der Echtzeitservices erfordert die

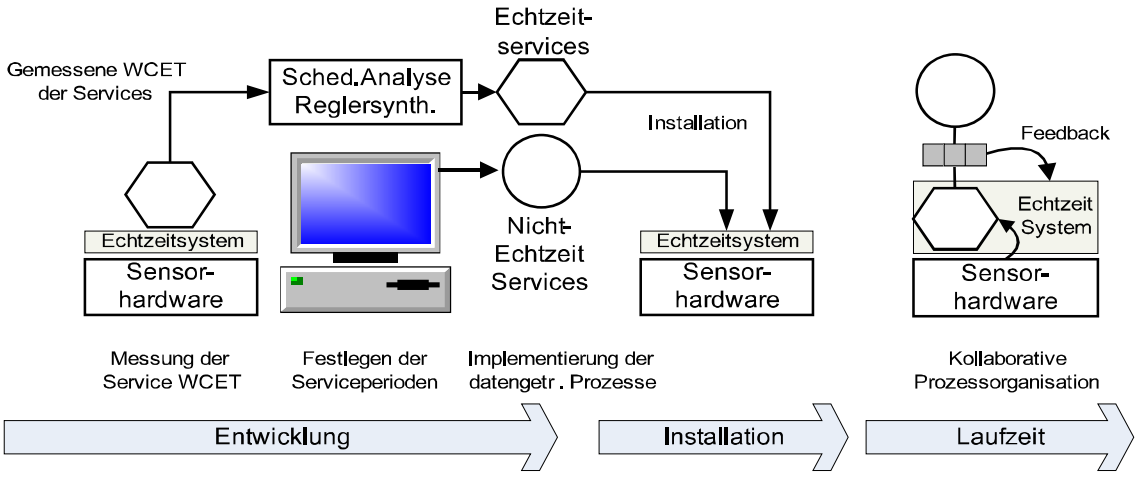


Abbildung 5.17: Vorgehen bei der Entwicklung von Echtzeitprozessen für ubiquitären Rechnersysteme.

Kenntnis der WCET. Die Echtzeitservices sind auf eine Funktionalität beschränkt, zum Beispiel Sensordatenakquise. Es findet keine weitere Datenverarbeitung statt. Sie haben daher eine konstante Ausführungsdauer, die der WCET entspricht. Zur Gewährleistung dieser Annahmen sind Echtzeitservices native Services im Sinne des Particle OS. Datengetriebene Services können wahlweise in Form von Java-Services realisiert werden. Die Schritte der Abbildung 5.17 im Einzelnen:

1. Die WCET wird durch Messungen der Ausführungsdauer auf der tatsächlichen Hardwareplattform des ubiquitären Rechnersystems durchgeführt. Die

Systembibliotheken des Particle OS enthalten eine Stoppuhr-Funktion, in die die Ausführung eingekapselt wird. In dieser Phase sollten keine anderen Prozesse ausgeführt werden, um akkurate Zeitmessungen zu erhalten. Echtzeitservices werden wiederverwendet. Der Messvorgang muss daher nur einmal pro Plattform erfolgen. Die ermittelte WCET wird am Service gespeichert, um in zukünftigen Applikationen wiederverwendet zu werden.

2. Der Entwickler legt die Ausführungsperioden und damit die Zeitschranken der Echtzeitservices fest. Mit dem Kriterium von Jeffay aus Abschnitt 5.1 wird die Schedulinganalyse durchgeführt. Die Analyse stellt fest, ob das *ungeregelte* System die Echtzeitservices in den gegebenen Zeitschranken auszuführen vermag. Mit dem Festlegen der Parameter für die Echtzeitservices wird über das Wurzelortskurvenverfahren der BFS Regler synthetisiert.

3. Die datengetriebenen Services werden implementiert. Es ist auch die Verwendung von in Java geschrieben Services möglich. Dabei sind keine Zeitschranken oder andere Implementierungseinschränkungen zu beachten. Bis auf die Funktionen, die die Echtzeitservices benutzen, kann jede Funktionalität des Particle OS verwendet werden. Der Entwickler muss den datengetriebenen Service am Austauschpuffer registrieren. Beide Services werden in einem Prozessgraphen zusammengefügt.

4. Echtzeitprozesse werden auf dem Rechnersystem installiert. Die Java Services können auch über die Luftschnittstelle übertragen und installiert werden. Das macht einen Austausch der Datenverarbeitung zur Laufzeit möglich. Eine erneute Schedulinganalyse muss nicht durchgeführt werden. Das Echtzeit-Laufzeitsystem wird die Kollaboration zwischen den Prozessteilen zur Laufzeit einstellen.

5.10 Anwendung Remembrance Camera

Anwendung findet die kollaborative Prozessorganisation von Echtzeitprozessen in der Remembrance Camera (RemCam) aus dem Motivationsbeispiel in Abschnitt 5.1. Die RemCam [87][10] ist eine von einer Person getragene sensorgesteuerte Kamera. Sie zeichnet automatisch Bilder von wichtigen Ereignissen und Situationen wie Arbeitsunterbrechungen, Meetings und bestimmten Bewegungsabläufen auf. In diesem elektronischen Logbuch können RemCam-Nutzer Ereignisse schnell auffinden und visuell rekapitulieren. Es wird keine kontinuierliche Aufnahme durchgeführt, bei der im Anschluss der Benutzer riesiges Datenmaterial sichtet, sondern nur Ereignisse von besonderer Bedeutung oder Veränderung zum bisherigen Umgebungskontext. Die RemCam ist eine *mobile Appliance*, die zur autonomen Situationserkennung und Aufzeichnung *in unbekannten Umgebungen* eingesetzt wird. Ein Desktop PC wird benötigt, um die Bilder zur Betrachtung zu übertragen.

Die Kamera besteht aus zwei Komponenten: Ein eingebettetes ubiquitäres Rechner- und Sensorsystem zur Erfassung der Umgebung und Verarbeitung der Sensorinformationen und eine Miniaturkamera, die vom Sensorgerät gesteuert wird. Die Kamera ist eine Apitek PenCam, 4.5x4x15 cm groß und wird um den Hals oder am Gürtel getragen. Die Abbildung 5.18 (links) zeigt den Prototypen. In Abbildung 5.18 (rechts) werden die Schritte der Datenverarbeitung gezeigt. Sensordaten

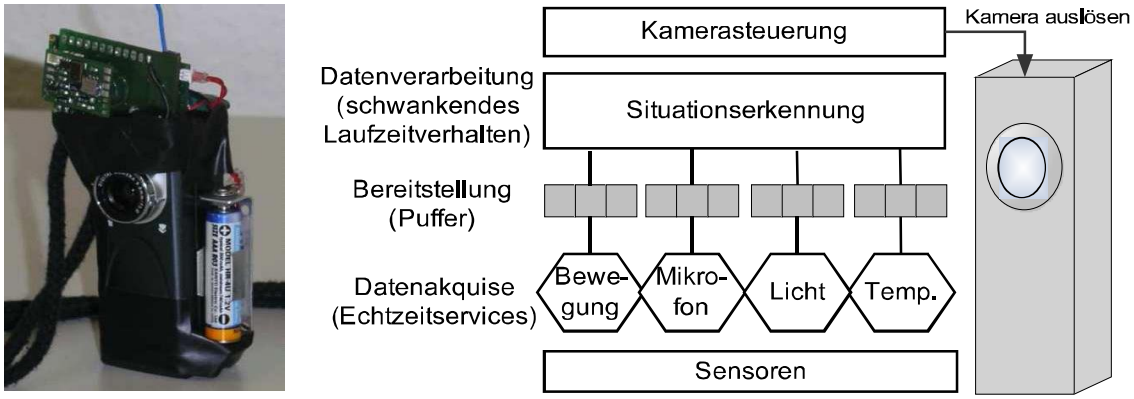


Abbildung 5.18: Links: Remembrance Camera (RemCam) bestehend aus einer digitalen Miniaturkamera und angebrachtem ubiquitären Rechner- und Sensorsystem. Rechts: Datenakquise und -verarbeitung auf der RemCam durch Echtzeit- und Nicht-Echtzeitprozesse

von Mikrofon, Bewegungs-, Licht- und Temperatursensoren werden periodisch in Echtzeit akquiriert und der Situationserkennung, einem eingebetteten Expertensystem, zur Verarbeitung bereitgestellt. Bei Übereinstimmung mit gespeicherten Mustern löst die Kamerasteuerung die Kamera aus. Trotz schwankendem Ausführungsverhalten der Situationserkennung, soll die Verarbeitung zeitnah zur regelmäßig ablaufenden Datenakquise stattfinden. Beide, Datenakquise und Verarbeitung, müssen kollaborieren.

5.10.1 Prozesse und Datenverarbeitung

Ein Prozess besteht aus dem Echtzeitservice als periodische Wurzel, einem Puffer und den daran angeschlossenen datengetriebenen Services. Die Abbildung 5.19 zeigt Prozesse und den Aufbau der Datenverarbeitung. Jedem Sensor ist ein periodischer

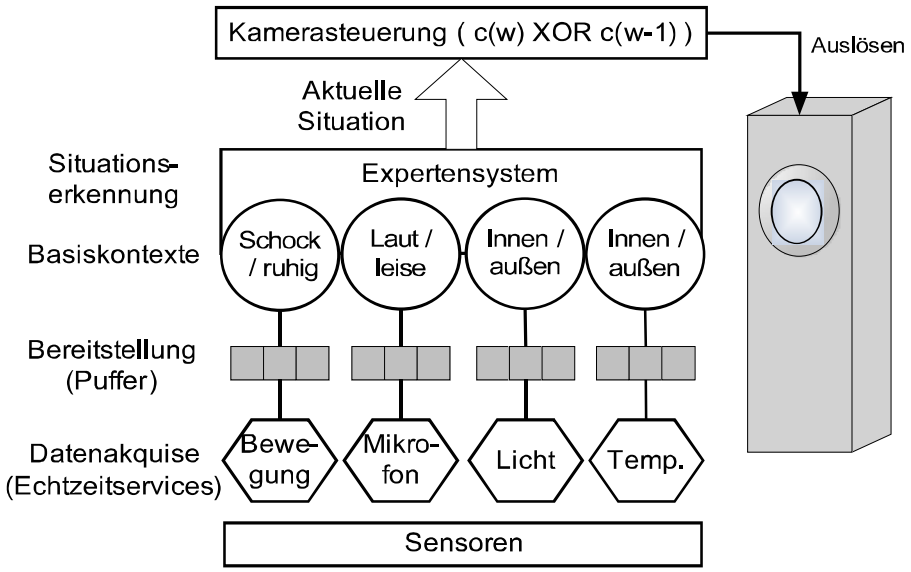


Abbildung 5.19: Prozesse und Datenverarbeitungsschritte der RemCam

Echtzeitservice zur Akquise von Sensorinformationen zugeordnet. Diese Daten werden einzeln in Puffern den datengetriebenen Services zur Verfügung gestellt. Die

Puffer trennen beide Servicetypen horizontal in zwei Schichten wie im Entwurf in Abschnitt 5.4 vorgesehen.

Die Verarbeitung der Daten erfolgt in einem zweistufigen Verfahren. Zuerst werden einfache Basiskontextinformationen extrahiert. Im zweiten Schritt werden mehrere Basiskontexte in einem Expertensystem kombiniert und mit zuvor trainierten Mustern der Ereignisse und Situationen verglichen [10]. In beiden Verarbeitungsstufen variieren die Ausführungsdauern in Abhängigkeit der Eingabedaten. Die aktuelle Situationsinformation $c(w)$ wird mit einer XOR Operation mit der zuletzt ermittelten $c(w-1)$ verrechnet. Bei Situationsveränderung wird die Kamera ausgelöst.

In Abbildung 5.19 lassen sich vier Prozesse unterscheiden. Expertensystem und Kamerasteuerung bilden die gemeinsame Datensenke und werden mit ihren Ausführungsverhalten einem der Prozesse zugeordnet. Es werden immer alle Services ausgeführt. Basiskontexte und ihre Kombinationen für die Situationserkennung in der RemCam sind in den Tabellen 5.10 und 5.11 zusammengefasst.

Sensorservice	**Basiskontext**
Beschleunigung	Erschütterung, ruhig/aufgeregt
Licht	innerhalb/außerhalb eines Gebäudes
Temperatur	innerhalb/außerhalb eines Gebäudes
Mikrofon	laut, sprechen, Ruhe

Tabelle 5.10: Sensoren und extrahierte Basiskontextinformationen

Situation	**Kombination der Basiskontexte**
arbeiten	{ruhig, innen, Ruhe}
rennen	{Erschütterung, außen}
stehen (außen)	{außen, ruhig}
Meeting	{innen, sprechen}
Arbeitspause	{Erschütterung, innen}
Arbeitsunterbrechung	{aufgeregt, innen , sprechen oder laut}

Tabelle 5.11: Situation und Ereignisse, die im Expertensystem aus den Basiskontexten abgeleitet werden.

5.10.2 Kollaborative Prozessorganisation

Zur Situationserkennung kombiniert die RemCam Basiskontextinformationen. Stehen nicht alle diese Teilinformationen zeitnah zur Verfügung, kommt es zu Verzögerungen oder sogar zu fehlerhaften Situationserkennungen. Feststellen und Berichtigen dieser Ausnahmen benötigen zusätzliche Rechenzeit und Speicheraufwand, die die Leistung der RemCam Anwendung mindern. Für eine hohe Anwendungsqualität ist es wichtig, dass die Situationserkennung häufig ausgeführt wird und keine Teilinformationen fehlen. Dies wird durch kollaborative Prozessorganisation mit BFS Regelung für Echtzeitprozesse erreicht.

Das folgende Beispiel soll die Prozessorganisation illustrieren. Die RemCam erkennt die Situation ARBEITEN. Es erfolgt eine Arbeitsunterbrechung, die sensorisch erfasst wird. Die RemCam muss die Situation ARBEITSUNTERBRECHUNG erkennen und die

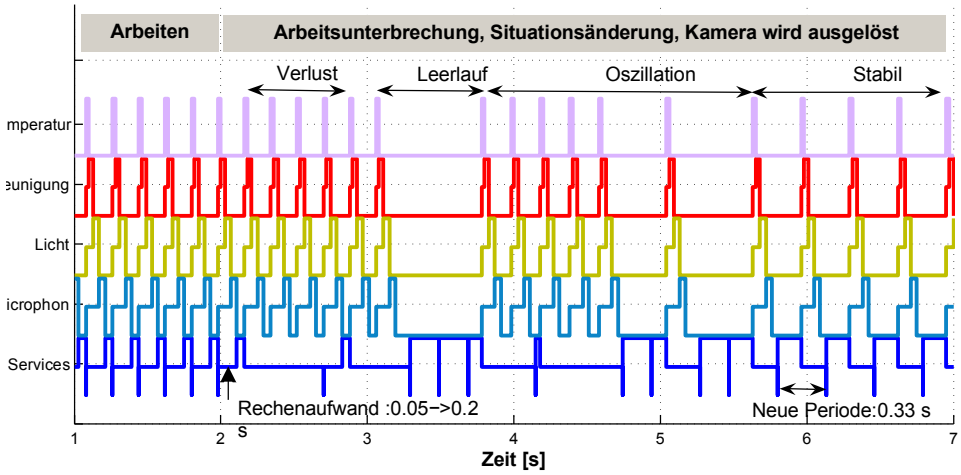


Abbildung 5.20: Aufzeichnung der Serviceaktivierungen unter BFS Regelung beim Übergang der Situation von ARBEITEN zu ARBEITSUNTERBRECHUNG. Die Kurve, die am nächsten zur Zeitachse liegt, beschreibt alle datengetriebenen Services.

Kamera auslösen. Abbildung 5.20 zeigt die zeitliche Abfolge von Serviceaktivierungen. Zur anfänglichen Ausführung aller Services mit Periode von 0.18 Sekunden (s) tritt kein Datenverlust oder Leerlauf auf. Mit Beginn der Arbeitsunterbrechung im Zeitpunkt $t = 2$s werden Sensorinformationen zu neuen Basiskontexten gemäß Tabelle 5.10 verarbeitet und im Expertensystem kombiniert. Der Rechenzeitaufwand steigt von 0.05s auf 0.2s an. Echtzeitservices liefern nun schneller Daten als diese verarbeitet werden und Datenverluste treten an den Puffern auf. Der BFS Regler erkennt den Verlust und dehnt die Perioden der Echtzeitservices. Zum Zeitpunkt $t = 5.7$s hat der BFS Regler die neue Periode von 0.33s stabilisiert. Abbildung 5.21 (links) zeigt für das Beispiel den Verlauf der BFS geregelten Periodenveränderung. Es tritt ein Überschwingverhalten auf und eine zeitweise Oszillation bis sich das System stabilisiert. Das Systemverhalten ist stabil und alle Zeitschranken werden eingehalten.

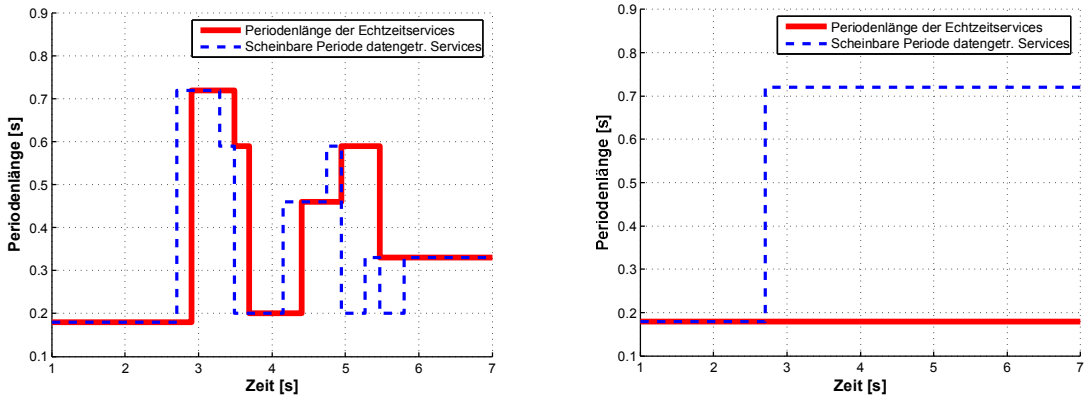


Abbildung 5.21: Auswirkung der BFS Regelung. Links: BFS Regelung der Perioden während des Übergangs zur Situation ARBEITSUNTERBRECHNUNG. Neue Periode nach Anpassung ist 0.33s. Rechts: Periodenverhalten ohne BFS Regelung. Datengetriebene Services laufen mit Periode 0.72s mehr als zweimal langsamer.

Zum Vergleich zeigt Abbildung 5.21 (rechts) das Verhalten ohne Regelung. Die scheinbare Periode der datengetriebenen Services wird durch die Unterbrechung der Echtzeitservices *übermäßig* gedehnt. Die Datenverarbeitung erfolgt mit Periode 0.72s viel langsamer als die Bereitstellung durch die Echtzeitservices mit Periode 0.18s. Permanenter Datenverlust an den Puffern tritt auf.

Das Beispiel zeigt einen *signifikaten Leistungssprung* unter kollaborativer Prozessorganisation. Die datengetriebenen Services werden mehr als zweimal häufiger ausgeführt als im ungeregelten Fall. Bei Kollaboration mit BFS Regelung tritt kein permanenter Datenverlust oder Leerlauf auf.

5.11 Was wurde erreicht?

Dieses Kapitel zeigte, wie das Konzept von Kooperation und Kollaboration durch rückgekoppelte Regelkreise aus Kapitel 3 erfolgreich für Echtzeitprozesse auf ubiquitären Rechnersystemen umgesetzt werden kann.

Zentral ist die *zeitnahe* Informationsverarbeitung zwischen Teilen eines Prozesses mit unterschiedlichen Laufzeitverhalten unter unbekannten Ausführungsbedingungen. Ziel ist es, unter diesen Umständen die Datenverarbeitung effektiv zu gestalten. Kollaboration, die Kopplung der Prozessteile, ist der Lösungsansatz.

Buffer Feedback Scheduling (BFS), modelliert in Abschnitt 5.5, berücksichtigt im Rahmen des dynamischen Echtzeitschedulers EDF zusätzlich den Datenaustausch mit nicht-echtzeitfähigen Prozessteilen. Bewertung und Regelung des zeitnahen Ausführungsverhalten von Echtzeitprozessen wurden formuliert. Harte Echtzeit wird durch eine erweiterte Schedulinganalyse in Abschnitt 5.5.4 auch für geregelte Prozesse garantiert.

BFS wurde in Abschnitt 5.6 detailliert untersucht und Grenzen für die zuverlässige Anwendbarkeit gefunden. Kollaboratives Verhalten der Prozessteile kann in unbekannten, sich ändernden Umgebungen hergestellt werden. Höchste Prozessorauslastung wird erreicht und die Datenverarbeitung ist *maximal effektiv*.

Echtzeitprozesse können mit den bereits im Kapitel 4 eingeführten Instrumenten der serviceorientierten Modellierung dargestellt werden. Abschnitt 5.8 zeigt, dass BFS und der Echtzeitscheduler EDF sich als zwei Komponenten nahtlos in das Betriebssystems Particle OS integrieren lassen.

Eine Anwendung der kollaborativen Prozessorganisation wurde im Beispiel Remembrance Camera (RemCam) gezeigt. In Abschnitt 5.10.2 konnte eine signifikante Leistungsteigerung der Geschwindigkeit der Situationserkennung der RemCam um *mehr als Faktor* 2 nachgewiesen werden. In unbekannten, sich schnell ändernden und mobilen Umgebungen ermöglicht es der RemCam, Situationen *schneller* zu erkennen und zu reagieren. Die Qualität der Appliance steigt bei gleichem Ressourcenaufwand.

6. Verteilte Prozesse

Die Vernetzung von Rechnersystemen erhöht deren Nutzen [110]. *Drahtlos vernetzte ubiquitäre Rechnersysteme* können die Umgebung über größere Räume detaillierter erfassen und neue Zusammenhänge durch gegenseitige Ergänzung von bestehenden Informationen gewinnen. Verteilte Prozesse kommunizieren über Rechnersysteme hinweg. Sie nehmen Informationen entfernter ubiquitärer Rechnersysteme entgegen, verarbeiten diese und verteilen ihrerseits die Ergebnisse an entfernte Systeme. Die Diversität der Prozessaufgaben und hochdynamische Netztopologien mit wechselnden Kommunikationsbeziehungen lassen kein einheitliches Koordinationsschema im Rechnernetz zu. Ziel dieses Kapitels ist, die Prozesse der Informationsverarbeitung mit mehreren ubiquitären Rechnersystemen unter diesen Einsatzbedingungen zu organisieren.

Zentraler Aspekt der Prozessorganisation ist die *zeitnahe Informationsverarbeitung* durch zeitlich abgestimmte Kommunikationsaktivitäten der Rechnersysteme. Anwendung findet dabei die Theorie der kooperativen und kollaborativen Prozessorganisation ubiquitärer Systeme aus Kapitel 3. Kollaboration und Kooperation etablieren einen Verbund, in dem Prozesse den Nachrichtenaustausch gemeinschaftlich, aber *ohne eine gemeinsame Synchronisation*, organisieren. Regelkreise überprüfen und regulieren dynamisch die Kommunikationsaktivitäten im Verbund. Es wird ein signifikant höherer Nachrichtendurchsatz bei kleinerer Verzögerung erreicht. Gleichzeitig bleiben alle anwendungsspezifischen Prozesseigenschaften, zum Beispiel Kommunikationshäufigkeit, Ausführungsperioden weiterer lokaler Prozesse, eines jeden teilnehmenden ubiquitären Rechnersystems erhalten.

Zu Beginn werden Eigenschaften verteilter Prozesse analysiert und Organisationsverfahren aus dem wissenschaftlichen Umfeld untersucht. Der serviceorientierte Systementwurf entwickelt und erklärt die Komponenten für den Nachrichtenaustausch vernetzter ubiquitärer Rechnersysteme. Mit FCUP-MAC wird ein neues Koordinationsverfahren für den Kanalzugriff vorgeschlagen. Die Umsetzung in das Budget/Kosten Regelkreismodell aus Abschnitt 3.4 formuliert die kollaborative und kooperative Organisation verteilter Prozesse mittels FCUP-MAC. Die Modellanalyse erlaubt die Ableitung der Kommunikationsaktivitäten für eine zeitnahe Informationsverarbeitung. Simulationen dienen der Untersuchung und Bewertung der Orga-

nisationsmechanismen in kleinen wie auch großen Einsatzszenarien. Eingang findet das Anwendungsbeispiel Collaborative Business Items (CoBIs) [6][111] sowie technische Parameter der Particle Computer Plattform. Es werden praktische Aspekte der Realisierung diskutiert.

6.1 Motivierendes Beispiel

Collaborative Business Items [6] sind physikalische Objekte mit eingebetteten, ubiquitären Rechnersystemen. In Abbildung 6.1 tauschen CoBIs Nachrichten miteinander aus, um gefährliche Materialkombinationen oder falsche Lagerungsbedingungen beim Umgang mit chemischen Containern zu detektieren. Eine feste Kommunikationsinfrastruktur und Vorwissen über Inhalt und Funktion der mobilen Container können nicht vorausgesetzt werden. Die Rechnersysteme führen jeweils noch weitere

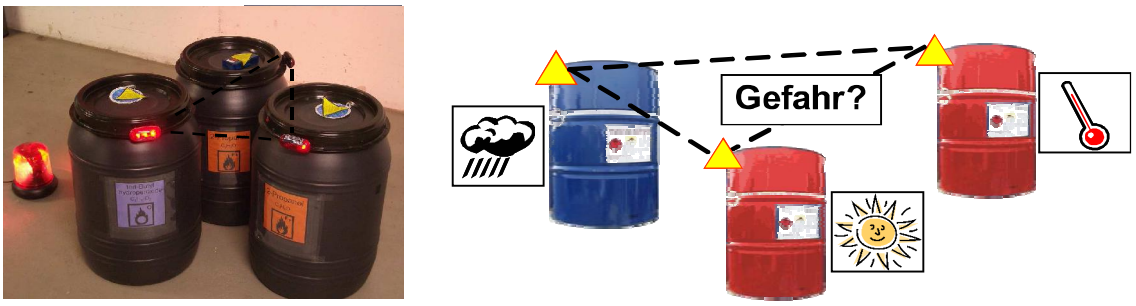


Abbildung 6.1: Collaborative Business Items (CoBIs). Links: Implementierungsbeispiel verteilter Gefahrendetektion. Rechts: Eingebettete Rechnersysteme mit Einzelprozessen, zum Beispiel Licht-, Temperatur-, Feuchtigkeitserfassung, und gemeinschaftlicher Detektion gefährlicher Stoffkombinationen.

Prozesse zur Erfassung von Licht-, Temperatur- und Feuchtigkeitsdaten aus. Alle Systeme haben ein jeweils eigenes Prozessausführungsverhalten. Sie müssen im Verbund den Nachrichtenaustausch kollaborativ regulieren, um unter *Beibehaltung des jeweils zeitlichen Ausführungsverhaltens* eine zeitnahe Datenverarbeitung für eine erfolgreiche Gefahrendetektion zu realisieren.

6.2 Analyse und Einordnung ins wissenschaftliche Umfeld

Prozesse, die über verschiedene ubiquitäre Rechnersysteme hinweg miteinander Nachrichten austauschen, werden als verteilte Prozesse bezeichnet. Sie nehmen Informationen entfernter ubiquitärer Rechnersysteme entgegen, verarbeiten diese und verteilen ihrerseits die Ergebnisse an entfernte Systeme. Eingebettete, ubiquitäre Rechnersysteme kommunizieren drahtlos und bilden ein Kommunikationsnetzwerk. Alle Kommunikationsaktivitäten eines Rechnersystems werden in einem singulären Kommunikationsprozess gebündelt. Drahtlos vernetzte ubiquitäre Rechnersysteme werden im folgenden auch als Netzwerkstationen oder einfach nur Stationen bezeichnet. Verteilte Prozesse entsprechen der Menge der miteinander kommunizierenden Netzwerkstationen.

6.2.1 Nachrichtenaustausch

Informationen und Daten werden in Form von Nachrichten ausgetauscht, die in Paketen übermittelt werden. Für ressourcenbeschränkte Rechnersysteme entspricht die Nachrichtengröße oftmals der durch das Kommunikationsprotokoll realisierten Paketgröße [112]. Alle Netzwerkstationen kommunizieren über ein geteiltes Medium. Zusätzlich verfolgen die Stationen jeweils eigene Kommunikationsintervalle oder sind zeitweilig abgeschaltet, um bei Batteriebetrieb Energie zu sparen.

Koordination ist notwendig, um (1). Stationen zu Kommunikationspartnern zusammenzuführen und um (2). Kollisionen von gleichzeitig gesendeten Nachrichten zu verhindern. Zwei Verfahren finden dabei Anwendung.

Synchronisation stellt einen gemeinsamen zeitlichen Gleichlauf her, nach dem die Kommunikationsaktivitäten aufeinander abgestimmt werden. Dabei werden die Zeitabschnitte vereinbart, in denen Stationen Kommunikationspartner haben und Kommunikation stattfinden kann.

Arbitrierung legt anschließend fest, welche Station Nachrichten sendet und welche sie empfängt.

In einigen Fällen gibt es keine Arbitrierungsphase, sondern die Synchronisation legt implizit fest, welche Station sendet und empfängt.

6.2.2 Eigenschaften verteilter Prozesse auf ubiquitären Rechnersystemen

Ubiquitäre Rechnersysteme mit verschiedenen Prozessaufgaben müssen in unbekannten Einsatzumgebungen zusammenarbeiten. Die verteilte Prozessorganisation wird wie folgt charakterisiert.

Netzwerkparameter. Rahmenbedingungen wie die Ansprechbarkeit, Kommunikationspartner, Nachrichtenaufkommen, lokale Prozessanforderungen und die Mobilität der Stationen können nur ungenau oder gar nicht zur Entwurfszeit angegeben werden.

Synchronisation. Verteilte Prozesse haben keine gemeinsame Kommunikationsperiode. Jede Station hat ihre eigene Kommunikationsperiode, die nicht verändert wird. Neben der Kommunikation werden weitere Prozesse ausgeführt, so dass für jede Station ein einzigartiges Ablaufverhalten entsteht. Verteilte Prozesse nutzen gemeinsam den selben Kommunikationskanal zum drahtlosen Nachrichtenaustausch. Unter *Beibehaltung des jeweiligen zeitlichen Ablaufverhaltens* muss Kanalzugriff und Austausch koordiniert werden. Verteilte Prozesse können nicht über mehrere Rechnersysteme hinweg synchronisiert werden.

Topologie. Eine Topologie der Kommunikationsbeziehungen wird nicht vorausgesetzt. Konkrete Kommunikationspartner sind sich beim Entwurf nicht bekannt. Der Verbund ist lose organisiert. Die Teilnehmer eines Netzwerkes sind nicht fest vorgegeben. Es können weitere Stationen hinzukommen oder sich auch aus der Kommunikationsreichweite bewegen oder abgeschaltet werden.

Adressierung. Als Folge der ungewissen Topologie ist die Kommunikation der Prozesse datenorientiert, d.h. Prozesse unterschreiben auf Nachrichtentypen, die gleichzeitig der Adressierung der Systeme dienen. Ein Beispiel für datenorientierte Adressierung, oftmals auch als semantische Adressierung bezeichnet, ist ConCom [113]. Es gibt im allgemeinen keine dedizierte Nachrichtensenke, die alle Nachrichten sammelt. Alle verteilten Prozesse sind gleichberechtigte Kommunikationspartner, die Nachrichten aufnehmen, verarbeiten und weiterleiten.

Energie. Kommunizierende Prozesse haben den größten Energiebedarf batteriebetriebener eingebetteter Rechnersysteme [114]. Ist der Energievorrat einzelner Stationen aufgebraucht, dann können Netzwerke partitionieren. Für längstmögliche Koordination müssen verteilte Prozesse die Energievorräte der Stationen egalisieren.

Damit Nachrichten zeitnah von verteilten Prozessen verarbeitet werden, muss ein *unmittelbarer und zügiger Nachrichtenaustausch* erfolgen. Im unsynchronisierten Nebeneinander der Kommunikationsaktivitäten muss dazu die Nutzung des gemeinsamen Kommunikationskanals koordiniert werden.

6.2.3 Organisationsmechanismen

Die Organisation der Nutzung gemeinsamer Ressourcen ist eine zentrale Fragestellung im Bereich der verteilten Systeme wie auch im Bereich der mit ubiquitären Rechnersystemen eng verwandten drahtlosen Sensornetzwerke. Die Abbildung 6.2 vergleicht die Organisationsmechanismen beider Bereiche miteinander. Es lassen sich Synchronisations- und Arbitrierungsphase unterscheiden. In verteilten Systemen

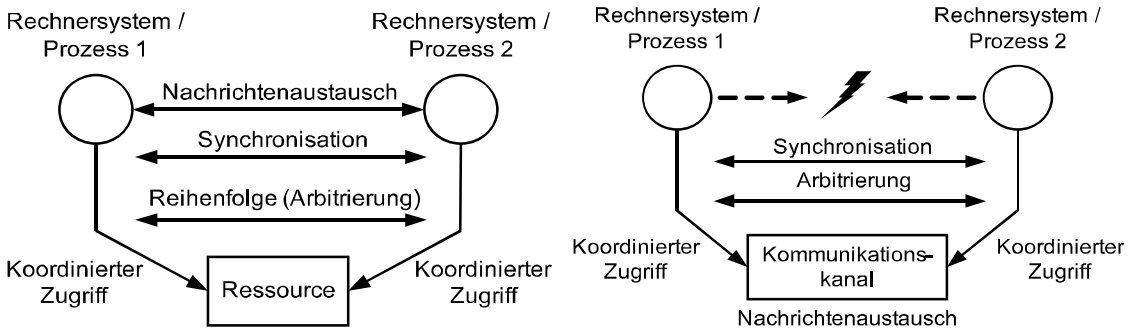


Abbildung 6.2: Organisation des gemeinsamen Ressourcenzugriffs. Links: verteilte Systeme Rechts: drahtlos kommunizierende Sensorsysteme

werden mittels Nachrichtenaustausch die Systeme synchronisiert, um eine Reihenfolge für den Zugriff auf eine gemeinsame Ressource, wie zum Beispiel ein Drucker oder verteilte Dateien, herzustellen. In Sensornetzwerken ist die gemeinsam genutzte Ressource der Kommunikationskanal. Nachrichtenaustausch kann erst nach der Koordination erfolgen. Durch Signalisierung wird Synchronisation und Arbitrierung für einen koordinierten Kanalzugriff der Stationen realisiert.

In Sensornetzwerken ist Koordination ein Problem der Sicherungsschicht, Schicht 2 im ISO/OSI Standardmodell, während sie in verteilten Systemen in der Anwendungsschicht des Internet-Modells, realisiert ist.

In den folgenden Abschnitten werden die Mechanismen beider Bereiche detaillierter betrachtet. Mit einer Diskussion der Eignung für drahtlose vernetzte ubiquitäre Rechnersysteme wird der Abschnitt abgeschlossen.

Organisation in verteilten Systemen

In verteilten Systemen arbeiten verschiedene Prozesse zusammen, um gemeinsam genutzte Ressourcen koordiniert anzusprechen. Zum Beispiel müssen sich Prozesse beim Zugriff auf einen gemeinsam genutzten Drucker einander einen zeitlich exklusiven Zugriff erteilen. Änderungsdaten verteilter Dateien müssen zeitlich synchronisiert sein, damit sie für Softwarewerkzeuge, wie zum Beispiel *make*, vergleichbar sind. Anwendung findet dabei die *Uhrensynchronisation*, die die lokalen Zeitgeber von Rechnersystemen aufeinander synchronisiert. Ein *zentraler Zeitserver* stellt die verbindliche Uhrzeit für Klienten bereit. Netzwerkverzögerungen und Verzögerungen im Klienten verursachen Ungenauigkeiten und werden zum Beispiel durch den Algorithmus Cristians [115] oder den Berkeley-Algorithmus [116] ausgeglichen. Das gebräuchliche Network Time Protocol (NTP) ist auf der Anwendungsschicht im Internet-Schichtenmodell implementiert und erreicht eine Genauigkeit zwischen 1 bis 50 Millisekunden [28]. Auf Basis synchronisierter Uhren kann dann eine Zugriffsreihenfolge festgelegt werden, die den koordinierten Ablauf realisiert.

Organisation in drahtlosen Sensornetzwerken

In drahtlosen Sensornetzwerken steht die gemeinschaftliche Nutzung des Kommunikationskanals im Mittelpunkt der Organisation. Die Kommunikationsaktivitäten der Stationen müssen synchronisiert werden, damit Sender- und Empfängerstationen gleichzeitig zum Nachrichtenaustausch bereit sind. Die Kanalnutzung durch die Senderstationen muss zeitlich nacheinander erfolgen, um Nachrichtenkollisionen zu vermeiden. Medienzugriffsverfahren, engl. MAC = media access control, implementieren den koordinierten Kanalzugriff auf Schicht 2, der Sicherungsschicht im ISO/OSI Standardmodell.

In [117] wird eine Einteilung der MAC Verfahren in die vier Klassen, Zeitschlitzorganisation, Zeitrahmenorganisation, Direktzugriff und Hybridzugriffsverfahren, vorgenommen. Alle Ansätze versuchen die Kanalnutzung einzelner Stationen und damit den Energieaufwand zum Erreichen eines erfolgreichen Nachrichtenaustausches zu minimieren.

Zeitschlitzorganisation Kommunikationsaktivitäten der Stationen werden auf festgelegte Zeitschlitze synchronisiert. Zu Beginn eines Zeitschlitzes werden alle wartenden Nachrichten ausgetauscht. Kollisionen werden durch Wettbewerb behandelt. Für die verbleibende Zeitschlitzdauer wird der Kommunikationskanal nicht genutzt. Synchronisation kann zentral durch eine ausgewählte Station vorgegeben werden oder verteilt in gemeinschaftlicher Übereinstimmung der teilnehmenden Stationen erreicht werden.
Vorteile: gemeinsamer, deterministischer Kommunikationsschedule erlaubt Garantien in Übertragungszeit, hohes Energiesparpotential
Nachteile: höhere Kollisionswahrscheinlichkeit, da alle Stationen zu Beginn des Zeitschlitzes kommunizieren
Beispiele: S-MAC [118][119], T-MAC [120], SCP-MAC [121]

Zeitrahmenorganisation In dieser Organisationsform werden Zeitschlitze in längere Zeitrahmen gruppiert. Stationen werden Schlitze zugeteilt, in denen sie Nachrichten austauschen. Die Zuteilung erfolgt verteilt oder über eine zentrale Station.
Vorteile: Zeitschlitzzuteilung verhindert Kollisionen, hohe Kanalnutzung möglich
Nachteile: Zeitsynchronisation auf mehreren Ebenen, aufwendiger in Zeitschlitzselektion, für relativ starre Netztopologien geeignet
Beispiele: LMAC [122], PEDAMACS [123], TRAMA [124]

Direktzugriff ohne Synchronisation Ohne vorherige Synchronisation wird auf den Kanal zugegriffen. Trägererkennung in den Senderstationen dient der Detektion einer Sendemöglichkeit. Trägererkennung in den Empfangsstationen dient der Koordination der Kommunikationspartner. Bei Kollisionen erfolgt Sendewiederholung.
Vorteile: sehr einfach zu implementieren, hohe Flexibilität bei Änderungen der Stationsanzahl, des Nachrichtenaufkommens und der Kommunikationsbeziehungen
Nachteile: häufiges Abhören des Kanals oder lange Präambel des Senders (= Verzögerung), hoher Energieaufwand
Beispiele: B-MAC [125], WiseMAC [126], Preamble Sampling [127]

Hybridzugriffsverfahren Diese Organisationsform kombiniert Direktzugriff und Zeitrahmensynchronisation. Die Stationen beobachten das Nachrichtenaufkommen und schalten zwischen den Verfahren um oder senken die Synchronisationsrate.
Vorteile: flexibel hinsichtlich Nachrichtenaufkommen und Topologieänderungen
Nachteile: höhere Implementierungskomplexität durch Akquise von Beobachtungsdaten und Entscheidungsfindung
Beispiele: Z-MAC [128], PMAC [129], Crankshaft [130]

Eignung für den Einsatz auf vernetzten ubiquitären Rechnersystemen

Die Tabelle 6.1 untersucht die Organisationsmechanismen hinsichtlich ihrer Eignung für verteilte Prozesse auf ubiquitären Rechnersystemen. Für verteilte Prozesse wie auch für Sensornetzwerke bildet der Kommunikationskanal die zugriffskoordinierte Ressource. Die Untersuchung in Tabelle 6.1 beschränkt sich daher auf die Taxonomie der MAC Protokolle von Sensornetzwerken.

Organisation	**Eignung**	**Erläuterung**
Zeitschlitz	ungeeignet	(−) Zeitschlitze erfordern gemeinsame Kommunikationsperiode der Prozesse.
Zeitrahmen	teilweise	(+) Zuteilung ermöglicht flexiblere Kommunikationsperioden, (−) dynamische Netztopologien erschweren Etablierung der Rahmen
Direktzugriff	teilweise	(+) unsynchronisiertes Verhalten erlaubt Nebeneinander verschiedener Prozess- und Kommunikationsaktivitäten, (−) häufige Trägererkennung erfordert kleine Kommunikationsperioden
Hybridverfahren	geeignet	(+) Kombinationsverfahren erlauben effektives Nebeneinander verschiedener Kommunikationsaktivitäten in dynamischen Netztopologien

Tabelle 6.1: Eignung verschiedener Synchronisationsmechanismen zur Organisation von Prozess- und Kommunikationsaktivitäten ubiquitärer Rechnersysteme

6.2.4 Fazit und Lösungsidee

Als Resultat der Analyse verteilter Prozesse auf drahtlos miteinander kommunizierenden ubiquitären Rechnersystemen ist festzuhalten

- Verteilte Prozesse tauschen Nachrichten über einen gemeinsam genutzen Kommunikationskanal aus.
- Die zeitnahe Datenverarbeitung erfordert einen unmittelbaren und zügigen Nachrichtenaustausch.
- Unsynchronisierte Kommunikationsaktivitäten müssen zum Nachrichtenaustausch organisiert werden.
- Der Energievorrat beschränkt die Fähigkeit zum Nachrichtenaustausch und die Erreichbarkeit im Netzwerk.

Ziel dieses Kapitels ist es, unter diesen Bedingungen die Informationsverarbeitung der verteilten Prozesse mit mehreren ubiquitären Rechnersystemen zu organisieren.

Die Lösungsidee ist, dass verteile Prozesse kollaborativ und kooperativ in einem *Verbund mit regulierendem Nachrichtenaustausch* arbeiten. Dazu wird ein geregeltes Organisationsverfahren in einem hybriden Zugriffsprotokoll vorgeschlagen. Kollaboration koppelt die verteilten Prozesse via Nachrichtenaustausch. Rückkopplung und Regelung koordinieren die zeitnahe Datenverarbeitung unter unbekannten und dynamischen Netzwerkbedingungen. Kooperativ egalisieren verteilte Prozesse Energiereserven, um eine maximale Kollaborationsdauer zu erreichen. Verteilte Prozesse erreichen damit einen signifikant höheren Nachrichtendurchsatz bei kleinerer Latenz.

6.3 Problemformulierung

Zentraler Aspekt der Organisation verteilter Prozesse ist die *zeitnahe Datenverarbeitung* von Nachrichten über mehrere Rechnersysteme hinweg. Die Verzögerungszeit zwischen Nachrichtenemission auf einer Station und der finalen Verarbeitung auf einer anderen soll klein sein. Das bedeutet, dass der Nachrichtenaustausch zeitlich abgestimmt werden muss. Bei dieser Koordination treten vier Probleme auf, die einen unmittelbaren und zügigen Nachrichtenaustausch behindern: Senden ohne Empfänger, vorübergehend abgeschaltete Stationen, Verzögerung stationseigener Nachrichten und vorzeitiges Aufbrauchen des Energievorrates.

Senden ohne Empfänger. Die Kommunikationsperioden, in denen eine Station Nachrichten senden kann, sind fest vorgegeben. In Abbildung 6.3 (links) ist die Empfängerstation nicht mit der Senderstation synchronisiert und kann daher die Nachricht nicht empfangen. Eine kleine Verlängerung der Kommunikationsdauer $C^{\text{comm.},*} = C^{\text{comm.}} + \Delta$ führt zum erfolgreichen Nachrichtenaustausch. Die Bestimmung der Verlängerung ist ein offenes Problem.

Vorübergehend abgeschaltete Stationen. Sind Kommunikationsperioden noch weiter gegeneinander verschoben, so dass eine Verlängerung der Kommunikationsdauer nicht mehr akzeptabel ist, dann können Stationen über längere Zeiträume nicht mehr erreicht werden. Aus der Sicht des Senders erscheinen sie abgeschaltet.

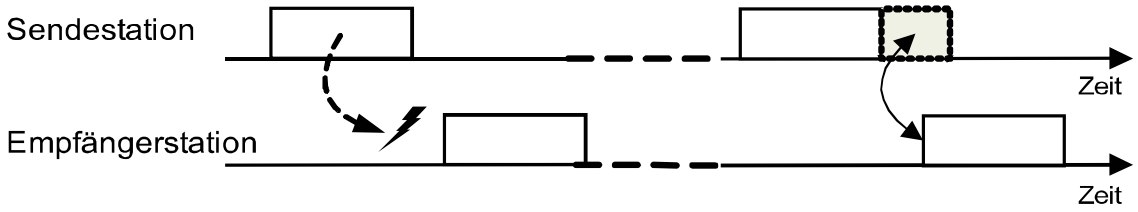


Abbildung 6.3: Links: Kommunikationsaktivitäten und erfolgloser Nachrichtenaustausch zweier Stationen. Rechts: Durch Verlängerung der Kommunikationsdauer kann ein erfolgreicher Nachrichtenaustausch herbeigeführt werden.

Abbildung 6.4 zeigt, dass eine Nachricht im Zeitschritt w nicht an die abgeschaltete Station 2 gesendet werden kann. Sie wird von Station 3 empfangen und in einem späteren Zeitschritt $w + k$ an Station 2 weitergeleitet. Die Anzahl der involvierten Zwischenstationen festzulegen, ausgedrückt durch den Parameter TTL (engl. Time To Live), ist ein offenes Problem. Obwohl durch Weiterleitung der Austausch nicht

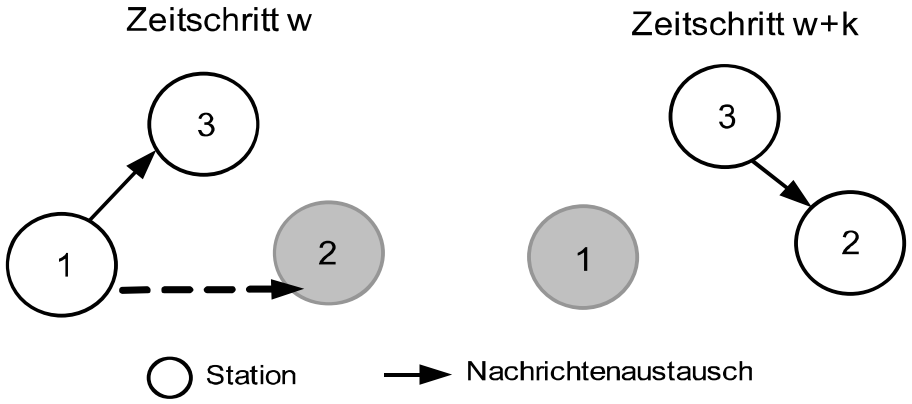


Abbildung 6.4: Die abgeschaltet erscheinende Station 2 wird durch Weiterleitung einer Nachricht via Zwischenstation 3 erreicht.

mehr unmittelbar ist, wird verhindert, dass die Senderstation lange Wartezeiten, aufwendiges Umsortieren oder sogar ein Überlaufen der Nachrichtenschlange riskiert. Zeitnähe ist immer noch gegeben, da die Nachricht auf dem Weg zum Empfänger an der Sendestation nicht unnötig verzögert wird.

Verzögerung stationseigener Nachrichten. Weiterleitung erhöht das Gesamtaufkommen an Nachrichten, die von einer Station empfangen werden. Das Absenden von Nachrichten lokaler Prozesse wird stark verzögert oder sogar unmöglich. Daher muss eine minimale Ansprechbarkeit des Systems definiert werden. Die Kommunikationsdauer $C^{\text{comm.}}$ pro Periode $T^{\text{comm.}}$ ist wie folgt zusammengesetzt: $C^{\text{comm.}} = C^{\text{lok.}} + C^{\text{entf.}}$ mit der minimalen Ansprechbarkeit $C^{\text{lok.}} = \text{konst.}$. $C^{\text{entf.}}$ entspricht der Verlängerung Δ aus dem ersten Problemfall.

Vorzeitiges Aufbrauchen des Energievorrates. Kommunizierende Prozesse haben den größten Energiebedarf batteriebetriebener eingebetteter Rechnersysteme [114]. Ist der Energievorrat E_n einer Station deutlich vor dem der anderen aufgebraucht, so fällt die Station aus und kann die Organisation verteilter Prozesse nicht mehr unterstützen. Der Ausfall der Station kann sogar das Netzwerk partitionieren. Es

muss daher eine Egalisierung aller Energievorräte, d.h. $\forall n : |E_n - E_{n+1}| < \epsilon$ für ein sinnvolles ϵ, zur Laufzeit erfolgen.

Es leitet sich die zentrale Problemformulierung der Organisation verteilter Prozesse in ubiquitären Rechnersystemen ab.

Problem 6.1 (Verteilte Prozessorganisation)**.** *Für eine zeitnahe Datenverarbeitung über mehrere ubiquitäre Rechnersysteme hinweg in Einsatzszenarien mit unbekannter und veränderlicher Netzwerktopologie bestimme die Kommunikationsparameter* $C^{comm.}, C^{lok.}, TTL, C^{entf.}, E_n$ *so, dass*

1. *Nachrichten trotz verschiedener Kommunikationsperioden ausgetauscht werden,*
2. *Nachrichten vorübergehend abgeschaltete Stationen erreichen,*
3. *Mindestansprechbarkeit des Rechnersystems gewährleistet ist und*
4. *Netzwerkpartionierung durch Energieegalisierung verhindert wird.*

Der Lösungsansatz besteht darin, dass verteilte Prozesse in einem Verbund gemeinschaftlich den Nachrichtenaustausch regulieren. Als Konsequenz wird ein erfolgreicher und zügiger Nachrichtenaustausch realisiert. Diese Kopplung beziehungsweise Kollaboration der Systeme via Nachrichten ermöglicht die zeitnahe Datenverarbeitung der Nachrichten. Die Prozesskooperation egalisiert die Energievorräte aller Netzwerkstationen.

6.4 Serviceorientierter Entwurf

Alle Aktivitäten zum Nachrichtenaustausch zwischen verteilten Prozessen werden in einem Kommunikationsservice gebündelt. Verteilte Prozesse erzeugen Nachrichten, die in periodisch stattfindenden Kommunikationsphasen vom Kommunikationsservice mit anderen Rechnersystemen ausgetauscht werden. Kommunikation ist ein periodischer Service, der sowohl den Empfang (RX) wie auch das Senden (TX) behandelt. RX und TX Komponenten bilden eine Wurzel für periodische Prozesse im Servicegraph. Prozesse des Servicegraphen schreiben Nachrichten in einen gemeinsamen Datenpuffer, aus dem sie von TX konsumiert werden. RX erhält die Nachrichten von anderen Rechnersystemen und gibt sie über Puffer an datenorientierte Services anderer Prozesse weiter. Abbildung 6.5 (links) zeigt die Eingliederung der Kommunikation verteilter Prozesse in einen Servicegraphen.

Die Systemarchitektur serviceorientierter Prozesse in Abschnitt 4.3.3 legt den Servicegraph horizontal über die Schichten der Systembibliotheken und die Hardwareplattform. Abbildung 6.5 (rechts) zeigt einen vertikalen Schnitt durch die Schichten. Der Kommunikationsservice übergibt die Daten an einen FIFO Puffer im Laufzeitsystem. Dort befindet sich der Kommunikationsprotokollstack, der Rahmenbildung und den Nachrichtenaustausch über die Kommunikationshardware durchführt. Alle Kommunikationsaktivitäten sind nicht-präemptiv und durchgehend periodisch organisiert.

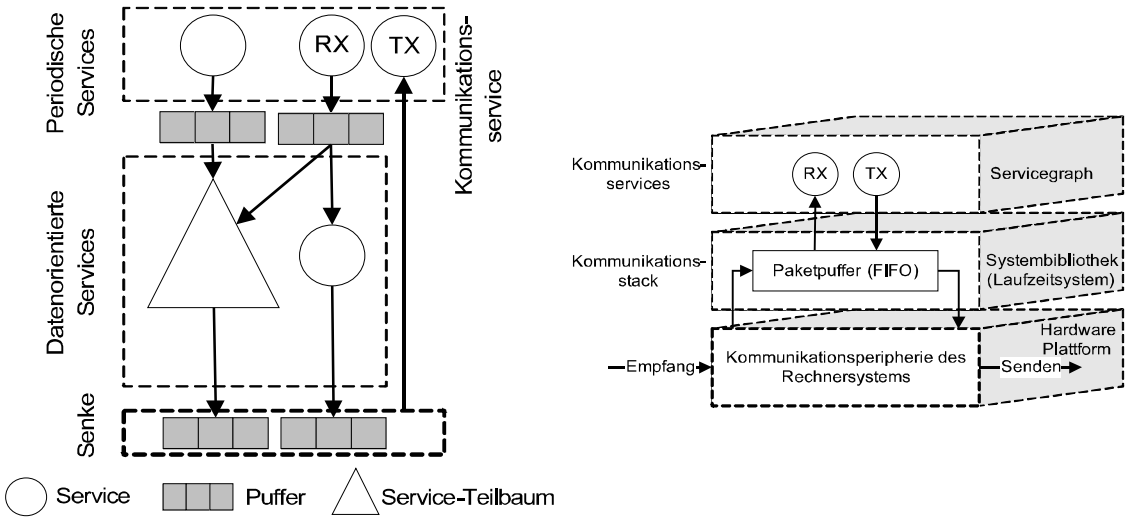


Abbildung 6.5: Links: Eingliederung verteilter Prozesse in den Servicegraph mittels eines periodischen Kommunikationsservice. Rechts: Vertikaler Schnitt durch alle Systemschichten zeigt die Verbindung bis zur Hardwareplattform.

6.4.1 Ablauf der Kommunikation

Durch den Systementwurf reduziert sich die Organisation verteilter Prozesse auf den Kommunikationsservice. Ein Kommunikationsservice hat zwei Aufgaben zu erfüllen.

1. **lokale Kommunikation:** Senden und Empfangen von Nachrichten in Vertretung der lokalen Prozesse des Rechnersystems

2. **Weiterleitung:** Senden und Empfangen von Nachrichten für andere entfernte ubiquitäre Rechnersysteme

Beide Aufgaben werden durch Mediation über einen zentralen FIFO Puffer im Laufzeitsystem vom Kommunikationsservice durchgeführt. Abbildung 6.6 (links) macht den Zusammenhang deutlich. Nachrichten von und zu Prozessen im Servicegraphen wie auch Nachrichten anderer Rechnersysteme werden *im selben Puffer* gespeichert. Der Kommunikationsservice vermittelt die Nachrichten entsprechend an den Kommunikationsstack und die Prozesse weiter. Die Kommunikation mit anderen Rechnersystemen findet in periodisch wiederkehrenden Kommunikationsintervallen statt. Abbildung 6.6 (rechts) illustriert den zeitlichen Ablauf in zwei Phasen. In der ersten Phase werden Nachrichten lokaler Prozesse über eine *feste (statische) Zeitdauer* $C^{\text{lok.}}$ ausgetauscht. In der zweiten Phase erfolgt die Weiterleitung von Nachrichten entfernter Stationen über eine *veränderliche (dynamische) Zeitdauer* $C^{\text{entf.}}$. Eine konkrete Reihung ist nicht notwendig. Die Phasenabschnitte beschreiben lediglich eine Dauer im Kommunikationsintervall.

Die Aufteilung des periodischen Intervalls gewährleistet die Mindestansprechbarkeit des Rechnersystems und Weiterleitung von Nachrichten für Stationen, die vorübergehend nicht erreicht werden können. Es bildet sich ein Verbund, in dem sich Prozesse gegenseitig beim Nachrichtenaustausch unterstützen.

Verantwortlich für den koordinierten Ablauf beim Senden und Empfangen ist das Zugriffsverfahren zur Nutzung des geteilten Kommunikationskanals.

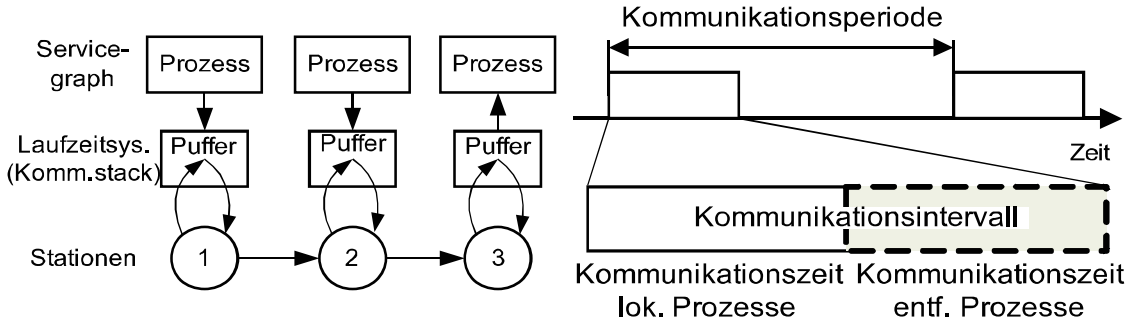


Abbildung 6.6: Links: Senden, Empfangen und Weiterleiten von Nachrichten von entfernten Rechnersystemen und lokalen Prozessen. Rechts: Zeitliche Unterteilung der Kommunikationsaktivitäten in periodische Intervalle und in Zeitabschnitte zum Nachrichtenaustausch lokaler Prozesse und zur Weiterleitung von Nachrichten.

6.4.2 Hybrides Zugriffsverfahren - FCUP-MAC

Kanalzugriffskoordination wird durch *Feedback controlled unsynchronized periodic MAC (FCUP-MAC)* realisiert. FCUP-MAC kombiniert periodische Kommunikationsaktivitäten mit einem wettbewerbsbasierten Direktzugriff auf den Kanal, aber *ohne Zeitrahmensynchronisation*. Es findet eine Organisation auf zwei Ebenen statt. Nachrichten werden in den *periodisch* wiederkehrenden Kommunikationsintervallen gesendet und empfangen. In den Intervallen erfolgt ein direkter *unsynchronisierter* Zugriff auf den Kanal. FCUP-MAC regelt die dynamische Kommunikationsdauer eines Kommunikationsintervalls aus Abbildung 6.6 (rechts)

FCUP-MAC ist wie in Abbildung 6.7 gezeigt auf Schicht 2 (Sicherungsschicht) realisiert. Es umfasst Rahmenbildung, direkten Kanalzugriff durch Verfahren aus Abschnitt 6.2.3 und Regelung der dynamischen Kommunikationsdauer. Nachrichten werden aus dem Puffer entnommen und versendet. Empfangene Datenströme werden vom MAC Protokoll dekodiert und in den Nachrichtenpuffer abgelegt. Neu ist, dass

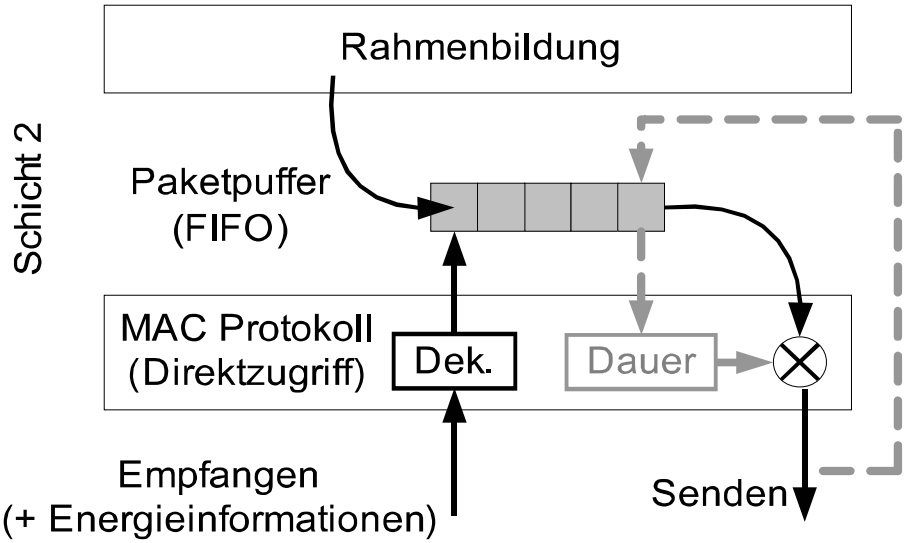


Abbildung 6.7: Schicht 2 Modell des hybriden Zugriffsprotokolls FCUP-MAC. Kommunikationsdauer wird gemäß des Pufferfüllstandes geregelt. Rückkopplung durch Kommunikationsaktivitäten schließt den Regelkreis.

die Kommunikationsdauer entsprechend des Pufferfüllstands geregelt wird. Kommunikationsaktivitäten verändern den Pufferfüllstand und erzeugen die Rückkopplungsinformation für einen geschlossenen Regelkreis. Die Regelung arbeitet vollkommen puristisch basierend auf der Dynamik des Nachrichtenaustauschs. Sie ist unabhängig vom konkreten MAC Protokoll des Kanalzugriffs und der Kommunikationshardware.

FCUP-MAC koppelt durch Regelung den Nachrichtenaustausch verteilter Prozesse mit verschiedenen Kommunikationsperioden. Es entsteht ein kollaboratives Kommunikationsverhalten. Übermittelte Energieinformationen kontrollieren den Grad der Kollaboration. Bei zu geringer Energiereserven wird die Kopplung aufgelöst. Dieser Ansatz dient der Energieegalisierung.

Der Entwurf adressiert alle Teile der Problembeschreibung aus Abschnitt 6.3.

6.5 Systemmodell der Organisation verteilter Prozesse

Um kollaboratives und kooperatives Verhalten zu beschreiben, wird FCUP-MAC als ein Regelkreis nach dem Budget/Kosten Modell modelliert. Zur Lösung der Problemstellung aus Abschnitt 6.3, sollen die verteilten Prozesse im Verbund arbeiten, um das Nachrichtenaufkommen gemeinschaftlich zu behandeln. Aufgabe des Kommunikationsservice ist dabei, das durch die verteilten Prozesse erzeugte Nachrichtenaufkommen zu bewältigen.

Im unmittelbar ersten Modellierungsschritt wird die Methode zur Übertragung des Budget/Kosten Modells auf spezifische Prozessklassen aus Abschnitt 3.4.4 bemüht. Es sind Budget, Kosten, Ziel der verteilten Prozessorganisation zu identifizieren. Sie sind in Tabelle 6.2 zusammengefasst. Der FCUP-MAC FIFO Puffer spiegelt

Budget	Nachrichtenaufkommen
Kosten	Kommunikation mit anderen Rechnersystemen
Ziel	Alle verteilten Prozesse verarbeiten das erzeugte Nachrichtenaufkommen.

Tabelle 6.2: Entsprechungen des Budget/Kosten Modells für die Organisation verteilter Prozesse

das Nachrichtenaufkommen im Netzwerk wider. Das Budget/Kosten Modell findet somit seine direkte Entsprechung im Pufferverhalten von FCUP-MAC. Es wird die konkrete Regelungsaufgabe für FCUP-MAC formuliert.

Regelungsziel: Erreichen eines vorgegebenen Pufferfüllstandes. Dies entspricht einem zu bewältigenden Nachrichtenaufkommen.

Stellgröße: Kommunikationsdauer zur Weiterleitung von Nachrichten entfernter Rechnersystemen

Regelstrecke: Nachrichtenaustausch durch Kommunikation mit anderen Netzwerkstationen. Pufferfüllstand bildet die Rückkopplungsgröße.

Störgröße: Nachrichtenverlust durch unsynchronisierten Nachrichtenaustausch und durch Einflüsse der Netztopologie und des Kommunikationskanals

Obwohl die Regelungsaufgabe sich nur auf ein einzelnes Rechnersystem im Verbund bezieht, berücksichtigt sie jedoch auch das gemeinsame Ziel aus Tabelle 6.2. Das gesamte Nachrichtenaufkommen wird ausschließlich durch Nachrichten in den FCUP-MAC Puffern aller Verbundsysteme erzeugt. Wird jedes Einzelziel erreicht, bedeutet das, dass das von allen verteilten Prozessen erzeugte Nachrichtenaufkommen verarbeitet wird.

6.5.1 Systemgleichung des FCUP-MAC Puffers

Der Kommunikationsservice wird mit fester Periode $T^{\text{comm.}}$ und der Ausführungsdauer $C^{\text{comm.}} = C^{\text{lok.}} + C^{\text{entf.}}$ ausgeführt. Alle Nachrichtenpakete werden über einen Kanal mit Bandbreite W $\left[\frac{\text{Nachrichten}}{\text{Zeiteinheit}}\right]$ übermittelt. W bezeichnet die Anzahl der Nachrichten, die pro Zeiteinheit kommuniziert werden können. Es unterliegt den oben genannten Störeinflüssen.

Kernidee der Modellierung ist: Die Dynamik des Nachrichtenaustauschs ΔW spiegelt sich direkt in der Pufferdynamik ΔN wider. Im Kommunikationszeitintervall $[kw; (k+1)w] = C^{\text{comm.}}$ wird die gesamte Dynamik in die zweite Phase $C^{\text{entf.}}(w)$ verschoben. Es muss also nicht ΔW modelliert werden, sondern für $W = \text{konst.}$ wird die Dynamik mittels $\Delta C^{\text{entf.}}$ ausgedrückt.

Mit der Herleitung aus Anhang J wird folgendes Systemmodell des Nachrichtenaustauschs in Z Transformation etabliert.

$$G(z) = \frac{W}{z-1} \tag{6.1}$$

Mit Umformen der Gleichung 6.1 wird ersichtlich

$$G(z) = \frac{W}{z-1} = z^{-1} W \frac{z}{z-1}$$

$G(z)$ entspricht bis auf einen Faktor der Transferfunktion des Budgets $\frac{z}{z-1}$ im Budget/Kosten Modell.

6.5.2 Bandbreite W von FCUP-MAC

Um Nachrichten erfolgreich auszutauschen, muss arbitriert werden. Alle Stationen sind beim Kanalzugriff gleichberechtigt. Die tatsächliche Bandbreite W einer periodisch kommunizierenden Station gegenüber der maximal möglichen Bandbreite ergibt sich aus dem Erfolg bei der Arbitrierung.

Im Direktzugriff auf den Kanal konkurriert eine sendewillige Station A mit n weiteren sendewilligen Stationen B_n. Es wird eine uniforme Verteilung der Kommunikationsperioden der Netzwerkstationen angenommen. Die Wahrscheinlichkeit für ein erfolgreiches Senden von A berechnet sich nach dem Satz der totalen Wahrscheinlichkeit

$$P_s(A) = \sum_n P(A|B_n) P(B_n)$$

B_n bezeichnet die Sendebereitschaft der Station n. $P(A|B_n)$ entspricht der Kollisionshäufigkeit von A mit B_n bei den jeweiligen Perioden. Mit Datenrate r und Paketlänge der Nachricht l berechnet sich die Bandbreite zu

$$W = P_s \frac{r}{l} \tag{6.2}$$

6.5.3 Regelung

Geregelt wird die zweite Phase des Kommunikationsintervalls aus Abbildung 6.6. Der Regler in FCUP-MAC implementiert ein P Regelungsgesetz. Die Fehlerdifferenz zum Regelungsziel wird entsprechend verstärkt als Änderung der zeitlichen Sende- und Empfangsaktivitäten in die Systemgleichung 6.1 weitergegeben. Die Transferfunktion des Reglers in Z Transformation ist

$$R(z) = \frac{U(z)}{E(z)} = K_p \tag{6.3}$$

Geregelt wird nicht die Kommunikationsdauer $C^{\text{entf.}}$, sondern die Dauer der Sende- und Empfangszeitabschnitte, $C_S^{\text{entf.}}(w)$ und $C_R^{\text{entf.}}(w)$, aus denen sich $C^{\text{entf.}}$ zusammensetzt. Mit bekannten $C^{\text{entf.}} = \text{konst.}$ und $u(w)$ kann das folgende Gleichungssystem für $C_S^{\text{entf.}}(w)$ und $C_R^{\text{entf.}}(w)$ gelöst werden.

$$\begin{aligned} u(w) &= C_R^{\text{entf.}}(w) - C_S^{\text{entf.}}(w) \text{ (siehe Anhang J)} \\ C^{\text{entf.}} &= C_R^{\text{entf.}}(w) + C_S^{\text{entf.}}(w) \end{aligned}$$

Die Sende- und Empfangsdauer berechnen sich zu

$$\begin{aligned} C_R^{\text{entf.}}(w) &= \frac{C^{\text{entf.}} + u(w)}{2} \\ C_S^{\text{entf.}}(w) &= \frac{C^{\text{entf.}} - u(w)}{2} \end{aligned} \tag{6.4}$$

6.5.4 FCUP-MAC Regelkreis

Wegen $C^{\text{entf.}} = C_R^{\text{entf.}}(w) + C_S^{\text{entf.}}(w) = \text{konst.}$, ist der Reglerausgang $u(w)$ beschränkt. Es tritt eine Sättigung $g = C^{\text{entf.}}$ auf, die durch ein entsprechendes Sättigungsglied im Regelkreis berücksichtigt wird. Damit ergibt sich der FCUP-MAC Regelkreis wie in Abbildung 6.8 dargestellt.

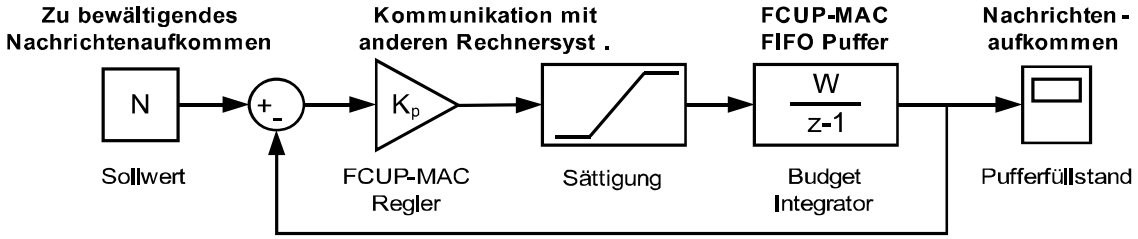


Abbildung 6.8: FCUP-MAC Regelkreis als Simulink Modell.

Es ist in Z Transformation

$$F(z) = \frac{K_p W}{z - 1 + K_p W} \tag{6.5}$$

die Transferfunktion des geschlossenen Regelkreises ohne Sättigungsglied für FCUP-MAC. Das Verhalten bei Sättigung wird in der folgenden Analyse diskutiert.

Mit dem Regelkreis gemäß der Budget/Kosten Modellierung ist es gelungen, Zusammenhänge zwischen (1) Bandbreite und Sollwert eines Pufferfüllstandes und

(2) Bandbreitendynamik und Kommunikationsdauer herzustellen. Zur Entwurfszeit noch unbekannte Kommunikationsbeziehungen werden so flexibel berücksichtigt. Nachrichtenverluste durch Bitfehler sind Störungen des Systemausgangs. Auf ein Kanalfehlermodell kann verzichtet werden, wenn der Regelkreis die Störungen ausreichend schnell kompensieren kann. Aus diesen Gründen ist FCUP-MAC für die Koordination der Kommunikation in unbekannten Einsatzumgebungen geeignet.

6.6 Prozesskollaboration

Verteilte Prozesse sind über den Nachrichtenaustausch miteinander gekoppelt. Sie bilden somit einen Verbund von voneinander abhängigen Nachrichtenlieferanten und -verarbeitern. Ziel ist, dass alle verteilten Prozesse das gemeinschaftlich erzeugte Nachrichtenaufkommen verarbeiten. Nach Definition 3.3 entspricht dieses zielgerichtete Zusammenwirken einer Prozesskollaboration.

Das Nachrichtenaufkommen wird ausschließlich durch die FCUP-MAC Puffer aller Stationen erzeugt. Es ist daher ausreichend, zu zeigen, dass die FCUP-MAC Regelung einer Station das Ziel erreicht. Das Kollaborationsziel wird als Sollwert N des FCUP-MAC Regelkreises aus Abbildung 6.8 quantifiziert. Kollaboration muss sicherstellen, dass ein vorgegebener Pufferfüllstand an Nachrichten behandelt wird.

Der Sollwert ist ein Maß für die Unterstützung des Nachrichtentransports für andere Stationen. Je größer der Sollwert, desto mehr Nachrichten werden von einer Station weitergeleitet. Der FCUP-MAC Regler verstärkt die Kopplung an den Verbund, um das erhöhte Nachrichtenaufkommen zu verarbeiten. Der Sollwert bezeichnet den Grad der Verbundteilnahme respektive den Grad der Kollaboration einer Station.

Dieser Abschnitt prüft, ob das Kollaborationsziel erreicht wird. Dazu wird der Regelkreis aus Abbildung 6.8 und dessen Transferfunktion $F(z)$ aus Gleichung 6.5 analysiert. Es werden wichtige Eigenschaften wie Stabilität, Schnelligkeit und Genauigkeit der Regelung bestimmt.

6.6.1 Stabilität

Es wird ermittelt, ob FCUP-MAC ein konstantes, zu verarbeitendes Nachrichtenaufkommen, einen sogenannten stationären Zielwert $\lim_{w\to\infty} F(w) = F_{ss} = \text{konst.}$, erreicht. Im Fall von Instabilität würde der FCUP-MAC FIFO Puffer überlaufen und der Nachrichtenaustausch mit anderen Stationen unkontrollierbar ablaufen. Das BIBO Stabilitätskriterium untersucht, ob alle Polstellen der Transferfunktion im Einheitskreis der komplexen Zahlenebene liegen.

Im ersten Schritt wird der Einfluss des Sättigungsgliedes diskutiert. Die Sättigung g begrenzt den Eingang des Systems $\frac{W}{z-1}$ auf das Intervall $[-g; g]$. Mit der Transferfunktion des Regler aus Gleichung 6.3 kann K_p so spezifiziert werden, dass für alle $E(z)$ der Reglerausgang auf $-g \leq U(z) \leq g$ eingeschränkt wird.

$$\begin{aligned} \forall g \neq 0 :\ \exists K_p \text{ mit } |K_p E(z)| &\leq |g| \\ K_p &\leq \left| \frac{g}{\max E(z)} \right| \end{aligned} \tag{6.6}$$

Für K_p aus Gleichung 6.6 ist das Sättigungsglied wirkungslos. Nun wird das Stabilitätskriterium untersucht. Die Polstellen von $F(z)$ (ohne Sättigung) aus Gleichung 6.5 sind

$$z = -K_p W + 1 \tag{6.7}$$

$F(z)$ ist für $-1 < z < 1$ BIBO stabil. Es wird K_p für die Pole im Intervall $0 < z < 1$ berechnet, um ein Überschwingverhalten mit kurzzeitiger Verletzung des Zielwertes zu verhindern. Für $K_p > 0, W > 0$,

$$0 < K_p < \frac{1}{W}$$

ist $F(z)$ stabil. Es existiert $\max E(z) < \infty$. Mit der Einschränkung durch die Sättigung aus Gleichung 6.6 ergibt sich die Reglerverstärkung K_p für ein stabiles Verhalten von FCUP-MAC *(mit Sättigung)* zu

$$0 < K_p < \min\left\{\left|\frac{g}{\max E(z)}\right|, \frac{1}{W}\right\} \tag{6.8}$$

mit $\max E(z) = N$. Die Bestimmung von K_p in der Ungleichung 6.8 wird auch als Reglersynthese bezeichnet.

6.6.2 Schnelligkeit

Die Lage der größten Polstelle bestimmt die Geschwindigkeit der Regelung. Die einzige und größte Polstelle ist $z = -K_p W + 1$. Mit $\max K_p = \min\left\{\frac{g}{N}, \frac{1}{W}\right\}$ aus Gleichung 6.8 kann die Polstelle so klein wie möglich gewählt werden. Die Regelung ist dann am stärksten und $F(z)$ ist am schnellsten. Es ist

$$k \approx \frac{\ln 0.02}{\ln(-K_p W + 1)} = \frac{\ln 0.02}{\ln(-\min\left\{\frac{g}{N}, \frac{1}{W}\right\} W + 1)}$$

die Anzahl der Schritte bis das System $F(z)$ 98% des Zielwerts erreicht hat. Um divergente Lösungen zu vermeiden, wird in der Praxis ein Wert knapp kleiner als $\max K_p$ verwendet.

6.6.3 Genauigkeit

Für stabile $F(z)$ mit K_p gemäß Gleichung 6.8, ist $F(1) = \frac{K_p F}{1-1+K_p F} = 1$, d.h. der stationäre Zielwert des Systems F_{ss} erreicht den vorgegebenen Sollwert N ohne stationären Regelungsfehler.

6.6.4 Verhalten bei hohem Paketverlust

FCUP-MAC kann den Sollwert nur erreichen, wenn die Kommunikationsressourcen vorhanden sind. Im Modellierungsansatz wurde die Dynamik der Bandbreite an die Kommunikationsdauer $C^{\text{entf.}}$ delegiert. Schwankt die Bandbreite sehr stark, zum Beispiel gehen viele Nachrichtenpakete verloren, dann kann während der regelbaren Dauer $C^{\text{entf.}}$ der Sollwert nicht erreicht werden. Das modellierte System ist gegenüber der Bandbreitendynamik zu langsam!

Eine weitere Beschleunigung wird nur durch eine größere Reglerverstärkung K_p erreicht. Gegebenenfalls muss hierfür die Sättigungsgrenze g erhöht werden. Dies geschieht durch Vergrößerung von $C^{\text{entf.}}$. Das System erhält so mehr Kommunikationsressourcen.

Die vorherigen Betrachtungen zu Stabilität und Geschwindigkeit haben die Zusammenhänge für die Reglersynthese geliefert. Damit ist es möglich, einen Regler für FCUP-MAC zur Laufzeit zu synthetisieren.

6.6.5 Ergebnisse

Die folgende Tabelle 6.3 fasst die Ergebnisse der vorausgegangenen Abschnitte zusammen. Das Kollaborationsziel $F_{ss} = N = \text{konst.}$ wird erreicht. FCUP-MAC erreicht ein stabiles und kontrolliertes Kommunikationsverhalten durch einen gemeinschaftlich regulierten Nachrichtenaustausch. Die Prozesse arbeiten in einem sich gegenseitig unterstützenden Verbund, in dem sie ihre Kommunikationsaktivitäten koordinieren und Nachrichten anderer weiterleiten.

Eigenschaft	**Systemverhalten**	**Interpretation**
Stabilität	$F(z)$ stabil für $0 < K_p < \min\left\{\left\lvert\frac{g}{\max E(z)}\right\rvert, \frac{1}{W}\right\}$	Kollaboration erreicht konstantes zu verarbeitendes Nachrichtenaufkommen.
Schnelligkeit	Zielwert F_{ss} nach $k \approx \frac{\ln 0.02}{\ln(-K_p W+1)}$ Zeitschritten erreicht.	Geschwindigkeit durch $\max K_p$ beschränkt, weitere Steigerung durch Vergrößerung von $C^{\text{entf.}}$ zur Laufzeit möglich.
Genauigkeit	wenn System stabil, dann wird $F_{ss} = N$ ohne stationären Regelungsfehler erreicht.	FCUP-MAC verarbeitet vorgegebenes Nachrichtenaufkommen.

Tabelle 6.3: Eigenschaften der kollaborativen Prozessorganisation durch FCUP-MAC.

6.7 Prozesskooperation

Der Grad der Kollaboration steht in einem direkten Zusammenhang zum Energiebedarf. Je größer der Sollwert, desto mehr Nachrichten werden von einer Station weitergeleitet und desto schneller wird der lokale Energievorrat vermindert. Fallen Stationen aus, dann unterstützen sie den Nachrichtenaustausch nicht mehr. Der Verbund der kommunizierenden verteilten Prozesse partitioniert.

Die Energievorräte der einzelnen Stationen und damit auch die Wirkung des Nachrichtenaustauschs auf die Energievorräte sind unabhängig voneinander. Ziel ist, die Energievorräte aller Stationen zu egalisieren. Dazu ist der Grad der Kollaboration, d.h. das Nachrichtenaufkommen einer Station im Verbund, entsprechend des Energievorrates der lokalen und aller entfernter Stationen nachzuregeln. Nach Definition 3.2 entspricht dieses zielgerichtete Zusammenwirken einer Prozesskooperation.

6.7.1 Regelkreis der Prozesskooperation

Die Pufferregelung von FCUP-MAC wird in einen Regelkreis zur Energieegalisierung eingebettet. Abbildung 6.9 zeigt die Kaskade der Regelkreise. Entfernte Stationen

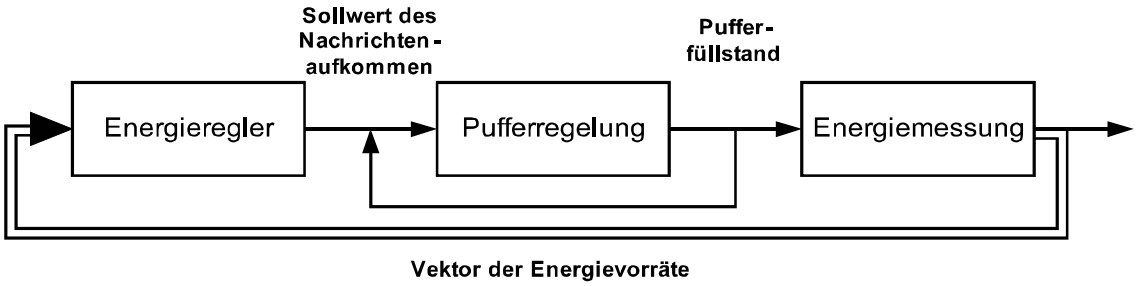


Abbildung 6.9: Einbettung der Pufferregelung von FCUP-MAC in den Regelkreis zu Energieegalisierung

senden mit den Nachrichten auch Informationen über ihren aktuellen Energievorrat. Der Regelkreis ermittelt in der Energiemessung den Energievorrat der lokalen Station und fügt die Angaben der entfernten Stationen zu einem Energievektor $E(w) = (E_1(w), \ldots, E_n(w))$ zusammen. E wird an den Energieregler zurückgeführt. Der Energievorrat wird in Einheiten, die zum Beispiel den noch verfügbaren Anteil der Gesamtkapazität der Batterie eines Rechnersystems bezeichnen, angegeben werden. Die Angabe sollte auf einen entsprechend großen Ganzzahlwert skaliert sein, damit eine feingranulare Regelung möglich ist. Die Messung erfolgt, nachdem der Pufferregler den Sollwert erreicht hat. Der äußere Regelkreis zur Energieegalisierung läuft also mehrere Faktoren langsamer als die Pufferregelung.

6.7.2 Energieregler

Der Energieregler ist ein MISO System. Die Eingabe ist der Vektor $E(w) = (E_1(w), \ldots, E_n(w))$ der Energievorräte der am kollaborativen Prozessverbund beteiligten Stationen. Die Ausgabe ist ein Sollwert für den FCUP-MAC Puffer der lokalen Station des Energiereglers. Da die Energievorräte unabhängig sind, wird vom Energieregler nur das lokale Nachrichtenaufkommen und nicht das entfernter Stationen geregelt.

Der Regler ist ein unstetiger 3-Punkt Regler. Es wird ein für alle Stationen identischer ϵ-Bereich definiert, in dem die Energievorräte als egalisiert betrachtet werden. Das ist das Kooperationsziel der Energieregelung. Das Ziel ist erreicht, wenn die maximale Energiedifferenz aller Stationen in den ϵ-Bereich fällt. Daraus folgt der Ansatz für die Regeldifferenz $D^E(w)$

$$D^E(w) = \epsilon - (E_{\max}(w) - E_{\min}(w)) \tag{6.9}$$

mit $E_{\max}(w) = \max_n E_n(w)$ respektive $E_{\min}(w) = \min_n E_n(w)$ als maximale respektive minimale Elemente des Vektors E. Im Regelungsgesetz vergleicht der Energieregler $R^E(w)$ den Energievorrat der lokalen Station $E^{\text{lok.}}$ mit den maximalen respektive minimalen Elementen und gibt einen Sollwert N an den FCUP-MAC Pufferregelkreis aus:

$$R^E(w) = \begin{cases} N_0, & \text{wenn } D^E(w) \geq 0 \\ N_{\min}, & \text{wenn } D^E(w) < 0 \text{ und } E^{\text{lok.}}(w) = E_{\min}(w) \\ N_{\max}, & \text{wenn } D^E(w) < 0 \text{ und } E^{\text{lok.}}(w) = E_{\max}(w) \end{cases} \tag{6.10}$$

$N_0, N_{\min}, N_{\max}$ sind initial, minimal und maximal zu verarbeitendes Nachrichtenaufkommen und werden vom Entwickler gewählt. Aufgrund des Zusammenhangs zwischen Nachrichtenaufkommen und Energieaufwand der Kommunikation ergeben sich mit der Reglerausgabe Energiegradienten. Für den Fall $N_{\min}$ wird einhergehend mit der minimalen Kommunikationsaktivität der Energieaufwand auf ein Minimum reduziert. Dies vermindert den Betrag des Energiegradienten und die Station steuert wieder in den ϵ-Bereich.

Für den Fall, dass der ϵ-Bereich noch nicht erreicht ist, die Pufferregelung aber schon $N_{\min}$ erreicht hat, kann der Betrag des Energiegradienten durch Reduktion der dynamischen Kommunikationsdauer $C^{\text{entf.}} = C^{\text{entf.}}_{\min}$ weiter vermindert werden. Einhergehend muss gemäß Gleichung 6.8 ein neuer Regler synthetisiert werden. Abbildung 6.10 zeigt die Erweiterung des Energiereglers. Sobald der ϵ-Bereich erreicht ist, wird auf den initialen Wert N_0 gewechselt.

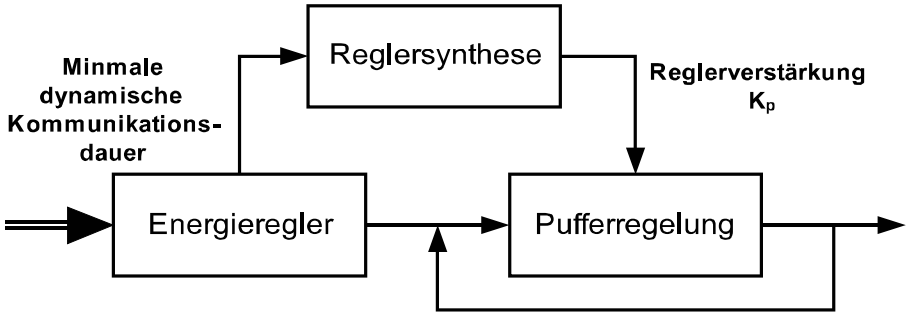


Abbildung 6.10: Erweiterung des Energiereglers, um den Energiegradienten weiter zu vermindern. Es muss ein neuer Pufferregler synthetisiert werden.

6.7.3 Eigenschaften

Die kooperative Energieregelung steuert die Gradienten so, dass die Beträge der Energievorräte im ϵ-Bereich zusammenlaufen. Es kann keine Garantie gegeben werden, dass die Beträge im ϵ-Bereich bleiben. Dies ist ein allgemeines Problem unstetiger Mehrpunktregler. Jedoch wird bei Verlassen des Bereichs sofort gegengesteuert. Es sind Schwingungen der Beträge der Energievorräte zu erwarten.

In einer Kaskade muss der äußere Regelkreis langsamer arbeiten als der innere, um dessen zeitliches Führungsverhalten zu berücksichtigen. Die Regelung der Energieegalisierung erfolgt, *nachdem* der Pufferregler das Nachrichtenaufkommen verarbeitet hat. Der Energieregler ist mehrere Faktoren langsamer als der Pufferregler.

Durch kooperative Energieegalisierung wird die Kollaborationsdauer maximiert.

6.8 Simulationsstudie

Dieser Abschnitt vergleicht das theoretisch vorhergesagte Kollaborations- und Kooperationsverhalten von FCUP-MAC mit den Ergebnissen einer Simulationsstudie. Dazu wird das Kommunikationsverhalten in einem kleinen Netzwerk mit nur vier Stationen analysiert. Die Parameter sind für alle Stationen identisch gewählt. Insbesondere sind die Perioden synchronisiert. Jedoch erfolgt die Arbitrierung durch einen

fairen Wettbewerb. Synchronisation ist keine Vorraussetzung für den Betrieb von FCUP-MAC. Die getroffenen Idealisierungen unterstützen lediglich die Erklärung der Zusammenhänge. Die FCUP-MAC Parameter sind in Tabelle 6.4 angegeben. Sie wurden zum Teil aus den technischen Parametern der Particle Computer Plattform abgeleitet. Simulator und Ergebnisse sind Teil der vom Autor dieser Arbeit

Stationen	4
Puffersollwert N_0	160
Kommunikationsperiode	10 Sekunden, synchronisiert
Kanalzugriff	faire Arbitrierung durch Wettbewerb
Dauer $C^{\text{comm.}}$	2 Sekunden
$C^{\text{entf.}} + C^{\text{lok.}}$	$1.5 + 0.5$ Sekunden
Bandbreite W [Nachr./s]	100
Kollaborationsregler K_p	$< \frac{1}{100}$
Kooperationsregler	$N_{\min} = 20, N_{\max} = 310$

Tabelle 6.4: Simulationsparameter

betreuten Studienarbeit von Emilian Peev [131]. Zur Erstellung der Simulation werden periodische Kommunikationsservices mit den Werkzeugen aus Abschnitt 4.6.3 spezifiziert. Der Simulationskern ist in Java programmiert. Er extrahiert die Parameter aus den Servicebeschreibungen und simuliert die Kommunikation im Netzwerk. Alle Stationen befinden sich in direkter Kommunikationsreichweite. Die Ergebnisse werden in einer Datenbank zur späteren Auswertung abgelegt.

6.8.1 Kollaborative FCUP-MAC Pufferregelung

Es wird das Pufferverhalten während der Kommunikationsdauer untersucht. Alle Stationen sind synchronisiert, und in regelmäßigen Abständen von 10 Sekunden beginnen die Prozesse mit dem Nachrichtenaustausch untereinander. Abbildung 6.11 (links) zeigt die Pufferfüllstandsänderung der Stationen. Es ist zuerst ein Anstieg und danach ein Abfall zu beobachten, der für die Stationen verschieden ist. Nach jeder Kommunikationsphase wird der Sollwert $N_0 = 160$ erreicht. Abbildung 6.11 (rechts) zeigt für den Zeitabschnitt 48 bis 50 Sekunden die Füll-

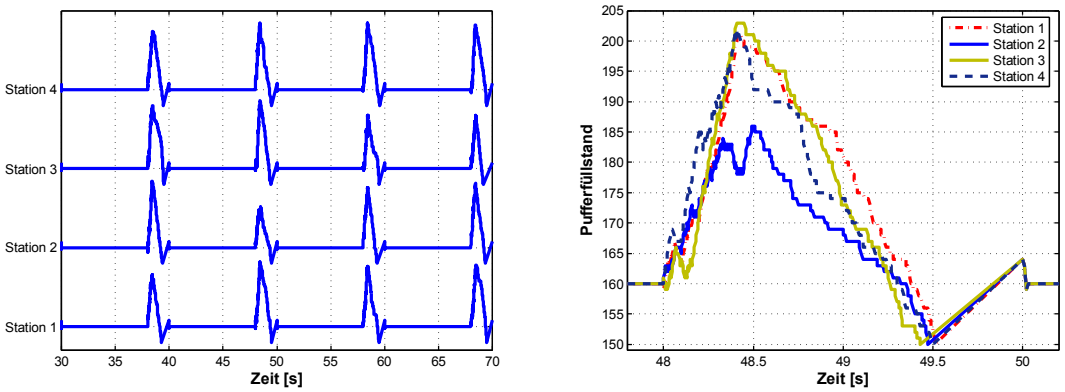


Abbildung 6.11: Links: Pufferfüllstände der vier Stationen über die Zeit. Rechts: Füllstände der Stationen im Zeitabschnitt $48 - 50$ Sekunden im Vergleich. Regelungsziel $N_0 = 160$ wird erreicht.

stände aller Stationen in einem Diagramm. Die Erläuterung der Regelungsabläufe ist in Tabelle 6.5 gegeben. Dass die Kommunikationsphasen in dieser Reihenfolge

Zeitausschnitt [s]	**Verhalten**
< 48	Pufferfüllstand auf Sollwert $N_0 = 160$
48 - ≈ 48.5	Stationen senden und empfangen. Regler berechnet $C_S^{\text{entf.}}(w)$ und $C_R^{\text{entf.}}(w)$ gemäß Gleichung 6.4. Die Arbitrierung ermittelt bei jedem Kanalzugriff den Sender, alle weiteren Stationen sind Empfänger. Es wird dreimal mehr empfangen als gesendet. Pufferfüllstand steigt und die Abweichung vom Sollwert ist maximal.
48.5 - ≈ 49.5	Zuvor berechnete Empfangsdauer $C_R^{\text{entf.}}(w) = 0$ ist erschöpft. Es wird nur noch gesendet. Pufferfüllstand geht zurück und erreicht Minimum.
49.5	Zeitdauer $C^{\text{entf.}}$ ausgeschöpft.
49.5 - 50.03	Mit festen Raten schreiben lokale Prozesse Nachrichten in den Puffer und entnehmen die für sie bestimmten Nachrichten. Dauer: $C^{\text{lok.}}$
50.03	Pufferfüllstand auf Sollwert $N_0 = 160$. Regelungsdauer: 1 Zeitschritt $= C^{\text{comm.}}$

Tabelle 6.5: Kollaborative Pufferregelung von FCUP-MAC im Zeitabschnitt 48 bis 50 Sekunden.

auftreten, ist der Synchronisation zum Zweck der übersichtlichen Erklärung geschuldet. Synchronisation und Reihung sind keine Voraussetzungen für die kollaborative FCUP-MAC Pufferregelung.

Der Nachrichtenaustausch der Stationen ist untereinander gekoppelt. Gemeinsam wird das Nachrichtenaufkommen so verarbeitet, dass alle Stationen den vorgegebenen Sollwert des Pufferfüllstandes erreichen. Die Stationen arbeiten kollaborativ im Verbund.

6.8.2 Kooperative Energieegalisierung

Die Stationen haben zum Startzeitpunkt der Simulation die (skalierten) Energievorräte $E(0) = (5000, 5300, 5500, 6000)$. Die Abbildung 6.12 zeigt die Änderung des Puffersollwertes über die Zeit. Je nach dem stationseigenen Energievorrat regelt der Energieregler zu den Zeitpunkten $t = 18, 58, 88$ Sekunden die Stationen nacheinander auf $N_{\min} = 20$. Der Pufferregler folgt dem neuen Sollwert. Die Station mit dem größten Energievorrat erhöht den Grad der Kollaboration, in dem der Energieregler den neuen Sollwert $N_{\max} = 310$ vorgibt. Die Pufferregler benötigt zwei Perioden, um $N_{\max}$ zu erreichen.

Die korrespondierenden Ergebnisse des Verlaufs der Energievorräte sind in Abbildung 6.13 angegeben. Die Energieregelung zu den Zeitpunkten $t = 18, 58, 88$ erfolgt an den Schnittpunkten der Verläufe der Energievorräte. Aufgrund der synchronisierten Perioden wird das Ziel mit einem sehr kleinen $\epsilon = 50$ erreicht. Alle Energievorräte sind für $t > 150$ Sekunden egalisiert. Die jeweiligen Energieregler geben dann die ursprünglichen Sollwerte N_0 wieder vor.

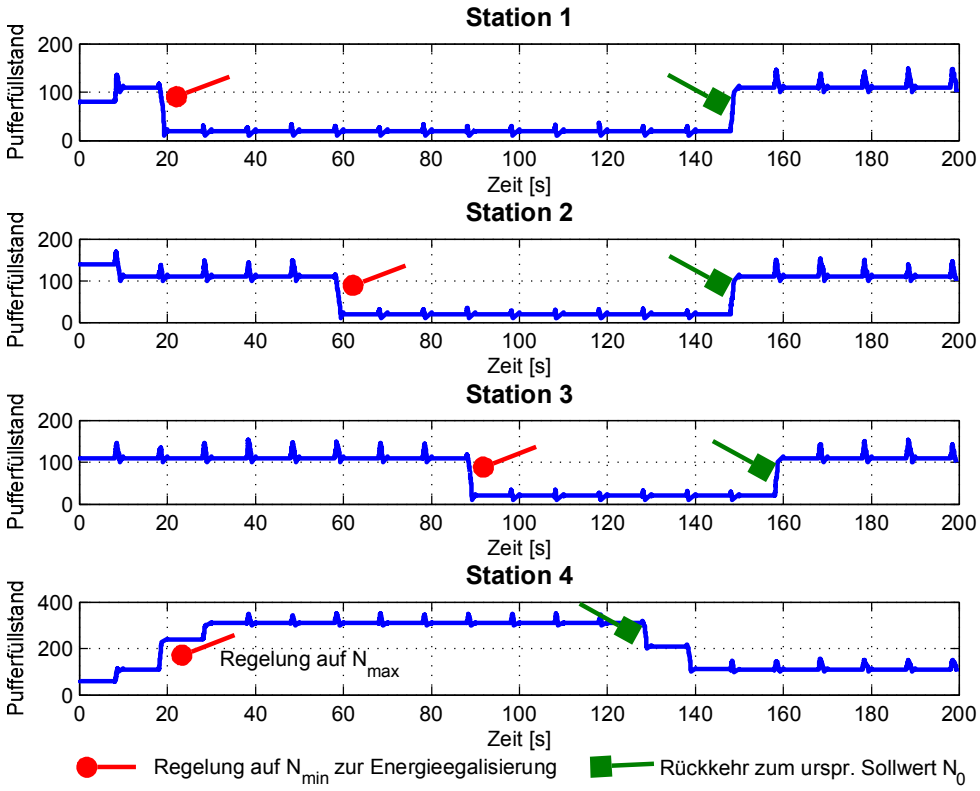


Abbildung 6.12: Regelung des Sollwertes des Pufferfüllstands bei kooperativer Energieegalisierung

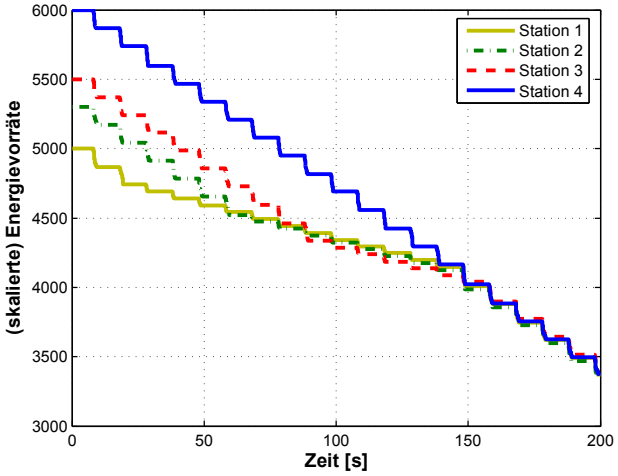


Abbildung 6.13: Zeitliche Entwicklung der Energievorräte bei kooperativer Energieegalisierung

6.9 Kollaborative Organisation in größeren Netzwerken

Anwendungen wie das Motivationsbeispiel CoBIs aus Abschnitt 6.1 oder ein verteiltes elektronische Siegel [7] bilden Netzwerke mit größerer Anzahl ubiquitärer Rechnersysteme. Alle Stationen befinden sich in direkter Kommunikationsreichweite. Die Kommunikationsperioden werden zufällig gewählt und sind *nicht synchronisiert*. Die Kollaboration verteilter Prozesse beim Nachrichtenaustausch ist daher besonders relevant.

Dieser Abschnitt untersucht mittels Simulation die Kollaboration in Netzwerken in Abhängigkeit der Anzahl teilnehmender Stationen. Tabelle 6.6 zählt die Simulationsparameter auf. Die kollaborative Organisation verteilter Prozesse mittels FCUP-

allgemeine Parameter	
Stationen	$10 \ldots 30$
Kommunikationsperioden	6-11 Sekunden, zufällig uniform verteilt
Kanalzugriff	faire Arbitrierung durch Wettbewerb
Simulationszeit	110 Sekunden
FCUP-MAC Parameter	
Puffersollwert N_0	10
Dauer $C^{\text{comm.}}$	1 Sekunde
$C^{\text{entf.}} + C^{\text{lok.}}$	$0.9 + 0.1$ Sekunden
Bandbreite W [Nachr./s]	100
Kollaborationsregler K_p	$< \frac{1}{100}$

Tabelle 6.6: Simulationsparameter für größere Netzwerke mit nicht synchronisierten und zufällig verteilten Kommunikationsperioden.

MAC wird dem ungeregelten Direktzugriff auf den Kommunikationskanal gegenübergestellt. Für die Bewertung werden die Metriken, durchschnittliche Latenzzeit und Anzahl erfolgreich ausgetauschter Nachrichten, verwendet. Bei der Erfassung der Latenzzeiten wird davon ausgegangen, dass jede neue Nachricht der jeweiligen lokalen Prozesse *jede* andere Station im Netzwerk erreichen muss. Es werden die Transportzeiten aller Nachrichten zu jeder Station ermittelt und anschließend zu einem Durchschnittswert verrechnet. Die Anzahl erfolgreich ausgetauschter Nachrichten wird für alle Stationen mitgezählt. Die Netzwerksimulationen und Berechnungen wurden mit dem Simulator aus der vom Autor dieser Arbeit betreuten Studienarbeit von Emilian Peev [131] erstellt. Abbildung 6.14 vergleicht die Kommunikationsleistung von FCUP-MAC und Kanaldirektzugriff für größer werdende Stationenanzahlen.

FCUP-MAC zeigt in beiden Fällen die bessere Leistung. Bei Kanaldirektzugriff vermindert sich zusätzlich mit größer werdender Stationenanzahl die Anzahl erfolgreich ausgetauschter Nachrichten um mehr als 30%. Es konkurrieren permanent mehr Stationen um den Kanal, was Nachrichten verzögert und den Nachrichtenverlust durch Kollision erhöht. Mit dem geregelten FCUP-MAC begrenzen die Stationen ihre Aktivität auf den zu erreichenden Pufferfüllstand. In der folgenden Tabelle 6.7 wird der Leistungsvergleich quantitativ zusammengefasst. Es werden mehr Nachrichten in kürzerer Zeit ausgetauscht. Die Zeit von der Nachrichtenemission bis zur Verarbeitung wird im selben Maße wie die Latenzzeit verkleinert.

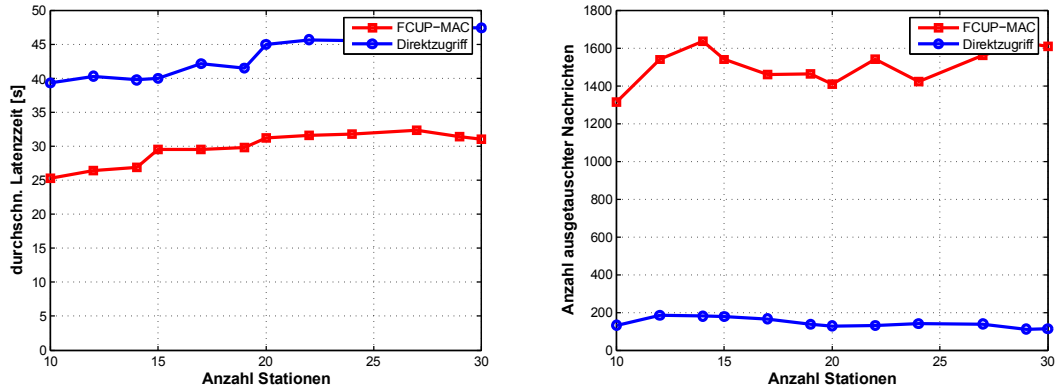


Abbildung 6.14: Leistungsvergleich des kollaborativ organisierten FCUP-MAC mit Kanaldirektzugriff bei steigender Netzwerkgröße. Links: Durchschnittliche Latenzzeiten. Rechts: Durchschnittliche Anzahl erfolgreich ausgetauschter Nachrichten.

durchschn. Latenzzeit	um 27% bis 32% kleiner
erfolgr. ausgetauschte Nachrichten	Faktor 8 bis 10 mehr Nachrichten

Tabelle 6.7: Verbesserungen durch kollaborative Organisation verteilter Prozesse mittels FCUP-MAC.

Diese Ergebnisse erlauben die folgende Schlussfolgerung: Die kollaborative Organisation mittels FCUP-MAC verbessert signifikant den Nachrichtenaustausch und erreicht eine zeitnahe Datenverarbeitung auf ubiquitären Rechnersystemen in Einsatzszenarien mit unbekannten Kommunikations- und Netzwerkbedingungen.

6.10 Weitere Ergebnisse

Die Kollaboration mittels FCUP-MAC bewirkt eine flutartige Weiterleitung der Nachrichten. Das aggregierte Nachrichtenaufkommen erzeugt Latenzen im Kommunikationsverbund und einen direkt mit dem Aufkommen zusammenhängenden Energieaufwand.

Daher ist die Anzahl der Weiterleitungsstationen, engl. hop count, durch die sogenannte TTL, engl. time to live, zu begrenzen. Die TTL ist ein numerischer Wert, der einer Nachricht beigefügt wird und bei Weiterleitung von der jeweiligen Station dekrementiert wird. Bei TTL = 0 wird die Nachricht nicht weitergeleitet; sie wird verworfen. Wird die TTL hoch angesetzt, so können die Nachrichten längere Wege über mehrere Stationen zurücklegen. Selten aktive Stationen können erreicht werden, da Nachrichten länger im Umlauf sind. Das größere Nachrichtenaufkommen erzeugt jedoch Latenzen und einen höheren Energieaufwand. Ist die TTL klein, dann ist die Nachricht nur kurz im Umlauf und einige Zielstationen werden nicht erreicht. Vorteilhaft ist aber der geringere Energieaufwand.

Für das kollaborativ organisierte Kommunikationsverhalten von FCUP-MAC wurde in zahlreichen Simulationen ein empirischer Zusammenhang zwischen der Anzahl der Stationen n eines Netzwerkes und einer geeigneten TTL gefunden. Dazu wurden die vier Fallstudien aus Tabelle 6.8 zur Bestimmung der TTL durchgeführt. Die Bewertung erfolgte durch Ermitteln der Latenzen und des Verhältnisses von

Fallstudie	**TTL**
Radix	TTL $= \sqrt{n}$
Linear	TTL $= n$
Feststehend, klein	TTL $= 6$
Feststehend, groß	TTL $= 15$

Tabelle 6.8: Fallstudien zur Bestimmung der TTL. n bezeichnet die Stationenanzahl.

Energieaufwand und Latenz. Große Latenzen implizieren Nachrichtenverluste bevor alle Zielstationen erreicht wurden. Ein erfolgreicher Nachrichtenaustausch ist seltener. Für die Energie-Latenz Metrik wird die Latenz im Verhältnis zur Simulationszeit normiert. Ist die Latenz groß im Verhältnis zur Simulationszeit, dann ist der Energieaufwand für einen (seltenen) erfolgreichen Nachrichtenaustausch entsprechend groß. Die Abbildung 6.15 zeigt die Ergebnisse der Fallstudien. Keine der

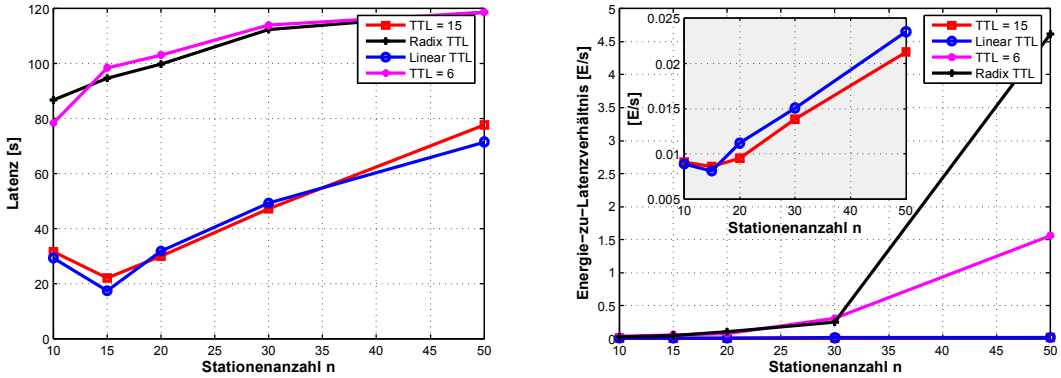


Abbildung 6.15: Ergebnisse der TTL Fallstudien. Links: Latenzzeiten der Studien. Rechts: Energie-zu-Latenzverhältnis. Die Vergrößerung zeigt das Verhalten der Linear TTL $= n$ und TTL $= 15$ Studien.

Fallstudien ist als generelle Lösung zur Bestimmung der TTL geeignet. Die Radix Studie und die Studie mit TTL $= 6$ haben sowohl eine hohe Latenz wie auch einen hohen Energieaufwand, um Nachrichten erfolgreich auszutauschen. Für die beiden anderen Strategien zeigt sich für $n = 15$ eine besonders geringe Latenz und Energieaufwand. Hier ist das beste Verhältnis aus Nachrichtenumlauf, Energieaufwand und verfügbaren Kommunikationsressourcen hergestellt. Eine Kombination aus der linearen Strategie und der Strategie mit fester TTL liefert die besten Resultate über alle Stationenanzahlen. Das Energie-zu-Latenzverhältniss für größere n ist bei genügend großer, aber feststehender TTL, besser als beim linearen Verlauf. Die sich ergebene Strategie zur Bestimmung der TTL ist in Abbildung 6.16 gezeigt. Die TTL steigt mit der Netzwerkgröße im gleichen Maß. Aber ab einer bestimmten Netzwerkgröße, Stelle $n = \text{TTL}^{\text{sat.}} = 15$ in Abbildung 6.16, ist die Kommunikationshäufigkeit der Stationen so dicht, dass eine weitere Vergrößerung der TTL nur die Umlaufzeiten erhöht, aber der Beitrag zum Nachrichtenaustausch gegenüber dem Energieaufwand zurückfällt.

Dieses Verhalten wird verständlich, wenn man sich die Kommunikationsaktivitäten zufällig verteilt auf die Durchschnittsperiode $\overline{T^{\text{comm.}}} = \frac{1}{n}\sum_{i=1}^{n} T_i^{\text{comm.}}$ vorstellt. Die Periodendauer ist weiterhin in Abschnitte der Größe der Kommunikationsdauer $C^{\text{comm.}}$ der Stationen unterteilt. Die TTL entspricht der Anzahl der Schritte, die

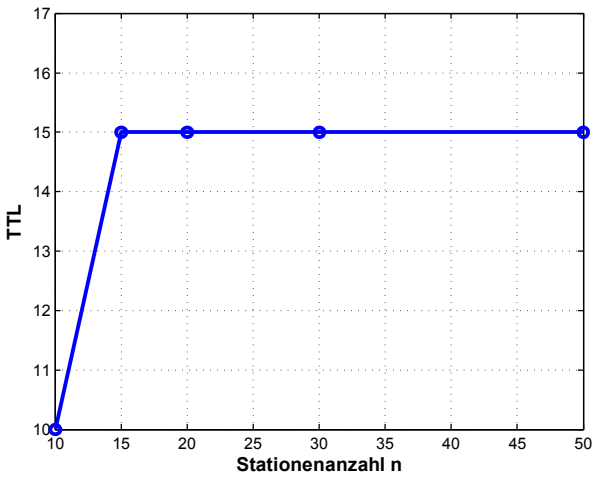


Abbildung 6.16: Strategie zur Bestimmung der TTL als Kombination der Strategien der linearen TTL Studie und der mit TTL = 15. Der Übergang zwischen beiden Strategien erfolgt bei $n = 15$.

notwendig sind, um im Verlauf der Durchschnittsperiode alle Stationen zu erreichen. Entspricht die Stationenanzahl der Anzahl der $C^{\text{comm.}}$-Zeitabschnitte in $\overline{T^{\text{comm.}}}$, dann sind im Mittel über die gesamte Periode hinweg alle Zeitabschnitte von *nacheinander*, d.h. weiterleitend, kommunizierenden Stationen belegt. Für die Kommunikation von Abschnitt zu Abschnitt wird die TTL jeweils dekrementiert. Eine weitere Erhöhung der Stationenanzahl lässt mehr Stationen *gleichzeitig* kommunizieren. Dabei bleibt die TTL konstant. Die TTL ist gesättigt.

Die Sättigungsstelle $\text{TTL}^{\text{sat.}}$ der TTL Kurve in Abbildung 6.16 kann mit diesem Ansatz wie folgt approximiert werden.

$$\text{TTL}^{\text{sat.}} = 2\frac{\overline{T^{\text{comm.}}}}{C^{\text{comm.}}} \tag{6.11}$$

Die Multiplikation mit dem Faktor 2 stellt sicher, dass sich die Kommunikationsintervalle überlappen und so ein erfolgreicher Nachrichtenaustausch ermöglicht wird. Mit uniformverteilten Perioden in den Grenzen $[T^{\text{l}}; T^{\text{h}}]$ ist $\overline{T^{\text{comm.}}} = 0.5\left(T^{\text{h}} - T^{\text{l}}\right)$. Mit Gleichung 6.11 lässt sich nun die TTL Strategie wie folgt formulieren.

$$\text{TTL} = \begin{cases} n, & n < \text{TTL}^{\text{sat.}} \\ \text{TTL}^{\text{sat.}}, & \text{sonst.} \end{cases} \tag{6.12}$$

Für die Fallstudien wurden uniformverteilte Perioden in den Grenzen $[6; 11]$ Sekunden und $C^{\text{comm.}} = 1$ Sekunde verwendet. Mit Gleichung 6.11 berechnet sich $\text{TTL}^{\text{sat.}} = 16$. Die Kombination der linearen mit der festen Strategie um diesen Punkt bestätigt die Simulationsergebnisse in Abbildung 6.15. Für die Implementierung müssen Stationenanzahl n und die Bereichsgrenzen $[T^{\text{l}}; T^{\text{h}}]$ ermittelt werden respektive bekannt sein.

6.11 Praktische Aspekte

Es werden praktische Aspekte der Gewinnung von Modellparametern der Regelung diskutiert. In unbekannten Netzwerkumgebungen müssen die Regelungsparameter

von den ubiquitären Rechnersystemen zur Laufzeit ermittelt werden können. Ein weiterer Aspekt ist die Verwendung von FCUP-MAC in Multi-Hop Netzwerken.

6.11.1 Bestimmung der Modellparameter

Der zentrale Parameter ist die Anzahl der Netzwerkstationen n. Mit n lassen sich die Bandbreite W, die TTL und der FCUP-MAC Regler bestimmen.

Anzahl der Netzwerkstationen n. Die Anzahl kann über das Sammeln von Identifikationsinformationen, zum Beispiel eine Geräteidentifikation, durchgeführt werden. Speicher- und vor allem Zeitaufwand dieser Maßnahme sind ungeeignet für den Einsatz in größeren Netzwerken. Statistische Schätzverfahren können den Vorgang beschleunigen. In [132] konnten Rechnergruppen mittels überlagerten Funksignalen auf extrem ressourcenbeschränkten Rechnersystemen gezählt werden. Jedoch wird dabei Synchronisation vorausgesetzt. Eine andere Methode verwendet Bitmasken, die im Nachrichtenpaket mitgeführt werden und von den jeweiligen Stationen bei Weiterleitung durch ein Zufallsexperiment verändert werden. Die Bitmaske bildet einen Hash über die Anzahl durchlaufener Stationen. Nath et al. [133] schlägt für Sensornetzwerke eine Modifikation des Flajolet-Martin Algorithmus zum probabilistischen Zählen von Elementen in Datenbanken vor. Computersimulationen, die im Rahmen dieser Arbeit durchgeführt wurden, zeigten, dass dieses Schätzverfahren effizient realisiert werden kann. Die Schätzungen sind schwankend. Eine Durchschnittsbildung liefert zufriedenstellende, stabile Resultate. Stark abweichende Schätzungen können mit einem gleitenden Mittel kompensiert oder durch eine Schwellwertfilterung behandelt werden. Der Berechnungsaufwand ist konstant. Vom Entwickler ist für das Sammeln der Hashwerte eine Anzahl von Kommunikationsperioden vorzugeben, bis die Schätzung erfolgt.

TTL. Sie beeinflusst Nachrichtenaufkommen und Energieaufwand der Kommunikation. Die Bestimmung der TTL im kollaborativen Kommunikationsverbund der verteilten Prozesse erfolgt gemäß der Strategie in Gleichung 6.12. Es wird die Kenntnis über die Bereichsgrenzen der Kommunikationsperioden bei uniformer Verteilung vorausgesetzt. Die Sättigungsgrenze aus Gleichung 6.11 kann vorberechnet und als Konstante abgelegt werden. Die Bestimmung der TTL gemäß Gleichung 6.12 vergleicht die Sättigungsgrenze mit der Stationenanzahl n. Sie muss mit Änderung von n wiederholt werden. Die Bestimmung der TTL ist mit konstantem Rechenaufwand auf ubiquitären Rechnersystemen durchführbar.

Bandbreite W. Sie wird vom Entwickler zur Entwurfszeit des Systems vorgegeben. Eine Neubewertung von W wird notwendig, wenn die Stationenanzahl sich stark gegenüber der Annahme zur Entwurfszeit verändert hat. Die Berechnung erfolgt gemäß der Vorschrift aus Abschnitt 6.5.2. Es wird die Kenntnis über die Bereichsgrenzen der Kommunikationsperioden bei uniformer Verteilung vorausgesetzt. Mit der Anzahl der Netzwerkstationen n lässt sich W berechnen. Der Rechenaufwand rechtfertigt eine Tabelle, die für verschiedene Bereichsgrenzen und Stationenanzahlen vorberechnete Werte enthält. Die Tabelle kann beispielsweise in einem externen Festspeicher des ubiquitären Rechnersystems abgelegt werden. Tabellenzugriffzeiten beeinflussen die Leistung kaum, da eine Neubewertung bei den eher selten auftretenden starken Änderungen der Stationenanzahl vorgenommen werden muss. Die Bandbreitenbestimmung kann mit konstantem Aufwand für ubiquitäre Rechnersysteme durchgeführt werden.

Reglersynthese. Änderungen der Bandbreite erfordern eine neue Reglersynthese. Der Regler K_p wird gemäß Gleichung 6.8 durch eine Division der Bandbreite und einem Vergleich berechnet. Die Bandbreite W ist immer nur als statistisches Mittel zu betrachten. Schwankungen können mit einer größeren Reglerverstärkung kompensiert werden. Gegebenenfalls muss hierfür die Sättigungsgrenze g erhöht werden. Dies geschieht durch Vergrößerung von $C^{\text{entf.}}$. Das System erhält so mehr Kommunikationsressourcen. In beiden Fällen können vorberechnete Tabellen verwendet werden. Die online Reglersynthese kann mit konstantem Aufwand auf ubiquitären Rechnersystemen durchgeführt werden.

6.11.2 Multi-Hop Netzwerke

Stationen, die sich außerhalb der Kommunikationsreichweite des Senders befinden, können nicht direkt erreicht werden. Zur Überbrückung der Entfernung wird die Nachricht über mehrere Zwischenstationen zum Ziel befördert. Dies bezeichnet man als Multi-Hop Netzwerk. Durch die Weiterleitung von Nachrichten realisiert FCUP-MAC bereits Multi-Hop Funktionalität. Multi-Hop Netzwerke wurden jedoch in den Simulationen nicht berücksichtigt. Es lassen sich jedoch aus den bisherigen Ergebnissen Maßnahmen ableiten.

Ein Muli-Hop Netzwerk besteht aus zwei oder mehreren Single-Hop Netzwerkzellen. Die TTL muss vergrößert werden, um neben Weiterleitung auch die Zellen zu überbrücken. Eine Verdopplung ist für Szenarien mit Quelle und Ziel in zwei aneinandergrenzenden Zellen geeignet. Noch größere Netzwerke erfordern entsprechende Multiplikationsfaktoren. Für die Bandbreitenbestimmung und Reglersynthese ist die Betrachtung einer Zelle ausreichend. Offen jedoch ist das Problem wie die Zellenanzahl respektive Zugehörigkeiten von Stationen zu Zellen effizient ermittelt werden kann. Möglich wäre eine Vorgabe eines Administrators durch spezielle Netzwerkkontrollpakete.

Gegenüber den Simulationsergebnissen aus Abschnitt 6.9 wird sich die Latenzzeit vergrößern. Stationen mit zuvor unmittelbarem Nachrichtenaustausch können sich außerhalb der Kommunikationsreichweite befinden. Eine vergrößerte TTL vergrößert zusätzlich die Latenzzeit, da Nachrichten länger im Umlauf sind und mehr Kommunikationsressourcen erfordern. Im Multi-Hop Betrieb von FCUP-MAC sollte die Kommunikationszeit $C^{\text{entf.}}$ drastisch erhöht werden, um das Regelungssystem zu beschleunigen.

6.12 Was wurde erreicht?

Dieses Kapitel zeigte, wie das Konzept von Kooperation und Kollaboration durch rückgekoppelte Regelkreise aus Kapitel 3 erfolgreich für die verteilte Prozessorganisation ubiquitärer Rechnersysteme umgesetzt werden kann.

Zeitnahe Verarbeitung von Nachrichten steht im Vordergrund. Netzwerkbedingungen und Kommunikationsbeziehungen sind unbekannt und zeitlich veränderlich. Ziel ist es, unter diesen Umständen Nachrichtenaustausch und -verarbeitung mit mehreren ubiquitären Rechnersystemen effektiv zu gestalten. Lösungsansatz ist die kollaborative und kooperative Prozessorganisation in einem *Verbund mit regulierendem Nachrichtenaustausch.*

Mit FCUP-MAC wurde in Abschnitt 6.4.2 ein neues, geregeltes, hybrides Kanalzugriffsverfahren für ubiquitäre Rechnersysteme vorgestellt. Mit der Einbettung in einen Regelkreis in Abschnitt 6.5 ist es gelungen, Zusammenhänge zwischen (1) Bandbreite und Sollwert eines Pufferfüllstandes und (2) Bandbreitendynamik und Kommunikationsdauer herzustellen. Zur Entwurfszeit noch unbekannte Kommunikationsbeziehungen werden somit flexibel berücksichtigt. Kollaboration mittels FCUP-MAC lässt verteilte Prozesse in einem Verbund mit gemeinschaftlich reguliertem Nachrichtenaustausch kommunizieren. Prozesskooperation egalisiert die Energievorräte und maximiert somit die Kollaborationsdauer. FCUP-MAC organisiert kollaborativ und kooperativ verteilte Prozesse auf ubiquitären Rechnersystemen in Einsatzszenarien mit unbekannten Kommunikations- und Netzwerkbedingungen.

Mehrere Simulationsstudien in den Abschnitten 6.8 und 6.9 belegen die theoretischen Vorhersagen des Budget/Kosten Modells. Kollaboration beim Nachrichtenaustausch mittels FCUP-MAC ermöglicht eine Prozesskoordination ohne Synchronisation. Gegenüber dem direkten, unkoordinierten Zugriff verbessert FCUP-MAC Latenz und Durchsatz um mehrere Faktoren. Eine zeitnahe Nachrichtenverarbeitung wird erreicht. Kollaboration und Kooperation mittels FCUP-MAC lösen die Problemstellung aus Abschnitt 6.3.

Abschnitt 6.11 zeigt, dass die Regelung sich effizient auf ubiquitären Rechnersystemen implementieren lässt. Große Veränderungen in der Netzwerkkonfiguration lassen sich durch eine Reglersynthese auf dem Rechnersystem behandeln. Verteilte Prozesse sind mit den in Kapitel 4 eingeführten Instrumenten der serviceorientierten Modellierung darstellbar. FCUP-MAC ist Teil des periodischen Kommunikationsservice. Schichtenmodell und Integration in das Betriebssystem Particle OS wurden in Abschnitt 6.4 vorgestellt.

Anwendungen wie CoBIs [6] oder verteilte elektronische Siegelsysteme [7] profitieren von der kollaborativen und kooperativen Organisation. Unter *Beibehaltung des zeitlichen Ausführungsverhaltens* jedes einzelnen Rechnersystems verarbeiten verteilte Prozesse Nachrichten zeitnah für eine gemeinschaftliche Detektion von Ereignissen. *Deterministische Prozessabläufe* von hochdiversen Aufgaben werden in Einsatzszenarien mit unbekanntem Kommunikationsverhalten ermöglicht.

7. Energiemanagement

Die batteriebasierte Energieversorgung bildet die zentrale lokale Ressource, die die Ausführung aller Prozesse eines ubiquitären Rechnersystems ermöglicht. Energie steht nur beschränkt zur Verfügung und wird bei Prozessausführung verbraucht. In unbekannten Einsatzumgebungen ist jedoch der Verbrauch schwer vorhersehbar. Ursachen sind das nichtlineare Entladeverhalten von Batterien und das komplexe Zusammenspiel von Prozessaufgaben und Ansteuerung peripherer Komponenten wie Sensoren, Aktuatoren, Speicher und Kommunikation. Ziel dieses Kapitels ist es, den Energieverbrauch so zu organisieren, dass ein ubiquitäres Rechnersystem eine vorgegebene Betriebsdauer unter unbekannten Einsatzbedingungen erreicht.

Für das zielgerichtete Energiemanagement wird die Theorie der kooperativen und kollaborativen Prozessorganisation ubiquitärer Systeme aus Kapitel 3 angewendet. Kooperation organisiert Prozesse in Betriebsphasen, in denen sie abschnittsweise aktiviert und deaktiviert werden. In einem Regelkreis werden Informationen über den aktuellen Energievorrat zurückgeführt, um die Phasen auf das Erreichen der Betriebsdauer abzustimmen. Bisherige Lösungsansätze verwenden zusätzliche Messhardware oder Benchmarks zur Charakterisierung des Energieverbrauchs. Diese Profiling-Methoden sind zeit- und speicheraufwendig und sehr stark an Prozessaufgaben und Ausführungsplattform mit ihren peripheren Komponenten gebunden. Kooperation abstrahiert von den Prozessaufgaben und der Rechnerplattform. Die vorgegebene Betriebsdauer wird *bestmöglich* ohne Verwendung von Profiling-Informationen erreicht.

Zu Beginn charakterisiert das Kapitel die Energienutzung ubiquitärer Rechnersysteme und untersucht Energiemanagementmechanismen aus dem wissenschaftlichen Umfeld. Der Entwurf entwickelt und erklärt die Architektur der abschnittsweisen Ablauforganisation. Die Einbettung in das Budget/Kosten Regelkreismodell aus Abschnitt 3.4 formuliert das kooperative Energiemanagement. Kern des Regelkreises ist PControl - ein optimaler Regler für den Energieverbrauch. Mit Modellanalysen und Simulationen wird das Verhalten in Einsatzszenarien mit schwankendem, nicht vorhersehbarem Energieverbrauch untersucht. Experimentelle Studien, die auf der Particle Computer Plattform durchgeführt werden, dienen der Bewertung des kooperativen Energiemanagements im tatsächlichen Einsatz. Schließlich wird gezeigt wie

Kooperations- und Kollaborationsmechanismen der Prozessklassen aus den vorangegangenen Kapiteln mit dem kooperativen Energiemanagement zusammenwirken.

7.1 Motivierendes Beispiel

In der AwareOffice Installation [134] sind mehr als 20 Gegenstände mit Particle Computer Rechnersystemen zur sensorischen Erfassung und Steuerung der Umgebung ausgestattet. Die Betriebsdauer der batteriebetriebenen Geräte beträgt etwa zwei Monate. Im Durchschnitt müssen somit die Batterien alle $\frac{60}{20} = 3$ Tage ausgetauscht werden. Zusätzlicher Aufwand entsteht durch das Ausbetten der Rechnersysteme - also der mechanische Aufwand, die Batterien zu wechseln.

Zielorientiertes Energiemanagement eingebetteter, ubiquitärer Rechnersysteme organisiert den Energieverbrauch so, dass der Batteriewechsel zu einer *vorgegebenen Zeit einplanbar* wird. Unterschiedliche Batteriekapazitäten wie auch Entladeverhalten durch unterschiedliche Prozessverhalten und -aufgaben müssen entsprechend berücksichtigt werden. Abbildung 7.1 zeigt, dass so für jedes beteiligte Rechnersystem eine spezifische Betriebsdauer gefordert wird, um das gemeinsame Ziel zu erreichen. Ein weiteres Beispiel für ein zielorientiertes Energiemanagement ist die

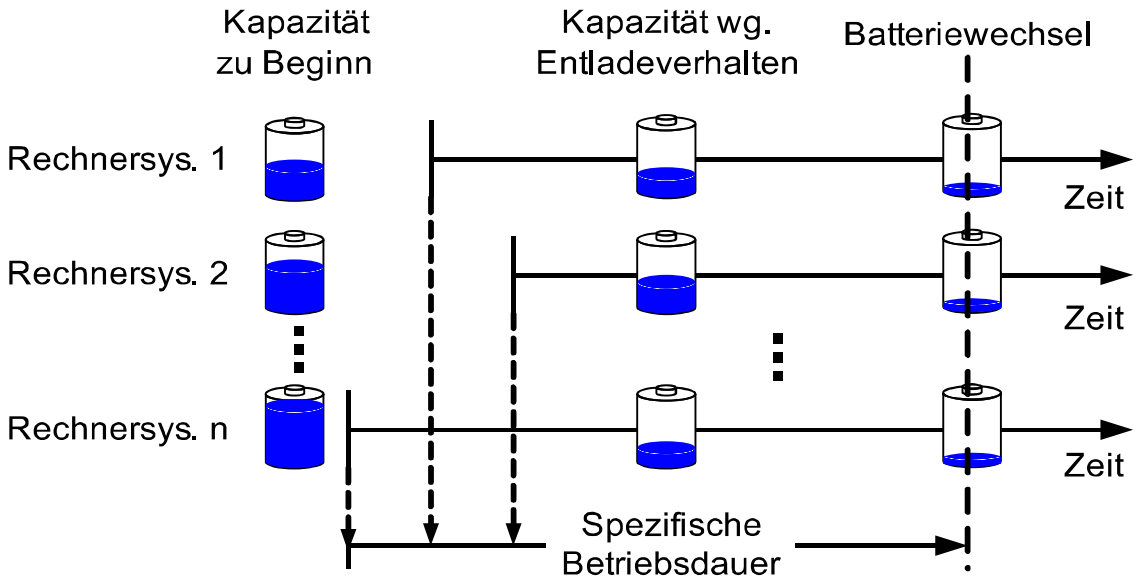


Abbildung 7.1: Zielorientiertes Energiemanagement berücksichtigt unterschiedliche Batteriekapazitäten wie auch Entladeverhalten, um eine spezifische Betriebsdauer bis zum gemeinsamen Batteriewechsel zu erreichen.

Sicherstellung des Betriebes von ubiquitären Rechnersystemen mit Solarzellen über Nachtzeiten hinweg. Es muss eine Betriebsdauer bis zum nächsten erwarteten Sonnenlichteinfall erreicht werden.

7.2 Analyse und Einordnung ins wissenschaftliche Umfeld

Der Bereich Energiemanagement umfasst Rechnersysteme, zum Beispiel in Anlagen wie Rechenzentren, wie auch einzelne miniaturisierte eingebettete Systeme. Es

erfolgt daher zuerst eine Eingrenzung des Themas in Bezug auf eingebettete, ubiquitäre Rechnersysteme. Energiecharakterisierung und Verfahren zum Energiemanagement werden im Anschluss diskutiert. Im Fazit wird schließlich ein neues kooperatives Verfahren für das Energiemanagement ubiquitärer Rechnersysteme skizziert.

7.2.1 Abgrenzung

Es wird zwischen Energiemanagement für Rechneranlagen und batteriebetriebenen Geräten unterschieden. In Rechneranlagen, zum Beispiel großen Rechenzentren, wird das Ziel verfolgt, die Integrationsdichte, d.h. mehr Rechner pro Raumeinheit, zu erhöhen. Kritisch ist dabei die Ableitung der entstehenden Wärme. Hotspots, d.h. räumlich begrenzte Regionen, die durch hohe Rechenleistung viel Wärme produzieren, begrenzen die Integrationsdichte und erfordern einen signifikanten Kühlaufwand. Letzteres repräsentiert fast 50% des Gesamtenergieaufwands für ein Rechenzentrum [135]. Das Energiemanagement soll Hotspots verhindern. Instrumente wie Umverteilung von rechenintensiven (= wärmeproduzierenden) Tasks, Regulierung der Rechnergeschwindigkeit bis zum Abschalten von Subsystemen wie Festspeicherlaufwerke, verteilen die Leistungsaufnahme über einen größeren Rauminhalt. Beispiele von Temperatur- und Energiemanagement werden in [136] diskutiert. Beim Einsatz der Instrumente muss zwischen Leistungsvermögen, zum Beispiel Durchsatz, Verzögerung und Antwortzeitverhalten, und Energieverbrauch abgewogen werden.

In batteriebetriebenen Geräten, insbesondere in mobilen und ubiquitären Rechnersystemen, soll eine hohe Betriebsdauer mit der zur Verfügung stehenden Batteriekapazität erzielt werden. Das Energiemanagement hat die Aufgabe die Batteriekapazität effizient zu nutzen. Dies wird umso dringlicher, da die Kapazitäten der Batterietechnologien langsamer steigen als der Strombedarf der Prozessoren. Dieser divergierende Zusammenhang wird als Batterielücke, engl. battery gap, bezeichnet. Durch effizientere Nutzung wirkt das Energiemanagement der Batterielücke entgegen. Häufig eingesetzte Instrumente sind Regulierung der Betriebsspannung, engl. voltage scaling, der Ausführungsgeschwindigkeit, engl. CPU throttling, und Abschalten des gesamten Rechnersystems über bestimmte Betriebsphasen, engl. duty cycling.

	Energiemanagement für Rechneranlagen	**Energiemanagement batteriebetriebener Geräte**
Ziel	Integrationsdichte erhöhen, Kühlaufwand senken	hohe Betriebsdauer mit begrenzter Batteriekapazität
Anzahl beteiligter Rechnersysteme	bis zu mehreren 10000	Einzelsysteme, kleine Verbünde
Leistungsaufnahme	bis mehrere 100 Watt pro Rechnersystem	Milliwatt bis zweistelligen Watt Bereich
Beitrag des Energiemanagements	Hotspots vermeiden	effiziente Nutzung der Batteriekapazität
Instrumente	Taskumverteilung, Abschalten von Subsystemen, Regulierung der Ausführungsgeschwindigkeit	Regulierung der Betriebsspannung und Ausführungsgeschwindigkeit, Abschalten des Rechnersystems

Tabelle 7.1: Unterscheidung des Energiemanagements nach Rechnerklasse

Die tabellarische Übersicht 7.1 grenzt die beiden Bereiche voneinander ab. Dieses Kapitel behandelt das Energiemanagement batteriebetriebener Geräte mit besonderem Fokus auf die Anforderungen eingebetteter, ubiquitärer Rechnersysteme.

7.2.2 Energiecharakterisierung ubiquitärer Rechnersysteme

Der Aufbau ubiquitärer Rechnersysteme in Abschnitt 2.2.3 identifiziert die drei Funktionsgruppen Mikrocontroller (MCU), Kommunikationssystem und Sensorik und Aktuatorik, die mit der Energieversorgung verbunden sind. Typische Sensoren sind Licht-, Temperatur-, Beschleunigungs-, Audio- und Kraftsensor [25]. Tabelle 7.2 zeigt die Leistungsaufnahme der Funktionsgruppen eines ubiquitären Rechnersystems - der Particle Computer Plattform. Die Leistungsaufnahme einer typischen

Komponente	**typische Leistungsaufnahme [mW]**
Mikrocontroller und Speicher	
PIC 18F6720	6.5
ext. Flashspeicher	15-50
Kommunikation	15.8
RFM TR1001 Emfpang	3.8
RFM TR1001 Senden	12
Sensorik	15.816
Lichtsensor TSL 25x/ 26x	2.5
Lichtsensor TSL2550 (I2C)	1.1
Beschleunigungssensor ADXL210	3.2
Temperatursensor TC74 (I2C)	1.0
Kapazitives Mikrofon + Verstärker	8.0
Dehnungsstreifensensor FSR-151AS	16 [μW]

Tabelle 7.2: Energiecharakterisierung ubiquitärer Rechnersysteme am Beispiel der Particle Computer Plattform

Sensorik gleicht der Leistungsaufnahme der drahtlosen Kommunikation. Sensoren werden in unterschiedlichen Intervallen oder ereignisgesteuert genutzt. Das Verhalten wird von den Prozessaufgaben bestimmt. In der Energiecharakterisierung sind sie relevant, aber die Nutzung ist schlecht planbar. Komponenten wie das Kommunikationssystem oder der externe Flashspeicher haben für sich betrachtet schon eine besonders hohe Leistungsaufnahme. Sie verursachen Leistungsspitzen von zusätzlich 15-50 mW, die von der Batterie aufgebracht werden müssen.

Unter diesen Umständen ist es schwierig, ein Energieprofil eines ubiquitären Rechnersystems zu erstellen. Die Schwankungsbreite kann sehr hoch ausfallen und zusätzliches Wissen über Prozessaufgaben soll nicht verwendet werden. Die Erstellung zur Laufzeit erfordert zusätzlichen Speicher- und Rechenaufwand. Eine Hardwareabstraktion wird nicht immer benutzt, da zum Beispiel erforderliche, spezielle Peripheriezugriffe ungenügend unterstützt werden. Prozesse implementieren diese Zugriffe an der Schnittstelle vorbei als Teil der Prozessaufgaben. Die Nutzung der Peripherie kann nur unzureichend erfasst werden. Eine Priorisierung der Peripherienutzung hinsichtlich des Energieverbrauchs ist somit nicht anwendbar.

Moderne Mikrocontroller unterstützen verschiedene Leistungsstufen. Zum Beispiel implementiert der PIC18F6722 Mechanismen, um verschiedene Taktraten einzustellen, mehrere Leerlaufmodi und einen besonders energiearmen Ruhemodus [137]. Schlüsselkomponenten wie Zeitgeber und Watchdog sind für eine besonders niedrige Leistungsaufnahme ausgelegt. Jedoch implementieren unterschiedliche Hersteller verschiedene Stufen mit verschiedenen Semantiken. Ein einheitlicher Standard ist noch nicht etabliert.

Die Messung des Energieverbrauchs ubiquitärer Rechnersysteme wird von den verwendeten Mikrocontrollern nicht unterstützt. Ansätze wie Energy Container [136] sind durch fehlende Ereigniszähler auf Hardwareebene schwierig zu realisieren. Eine breite Instrumentierung der Software für diverse Sensorik- und Kommunikationskomponenten erzeugt zusätzlichen Aufwand hinsichtlich Laufzeit, Programmcode und Speicher. Abwägungen zwischen diesem Aufwand und Genauigkeit der Erfassung sind zu bestimmen. In [138] wird eine ähnliche Problemstellung für Energy Container in TinyOS diskutiert. Jedoch bieten sich für ubiquitäre Rechnersysteme in unbekannten Einsatzszenarien und bei Abstraktion von den Prozessaufgaben keine geeigneten Anhaltspunkte für solche Abwägungen an. Separate Hardwarelösungen, die zusätzlich in ubiquitäre Rechnersysteme integriert werden, erfassen den Energieverbrauch direkt durch Messung der Batterie. In beiden Fällen ist die Erfassung sehr stark an die Energiequelle gebunden und muss gegebenenfalls an den verwendeten Batterietyp angepasst werden.

Beim Einsatz der Rechnersysteme in unbekannten Umgebungen sind Sensorsteuerung und Datenverarbeitungsaufwand im Voraus schwer planbar. Das macht den Energieverbrauch schwer vorhersehbar. Ein Energiemanagement muss eine minimale Ansprechbarkeit respektive minimale Systemleistung garantieren. Beispielsweise soll das System, wenn auch seltener, regelmäßig kommunizieren oder Sensordaten verarbeiten können. Diese untere Grenze soll vom Entwickler bestimmt werden.

Zusammenfassend, ist die Energienutzung ubiquitärer Rechnersysteme wie folgt zu charakterisieren:

- Leistungsaufnahme der Sensorik ist relevant, aber schlecht planbar.
- Einzelne Komponenten für Kommunikation oder Speicher erzeugen Leistungsspitzen.
- Erstellung eines Energieprofils ist nicht praktikabel.
- Hardwareunterstützte Energiemodi sind noch nicht standardisiert.
- Erfassung des Energieverbrauchs ist eng an die Energiequelle (Batterie) gebunden.
- Einsatz in unbekannten Szenarien erfordert eine minimale Ansprechbarkeit des Rechnersystems.

In den folgenden Abschnitten werden Energiemanagementverfahren für batteriebetriebene, eingebettete Sensor- und Rechnersysteme kategorisiert und auf ihre Eignung für ubiquitäre Rechnersysteme analysiert.

7.2.3 Grundlagen: Batterien

Schlüssel für viele Energiemanagementverfahren ist das Wissen um die zur Verfügung stehende Batteriekapazität. Aus ihr kann die Nutzungsdauer und schließlich die Betriebsdauer des Rechnersystems abgeleitet werden. Dafür ist ein Verständnis der Funktionsweise von Batterien notwendig.

Batterien wandeln die in ihnen gebundene chemische Energie in elektrische Energie um. Man unterscheidet Primärzellen, bei denen dieser Vorgang nicht umkehrbar ist, und Sekundärzellen oder Akkumulatoren, die zusätzlich die umgekehrte Umwandlung ermöglichen und wiederverwendbar sind. Für die folgenden Betrachtungen werden die Begriffe Batterie und Akkumulator synonym verwendet. Je nach Batteriechemie unterscheidet man verschiedene Batterietypen. Gebräuchlich sind Nickel-Metallhydrid (NiMH), Nickel-Cadmium (NiCd) und Lithium-Ionen (Li^+) Batterien. Große elektrische Leistungen können von Bleiakkumulatoren aufgebracht werden.

Das Verhalten einer Batterie bei Entladung hängt von Einflussfaktoren wie Batteriechemie, Entladestrom, Temperatur, Alter, Ladezyklen und anderen ab. Mehrere elektrochemische Modelle für unterschiedliche Batterietypen wurden in der Literatur beschrieben [139][140][141]. Diese Modelle beschreiben die Elektrochemie der Batterien mittels partieller Differentialgleichungen, die gewöhnlich eine große Anzahl an Parametern enthalten und spezifisch für einen bestimmten Batterietyp sind. Die Lösung ist numerisch sehr aufwendig [142] und kann daher nicht auf ubiquitären Rechnersystemen ausgeführt werden. Jedoch lassen sich durch Messungen einige Zusammenhänge empirisch bestimmen, die in den folgenden Paragraphen aufgezeigt werden.

7.2.3.1 Rate-Capacity Effekt

Die Kapazität C einer Batterie bestimmt den Entladestrom I, den sie über die Betriebsdauer $t^{\text{betr.}}$ liefern kann. Im Idealfall kann über den Zusammenhang $C = t^{\text{betr.}} I$ die Kapazität berechnet werden.

Reale Batterien unterliegen jedoch dem *Rate-Capacity Effekt*, der die nutzbare Kapazität in Abhängigkeit vom Entladestrom einschränkt. Dieser Umstand ist die Basis der Peukert Gleichung [143], die die nutzbare Batteriekapazität C unter verschiedenen, konstanten Entladeströmen I modelliert. Peukerts Gleichung beschreibt diesen Zusammenhang als *empirische* Gleichung

$$C = t^{\text{betr.}} I^{\alpha} \tag{7.1}$$

Der Parameter $\alpha > 1$ bezeichnet den batteriespezifischen Peukertwert, der typischerweise mit Werten aus dem Intervall $1.2 \leq \alpha \leq 1.4$ angegeben wird. Mit Gleichung 7.1 kann nach Messung des Entladestromes die Restkapazität bestimmt werden. Peukerts Gleichung wurde ursprünglich für Bleiakkumulatoren entwickelt. In der linearisierten Form [144] wird sie auch für das Energiemanagement mobiler Geräte, zum Beispiel Audio Recorder, mit kleinen Entladeströmen zwischen 15mA und 760mA [145], verwendet. Abbildung 7.2 zeigt als Folge des Rate-Capacity Effekts die Änderung der Batteriespannung für verschiedene Entladeströme. Fortgeschrittenere Batteriemodelle wie das Kinetic Battery Model (KiBaM) [146], SPICE [147] und Battery Energy Storage Test (BEST) Modell [148] berücksichtigen veränderliche Entladeströme. Sie müssen für den jeweiligen Batterietyp kalibriert werden.

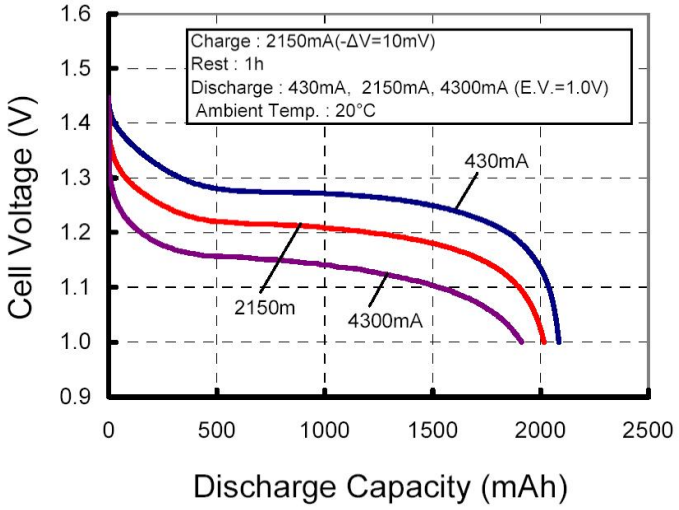


Abbildung 7.2: Änderung der Batteriespannung als Folge des Rate-Capacity Effekts für verschiedene Entladeströme. Batterie: Sanyo HR-4/5AU Quelle: http://batteries.sanyo-component.com/

7.2.3.2 Recovery Effekt

Beim Recovery Effekt kann die Kapazität einer Batterie nach hohen Entladeströmen ansteigen. Er entsteht durch Ladungsumverteilung durch Ionendiffusion in der Batterie zum Ausgleich des zuvor entstandenen Potentialgefälles, nachdem die Batterie von einem hohen Entladestrom getrennt wurde. Der Effekt ist als erhöhte Batteriespannung in dieser Phase messbar. Energiemanagementmechanismen können dieses Verhalten nutzen, um der Batterie nach hohen Strömen Zeit zur Regenerierung der internen chemischen Gleichgewichte zu geben. Dadurch wird die nutzbare Kapazität erhöht und die Betriebsdauer wird verlängert [143]. Diese Wirkung des Recovery Effekts als Folge abschnittsweiser Entladung und Regenerierung ist in Abbildung 7.3 dargestellt. KiBaM verwendet ein mechanisches Analogon dieser Vorgänge und kann den Recovery Effekt intuitiv beschreiben. [150] zeigte durch Messung und Simulation

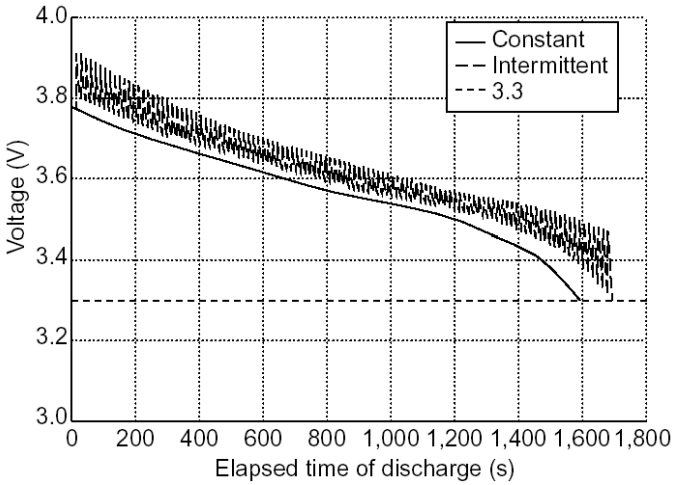


Abbildung 7.3: Phasenweise Entladung erreicht durch den Recovery Effekt eine längere Betriebsdauer im Vergleich zur kontinuierlichen Entladung. Quelle: [149]

einer Li^+ Batterie wie sie für Laptop Computer eingesetzt wird, dass beim Auftreten

von Spitzenströmen der Rate-Capacity Effekt die nutzbare Kapazität dominiert. Im Vergleich dazu hat der Recovery Effekt hat keine signifikanten Kapazitätsauswirkungen. Für ubiquitäre Rechnersysteme, die eine typische Leistungsaufnahme von 30 - 50 mA haben, ist der Recovery Effekt daher nicht relevant und wird nicht berücksichtigt.

7.2.3.3 Auswirkung der Temperatur, Lagerung und Ladezyklen

Die Umgebungstemperatur beeinflusst die chemische Reaktion zur Erzeugung der elektrischen Energie. Am besten arbeiten Batterien bei Zimmertemperatur von 20°C. Höhere Temperaturen reduzieren die Kapazität um 20% (bei 30°C) bis 50% (bei 45°C). Bei niedrigen Temperaturen nimmt Leistung der Batterie drastisch ab. Bei 0°C nimmt die verfügbare Kapazität schon um 10% ab. Bei −20°C arbeiten die NiMH, Säure-Blei und Li^+ Batterien nicht mehr.

Die verfügbare Kapazität sinkt, während Batterien ungenutzt gelagert werden. Der Grund ist Selbstentladung durch Alterung der Batteriechemie. Eine NiMH Batterie verliert 40% ihrer Kapazität innerhalb eines Monats. Die Selbstentladung wird durch höhere Temperaturen beschleunigt, da die Batteriechemie schneller altert.

Die Kapazität wiederaufladbarer Batterien nimmt mit der Anzahl der Ladezyklen ab. Das chemische Ausgangsmaterial wird beim Laden verändert und abgebaut. Der Effekt ist vergleichbar mit der Alterung. Wiederaufladbare NiMH-Zellen haben nach 500 Ladezyklen nur noch ca. 90% ihrer ursprünglichen Kapazität.

7.2.3.4 Spannungsverlauf

Die Spannung einer Batterie bleibt über die Entladezeit nicht konstant. Spannungskurven sind bis auf Recovery Effekte immer monoton fallend. Die Steilheit wird primär durch Batterietyp und Entladestrom bestimmt. Sie ist jedoch über die Zeit nicht konstant. Rechnersysteme benötigen eine Mindestspannung zum Betrieb. Wird dieser Wert unterschritten, so fällt das System aus. Der Schwellenwert wird auch als Cut-off Voltage bezeichnet.

NiMH Batterien weisen am Anfang und Ende der Entladedauer stark fallende Spannungsverläufe auf. Während des Großteils der Entladezeit ist der Spannungsabfall sehr gering. Lithium Zellen weisen einen sehr regelmäßigen Abfall der Spannung auf. Der Spannungsverlauf ändert sich noch einmal, wenn verschiedene Entladeströme betrachtet werden. Sowohl Änderungen der Steilheit wie auch eine Verschiebung in einen anderen Spannungsbereich sind zu erwarten. Beispiele des Spannungsverhaltens werden in Abbildung 7.4 gezeigt.

Obwohl ein universeller Zusammenhang zwischen Kapazität respektive Nutzungsdauer und der Batteriespannung besteht, machen es diese Effekte sehr schwer, von der gemessenen Spannung auf die Nutzungsdauer zu schließen.

7.2.3.5 Messung

Zur Bestimmung der Batteriekapazität werden elektrische Kenngrößen wie Stromaufnahme und Spannung erfasst. Spezialhardware wie Batteriestatusmonitore, zum Beispiel Maxim DS2764 [152], erfassen zusätzlich noch Temperatur, Alter und Ladezyklen. Fuel Gauges, zum Beispiel Maxim DS2781 [153], erweitern diese Messungen und schätzen die Batteriekapazität anhand von gespeicherten Kapazitätskurven.

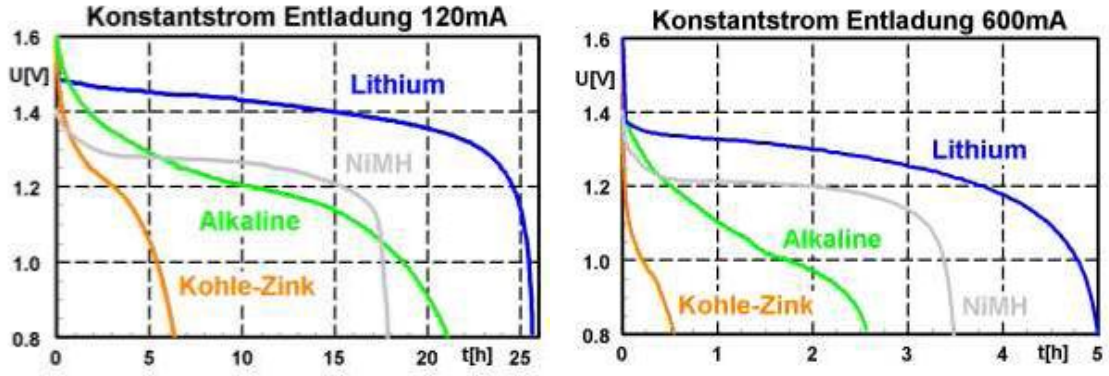


Abbildung 7.4: Beispiele des Spannungsverlaufs bei konstantem Entladestrom für eine Lithium, Alkali (Energizer Ultra+), Kohle-Zink und NiMH (Energizer 2100mAh) Batterie. Links: Entladestrom 120mA Rechts: Entladestrom 600mA. Quelle: [151]

Kenngröße	**Messung**	**Bewertung**
Spannung	direkte Analog-Digital Wandlung (ADC) der Batteriespannung	(+) einfach zu realisieren, kaum zusätzlicher Hardwareaufwand, ADC ist bereits in MCU integriert, (−) Spannungsverlauf nicht regelmäßig, schwer über längere Zeitperioden vorhersagbar
Stromaufnahme	Messung des Spannungsabfalls über festen Shunt-Widerstand, Anw. des ohmschen Gesetzes $I = \frac{U}{R}$	(+) geringer Hardwareaufwand, (−) Messung ungenau, da Shunt Widerstand sich ändert, benötigt unabhängige MCU, da sonst Messung des Messprogramms
Statusmonitor DS2764	momentane Stromaufnahme, kumulierte Stromaufnahme, Spannung, Temperatur	(+) akkurate, unabhängige Messung vieler Kapazitätseinflussgrößen, (−) signifikanter Hardwareaufwand und zusätzliche Anschaffungskosten
Fuel Gauge DS2781	momentane Stromaufnahme, kumulierte Stromaufnahme, Spannung, Temperatur, Ladezyklen, Alter	(+) schätzt akkurat die Batteriekapazität, kein zusätzlicher Rechen- oder Speicheraufwand, (−) signifikanter Hardwareaufwand und zusätzliche Anschaffungskosten, oft nur auf einen Batterietyp (Li^+) beschränkt

Tabelle 7.3: Überblick über Messmethoden und ihre Eignung für den Einsatz auf ubiquitären Rechnersystemen.

Vor- und Nachteile der Methoden sind in Tabelle 7.3 zusammengefasst. Der Aufwand für den Einsatz von integrierten Lösungen wie Statusmonitor und Fuel Gauge umfasst zusätzliche Bauteile für die Beschaltung, höhere Herstellungskosten, mehr Raumbedarf auf dem Platinendesign und die Anschaffungskosten der integrierten Schaltkreise. Für kostensensitive ubiquitäre Rechnersysteme sind daher die einfachen Methoden zur Spannungs- und Strommessung zu bevorzugen.

7.2.4 Kategorisierung der Energiemanagementverfahren

Verfahren zum Energiemanagement unterteilen sich in zwei Kategorien. Die erste Gruppe, offene Verfahren (engl. open-loop policies), verändert die Systemausführung so, dass ein bestimmtes Energieverbrauchsprofil erreicht wird. Zum Beispiel wird je nach Benutzerverhalten eines tragbaren MP3 Players in verschiedene Ruhezustände, wie Leerlauf oder Ausgeschaltet, gewechselt. Die zweite Gruppe, geschlossene Verfahren, engl. closed-loop policies, berücksichtigt die Kapazität und die Entladeströme der Batterie und führt diese Information an das Managementverfahren zurück. Die Batteriekapazität nach Peukert aus Gleichung 7.1, ist Rückkopplungsinformationen für viele geschlossene Verfahren.

Weiterhin werden Verfahren nach dem Teil des Rechnersystems unterschieden, der beeinflusst wird. Es werden weiterhin die folgenden Kategorien gebildet.

Batterie-zentrisch Es wird nur das Batterieverhalten und die effiziente Nutzung der Kapazität betrachtet. Rechnersystem, Peripheriekomponenten und Softwareverhalten werden vollständig als ein einzelner Energieverbraucher abstrahiert.

Scheduling-zentrisch Der Energieverbrauch der Ausführung einzelner Teile der Software des Rechnersystems steht im Mittelpunkt der Betrachtung.

Duty Cycling Die Aktivität des Rechnersystem wird in zeitliche Arbeits- und Ruhephasen unterteilt. In Arbeitsphasen kann die Batterie maximal genutzt werden. In Ruhephasen wird das Rechnersystem nahezu ausgeschaltet.

Im folgenden werden die Verfahren vorgestellt, ihre Eigenschaften analysiert und ihre Eignung für ubiquitäre Rechnersysteme bewertet.

7.2.5 Batterie-zentrische Ansätze des Energiemanagements

Auf Basis eines Batteriemodells wird durch Steuerung der Entladung einer oder mehrerer Batterien der Recovery Effekt nutzbar gemacht. Kurze, gepulste Entladung [154] oder auch der längerfristige Wechsel in Systemzustände [155] mit niedrigerer Leistungsaufnahme schaffen die Möglichkeit zur Regenerierung einer einzelnen Batterie und der effizienteren Nutzung der Batteriekapazität. In einer virtuellen Batterie sind mehrere Batterien kombiniert [156]. Sie werden in festgelegter Reihefolge, zufällig oder stückweise nacheinander entladen [157]. Dabei wird der Entladestrom auf die Einzelbatterien des Verbundes verteilt. Geschlossene Verfahren erlauben heterogene Batteriekombinationen und entladen die Einzelbatterien gemäß ihrer Restkapazität [145]. Die Implementierung umfasst zusätzliche Hardware für Umschalten und Messen des Batterienverbundes.

Vorteile: Die Ansätze abstrahieren vollständig von den Details des Rechnersystems und der Prozessausführung. Sie sind somit vielseitig einsetzbar.
Nachteile: Ein akkurates Batteriemodell ist notwendig. Die Vielzahl an Batterien und die Modellierung der elektrochemischen Effekte erfordern zusätzliche Mess- und Umschalthardware. Recovery Effekt ist für ubiquitäre Rechnersysteme nicht signifikant (siehe Abschnitt 7.2.3).
Eignung: bedingt - der zusätzliche Hardwareaufwand und die Genauigkeit des Batteriemodells müssen hinsichtlich des Einsatzszenarios abgewogen werden.

7.2.6 Scheduling-zentrische Ansätze des Energiemanagements

Verfahren dieser Kategorie verändern die Ausführungsreihenfolge (Schedule) und Ausführungsgeschwindigkeit von Prozessen, um Leistungsspitzen zu verhindern. Die Systemsoftware wird hinsichtlich ihrer Leistungsaufnahme charakterisiert [158]. Gebräuchliche Energiemanagementverfahren umfassen Task-Sequencing [159] und Spannungs- und Frequenzregelung (V/f Regelung) [150]. Letztere sind geschlossene Verfahren und berücksichtigen die Batterieverhalten wie Rate-Capacity Effekt und Recovery Effekt für einzelne Batterien und Verbünde. Ein signifikanter Rechenaufwand entsteht bei der Berechnung eines energiegewahren Schedule, der Zeitschranken und Abhängigkeiten von Prozessen untereinander berücksichtigt [160]. Es muss ein Optimierungsproblem gelöst werden. Die Implementierung erfolgt in Software.
Vorteile: Berücksichtigung von Zeitschranken und Abhängigkeiten von Prozessen. Leistungsspitzen werden verhindert.
Nachteile: Wissen um den zeitlich veränderlichen Energieverbrauch der Prozesse erfordert internes Wissen über Prozessaufgaben oder Messung und Profiling. Um bei V/f Regelung die Kommunikation mit Peripheriekomponenten, zum Beispiel Sensoren, nicht zu stören, muss ebenfalls internes Wissen über die Prozessaufgaben genutzt werden.
Eignung: ungeeignet, da Wissen um die Prozessaufgaben nicht vorausgesetzt werden soll.

7.2.7 Duty Cycling

Der Duty Cycle (DC) beschreibt das Verhältnis von Arbeitsphase mit der Dauer $t^{\text{akt.}}$ zur Zykluslänge, zusammengesetzt aus Arbeits- und Ruhephase $t^{\text{akt.}} + t^{\text{inakt.}}$, nach der sich die Arbeitsphase wiederholt.

$$V^{\text{DC}} = \frac{t^{\text{akt.}}}{t^{\text{akt.}} + t^{\text{inakt.}}} \tag{7.2}$$

Die Prozessausführung erfolgt in $t^{\text{akt.}}$. Duty Cycling bietet hohes Energiesparpotential, da das System in der Ruhephase ausgeschaltet wird. In der Arbeitsphase gibt es keine Einschränkungen bei Prozessausführung oder Batterienutzung. Im Unterschied zum Dynamic Powermanagement wird es für Systeme mit geringer Leistungsaufnahme von typischerweise 50 mW eingesetzt, bei denen die Wirkung des Recovery Effekts nicht signifikant ist. Gleichung 7.2 ist unterbestimmt. Der Duty Cycle wird daher in der Lösung eines Optimierungsproblems zur Maximierung der Betriebsdauer des Rechnersystems bestimmt oder oftmals an die Kommunikationsperioden gekoppelt. Alle Prozesse sind dann dem Kommunikationsverhalten untergeordnet [19]. Quasi-Markov Sleep Scheduling [161] bestimmt den Duty Cycle für jedes Rechnersystem lokal, um den Kommunikationsaufwand für die Synchronisierung mit anderen

Systemen zu verringern. Nachrichten werden während eines Niedrigenergiemodus der Kommunikationsschnittstelle gepuffert. Verteilte Verfahren [162][163] verwenden Wissen über Sensorsampling und Kommunikation und teilen die Aufgaben auf mehrere Netzwerkstationen auf, die nacheinander aktiviert werden. Eine optimale Aktivierungsdauer in der Arbeitsphase in Abhängigkeit der Stationenanzahl in der Sensor- und Kommunikationsreichweite wird erreicht.

Alle Mikrocontroller, die auf drahtlosen Sensorknoten und eingebetteten, ubiquitären Rechnersystemen zum Einsatz kommen, unterstützen einen extrem energiesparenden Ruhemodus. Diese Voraussetzung ermöglicht eine einfache Realisierung. Duty Cycling ist das dominierende Verfahren zum Energiemanagement auf diesen Plattformen.
Vorteile: Funktioniert mit jeder Batterietechnologie und kann auf Prozessausführung abgestimmt werden; zum Beispiel schaltet TinyOS die Hardware in den Ruhemodus, sobald keine Taskausführung geplant ist.
Nachteile: Fehlt Wissen über Prozessaufgaben, dann fehlen Kriterien zur Bestimmung der Zeitintervalle für Arbeits- und Ruhephasen. Auftreten von Leistungsspitzen in der Arbeitsphase ist möglich.
Eignung: Universell einsetzbar hinsichtlich Plattformen und Batterietypen. Offen ist die Bestimmung der Phasendauern ohne explizites Prozesswissen.

7.2.8 Fazit und Ansatz des kooperativen Energiemanagements

Es müssen entweder ein zusätzlicher Hardwareaufwand geleistet werden (batteriezentrische Ansätze) oder internes Wissen über Prozessaufgaben respektive ein Energieprofil der Systemsoftware verfügbar sein (scheduling-zentrische Ansätze). Der Anspruch an ubiquitäre Rechnersysteme ist jedoch ein vielfältiger Einsatz in Szenarien, in denen die Bedingungen der Prozessausführung unbekannt sind.

Universell einsetzbar in Verbindung mit allen Batterietypen und auf typischen Rechnerplattformen ubiquitärer Systeme wie sie in dieser Arbeit betrachtet werden, sind Duty Cycling Ansätze. Zur Bestimmung der Arbeits- und Ruhephasen des Duty Cycle Intervalls können verschiedene Kriterien flexibel eingesetzt werden. Der Beitrag dieses Kapitels ist ein neues *geschlossenes, kooperatives Energiemanagementverfahren*. Informationen über die Batteriespannung werden zurückgekoppelt, um Länge und Unterteilung des Duty Cycle Intervalls zu bestimmen. Kooperationsziel der abschnittsweisen Prozessausführung ist das Erreichen einer vorgegebenen Betriebsdauer.

In den folgenden Abschnitten wird gezeigt, dass diese Informationen genügen, um ein *optimales Duty Cycling* Verhalten *ohne internes Prozesswissen und Energieprofil* in unbekannten Einsatzumgebungen zu realisieren.

7.3 Problembeschreibung

Ziel des Energiemanagements ist es, in unbekannten Einsatzumgebungen batteriebetriebener ubiquitärer Rechnersysteme, eine vorgegebene Betriebsdauer und minimale Systemleistung respektive Mindestansprechbarkeit des Rechnersystems zu erreichen.

Universell für alle Batterien und am einfachsten messbar ist der Zusammenhang zwischen Batteriespannung und Betriebsdauer eines Rechnersystems. Die gemessene Batteriespannung gibt Aufschluss über den Energievorrat des Systems und damit die Betriebsdauer. Das Problem ist, dass zuerst nicht unterscheidbare Spannungsverläufe mit sehr verschiedenen Betriebsdauern korrelieren. Ursache ist das in Abschnitt 7.2.3 analysierte *nicht-ideale Entladeverhalten* von Batterien. Dies wird in Abbildung 7.5 deutlich. Sie zeigt drei gemessene Spannungsverläufe bis zu einer Cut-off Voltage $V_{\text{cut}} = 1190\text{mV}$. In jeder Messreihe wurden Prozesse mit einer anderen Periode, aber jeweils *konstantem* Entladestrom, ausgeführt. Bis zu einer Betriebs-

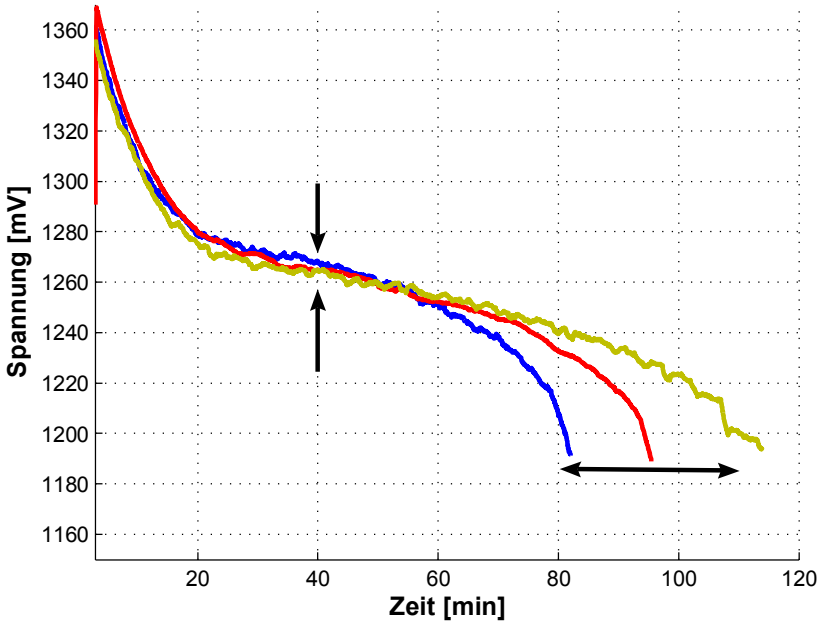


Abbildung 7.5: Spannungs-Zeit-Diagramm für drei verschiedene, periodische Prozessausführungen auf der mit einer NiMH Batterie betriebenen Particle Computer Plattform. Die Pfeile symbolisieren, wie die zuerst nicht unterscheidbaren Spannungsverläufe zu sehr verschiedenen Betriebsdauern führen.

dauer von 60 Minuten ist der Verlauf sehr ähnlich. Die Spannungsverläufe schneiden sogar mehrmals einander. Erst danach trennen sie sich und erreichen V_{cut} bei 80, 95 und 110 Minuten. Eine Betriebsdauervorhersage basierend auf den Spannungswerten kann über lange Zeiträume diese Informationen nicht nutzen. Prozesse mit variablen Entladeströmen wechseln Spannungs-Zeit-Kurven in einer Kurvenschar bis schließlich eine Kurve, die mit der tatsächlichen Betriebsdauer korreliert, dominiert. Zuerst nicht unterscheidbare Spannungsverläufe führen zu sehr verschiedenen Betriebsdauern.

Duty Cycling kann mit abschnittsweisen Ruhephasen, in denen kein Prozess ausgeführt wird, einen bekannten Spannungsverlauf erzwingen und somit eine vorgegebene Spannungs-Zeit-Kurve erreichen. Dafür ist die Information über die Batteriespannung ausreichend. Duty Cycling schaltet die Prozessausführung und das Rechnersystem nach der aktiven Arbeitsphase ab. Die inaktive Ruhephase muss auf einen endlichen Wert begrenzt werden, um eine minimale Prozessleistung respektive Mindestansprechbarkeit zu garantieren.

Es leitet sich die zentrale Problemformulierung des Energiemanagements für ubiquitäre Rechnersysteme ab.

Problem 7.1 (Energiemanagement)**.** *Für ubiquitäre Rechnersysteme mit der Energiecharakterisierung aus Abschnitt 7.2.2, bestimme durch Spannungsmessung einer Batterie mit nicht-idealem Entladeverhalten bei Prozessausführung die Arbeits- und Ruhephasen des Duty Cycling so, dass mit dem vorhandenen Energievorrat*

1. *eine vorgegebene Betriebsdauer bestmöglich erreicht wird, und*
2. *eine minimale Systemleistung respektive Mindestansprechbarkeit garantiert wird.*

7.4 Entwurf

Für die Realisierung des Duty Cycling ist eine zeitliche Unterteilung der Ausführung erforderlich. Die Prozessausführung mit Services und Prozessen, wie in Abschnitt 4.3 eingeführt, wird in zeitliche Rahmen organisiert. In einem einzelnen Rahmen der Dauer $t^{\text{rahm.}}$ werden sowohl aktive Arbeitsphasen $t_i^{akt.}$, in denen Prozesse der Dauer C_i ausgeführt werden, wie auch inaktive Ruhephasen $t_i^{inakt.}$, in denen das Rechnersystem abgeschaltet wird, eingegliedert. Die Unterteilung ist in Abbildung 7.6 gezeigt. Ein Rahmen beginnt immer mit einer Prozessausführung. Dabei kann entweder die

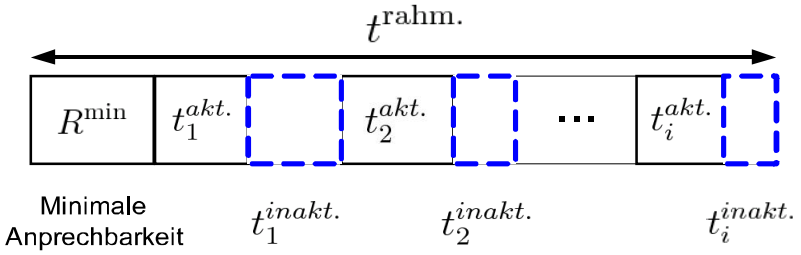


Abbildung 7.6: Unterteilung der Prozessausführung innerhalb eines Zeitrahmens mit Arbeits- und Ruhephasen.

Anzahl der Prozesse $N^{\min}$ oder die reservierte Zeitdauer der Prozessausführung bestimmt werden. Letztere definiert nur eine untere Grenze $C^{\min}$, da die konkrete Ausführungsdauer unbekannt ist. Das Verhältnis von Zeitdauer respektive Prozessanzahl und Rahmendauer definiert die Ansprechbarkeit oder minimale Systemleistung.

$$\begin{aligned} R^{\min} &= \frac{N^{\min}}{t^{\text{rahm.}}} \\ &\leq \frac{C^{\min}}{t^{\text{rahm.}}} \end{aligned} \tag{7.3}$$

Über die Vorgabe von $R^{\min}$ durch einen Entwickler wird auch die Rahmenlänge $t^{\text{rahm.}}$ festgelegt. Sinnvollerweise wird die Rahmenlänge mit der Periode des wichtigsten Prozesses, zum Beispiel Kommunikation, identifiziert. Schließlich komponiert sich der Rahmen zu

$$\begin{aligned} t^{\text{rahm.}} &\geq R^{\min} + \sum_i t_i^{\text{akt.}} + \sum_i t_i^{\text{inakt.}} \\ \text{mit} \quad & t_i^{\text{akt.}} = C_i \end{aligned}$$

Aufgrund der nicht-präemptiven Ausführung und unbekannter Ausführungsdauer der Prozesse lässt sich der =-Operator nicht garantieren. Die Mindestansprechbarkeit kann daher nur bis auf diesen Jitter eingehalten werden.

Es wird nun ein neues Verfahren für die Regelung der Nutzung des Energievorrates (Power control, kurz: PControl) vorgestellt. Die Architektur ist in Abbildung 7.7 gezeigt. PControl bestimmt Arbeits- und Ruhephasen der Prozessausführung. Es

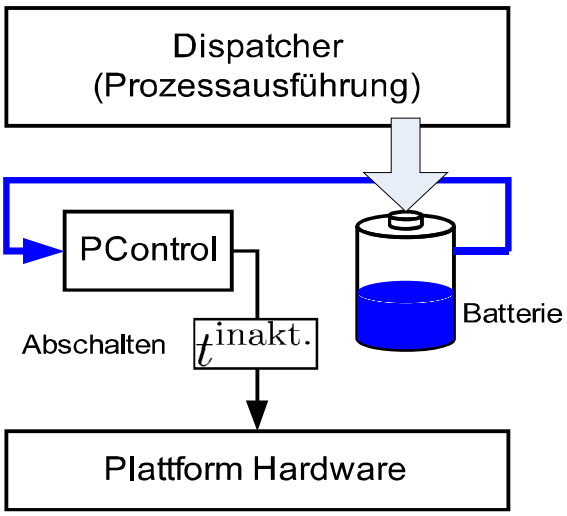


Abbildung 7.7: Entwurf des PControl Verfahrens zur Regelung der Nutzung des Energievorrates. PControl schließt die Batterie ein, um die Abschaltdauer $t^{\text{inakt.}}$ das Rechnersystems zu bestimmen.

schließt die Batterie ein und ermittelt den Energievorrat durch Messung der Batteriespannung. In dem geschlossenen Verfahren wird bei geringem Energievorrat vor dem Erreichen der Betriebsdauer die Ruhephase verlängert. Die Längen der Arbeitsphasen ergeben sich implizit durch die Prozessausführungsdauer. Die Dauer der Ruhephase $t^{\text{inakt.}}$ ist die Stellgröße von PControl.

Zu Beginn eines Rahmens werden die Prozesse mit der Periode der Rahmenlänge ausgeführt. PControl ist für die Dauer von $R^{\min}t^{\text{rahm.}}$ aktiv. Diese Maßnahme garantiert die Mindestansprechbarkeit. Für die verbleibende Dauer führt PControl nach jeder Prozessausführung i die folgenden Schritte aus:

1. Bestimmung des aktuellen Energievorrates V_i
2. Vergleich mit dem zu erreichenden Vorrat B
3. Berechnung der Dauer der inaktiven Ruhephase $t_i^{\text{inakt.}}$, um B zu erreichen. Die inaktive Ruhephase sollte spätestens bei $t^{\text{rahm.}}$ enden, um die Ansprechbarkeit zu gewährleisten.

Die PControl geregelte Prozessorganisation kombiniert aktive und inaktive Phasen. Es werden keine prozessinternen Informationen, zum Beispiel über die konkrete Prozessaufgabe, verwendet. PControl ist ein Mechanismus im Laufzeitsystem, der unabhängig von der Prozessaufgabe die Energienutzung regelt. Sollen in diesem Zusammenhang Prozessausführungen garantiert werden, so müssen diese in $R^{\min}$ eingeplant

werden. Durch Setzen der Ausführungsperiode jener Prozesse auf die Rahmenzeit kann dies vom Entwickler bestimmt werden.

Zusammenfassend sind in Tabelle 7.4 die Entwurfsparameter der PControl Regelung des Duty Cycle angegeben. Der Duty Cycle, der durch PControl realisiert wird, ist

Parameter	**Eigenschaft**	**Erklärung**
$t^{\text{betr.}}$	konst.	Betriebsdauer, wird bei Systemstart festgelegt
$t^{\text{rahm.}}$	konst.	Rahmendauer, festgelegt durch $R^{\min}$
$t^{\text{akt.}}$	$\sum_i C_i$	Konkrete Summe ist unbekannt. Die Anzahl i der ausgeführten Prozesse wird von PControl bis zum Ende von $t^{\text{rahm.}}$ sukzessive erhöht.
$t^{\text{inakt.}}$	$\sum_i t_i^{\text{inakt.}}$	Die Einzeldauern $t_i^{\text{inakt.}}$ sind variabel. Es ist die Stellgröße von PControl und wird gemäß Energievorrat und Betriebsdauer geregelt. Es ist $t^{\text{inakt.}} < t^{\text{rahm.}}$.
$R^{\min}$	konst.	Ansprechbarkeitsgarantie gemäß Gleichung 7.3

Tabelle 7.4: Entwurfsparameter der PControl Regelung.

pro Frame und je nach Leistungsaufnahme der Prozesse und Energievorrat variabel. Der konkrete Wert ist für die Problemlösung nicht relevant.

7.5 Systemmodellierung des Energiemanagements

Im unmittelbar ersten Modellierungsschritt wird die Methode zur Übertragung des Budget/Kosten Modells auf spezifische Prozessklassen aus Abschnitt 3.4.4 bemüht. Es sind Budget, Kosten, Ziel des Energiemanagements zu identifizieren. Sie sind in Tabelle 7.5 zusammengefasst. Der Energievorrat der Batterie wird als Spannungs-

Budget	Energievorrat der Batterie
Kosten	Prozessausführung
Ziel	Eine vorgegebene Betriebsdauer wird bestmöglich erreicht.

Tabelle 7.5: Entsprechungen des Budget/Kosten Modells im Energiemanagement

budget identifiziert. Das Verhalten bei Entladung wird damit dem Spannungsverlauf gleichgesetzt. Das erscheint ungewöhnlich. Zentral im PControl Verfahren ist jedoch nicht die Vorhersage der Kapazität, sondern das Erreichen einer gegebenen Betriebsdauer. Diese wird immer erreicht, wenn die Batteriespannung bis zum Ablauf dieser Zeitspanne die Cut-off Voltage noch nicht unterschritten hat. Es lässt sich folgende Regelungsaufgabe von PControl ableiten:

Regelungsziel: Batteriespannungsverlauf mit Rate B innerhalb von $t^{\text{rahm.}}$

Stellgröße: Dauer der inaktiven Ruhephase $t^{\text{inakt.}}$

Regelstrecke: Verminderung der Batteriespannung durch Prozessausführung. Gemessene Batteriespannung bildet die Rückkopplungsgröße.

Störgröße: unbekannte Ausführungsdauer und Leistungsaufnahme, zum Beispiel durch Nutzung peripherer Komponenten, der Prozesse.

Obwohl die Regelungsaufgabe sich nur über einen Zeitrahmen erstreckt, wird durch Zusammensetzen der Rahmen auch das Ziel aus Tabelle 7.5 berücksichtigt. Für das Erreichen der Betriebsdauer ist es ausreichend, zu zeigen, dass jeder Zeitrahmen ein vorgegebenes (Teil-)Ziel bestmöglich erreichen kann. Die Zerlegung in Teilziele erlaubt zudem eine höhere Flexibilität. Es können dynamisch weitere Batterien eingebunden und das Ziel entsprechend angepasst werden.

7.5.1 Batterieverhalten und Spannungsbudget

Der Spannungsverlauf von Batterien bei Entladung lässt sich aufgrund der Analyse in Abschnitt 7.2.3 mit zwei Eigenschaften beschreiben. Sie lassen sich den Arbeits- und Ruhephasen des Duty Cycling zuordnen.

Arbeitsphase: Bei Prozessausführung vermindert sich die Kapazität und der Spannungsgradient ist negativ, d.h.

$$\Delta_{t^{\text{akt.}}} V < 0 \tag{7.4}$$

Ruhephase: Wird das Rechnersystem abgeschaltet, dann ist die Kapazitätsänderung und der Spannungsgradient nicht messbar.

$$\Delta_{t^{\text{inakt.}}} V = 0 \tag{7.5}$$

Die genannten Eigenschaften sind universell und können beim Entladen aller betrachteten Batterietypen aus Abschnitt 7.2.3 beobachtet werden. Der Recovery Effekt wird für die ubiquitäre Rechnersysteme dieser Arbeit aus zuvor analysierten Gründen vernachlässigt. Die Ausregelungszeit von PControl erstreckt sich nur über die Dauer eines Zeitrahmens. Kapazitätsminderung durch Temperaturänderung und Lagerung werden vernachlässigt, da die Zeitrahmendauer $t^{\text{rahm.}}$ klein gegenüber den Zeiträumen dieser Effekte ist. Die Batteriespannung im w-ten Duty Cycle resultiert aus der Summation der Spannungsgradienten. Die zugehörige Differenzengleichung ist $V(w) = V(w-1) + \Delta_{t^{\text{akt./inakt.}}} V$. Damit lässt sich die Transferfunktion des Batterieverhaltens angeben. Es ist

$$V(z) = \frac{z}{z-1} \tag{7.6}$$

die Transferfunktion der Regelstrecke. Gleichung 7.6 überführt das Batterieverhalten in den Arbeits- und Ruhephasen in ein Spannungsbudget. In dieser Formulierung bildet das Spannungsbudget nicht zwangsläufig die tatsächliche Batteriespannung ab. Es entspricht der Transferfunktion des Budgets im Budget/Kostenmodell.

7.5.2 PControl Regelung

Die PControl Regelung kombiniert Arbeitsphasen mit verschieden langen Ruhephasen, um einen vorgegebenen Spannungsbudgetverlauf zu erreichen. Die gesamte Betriebsdauer wird in k Zeitrahmen zerlegt. In jedem Rahmen wird der Spannungsverlauf mittels einer linearen Anstiegsfunktion $V_k^{\text{soll}}(t)$ mit dem Gradient $B_k < 0$ vorgegeben. Während des k-ten Zeitrahmens linearisiert PControl nach jeder Prozessausführung den Verlauf des Spannungsbudgets durch die Punkte $V_k(0) \ldots V_k(n)$. Das Vorgehen wird für jeden Rahmen wiederholt, um dem Spannungsverlauf bis

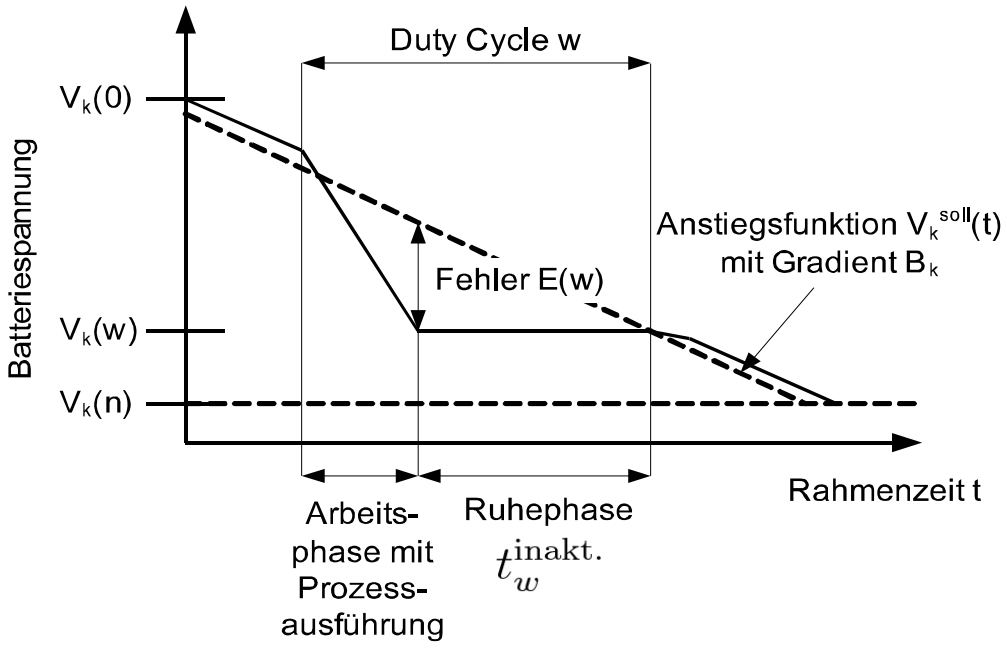


Abbildung 7.8: PControl linearisiert durch Ruhephasen den Spannungsverlauf nach der Prozessausführung.

zur Cut-off Voltage am Betriebsdauerende zu folgen. Die folgenden Betrachtungen beschränken sich auf die PControl Regelung im k-ten Zeitrahmen. Abbildung 7.8 illustriert beispielhaft den Ablauf in einem solchen Zeitrahmen. Die Rahmenzeit t gibt den aktuellen Zeitpunkt innerhalb eines Rahmen an. Der Duty Cycle w setzt sich aus $t_w^{\text{akt.}} + t_w^{\text{inakt.}}$ zusammen. Nach der Prozessausführung $t_w^{\text{akt.}}$ wird die Batteriespannung gemessen, in den Regelkreis eingekoppelt und das aktuelle Spannungsbudget $V_k(w)$ des w-ten Duty Cycle bestimmt. Existiert ein Fehler $0 < E(w) = V_k^{\text{soll}}(t_w^{\text{akt.}}) - V_k(w)$ wird er über die Ruhephase $t_w^{\text{inakt.}}$ entsprechend dem Regelungsziel $V_k^{\text{soll}}(t_w^{\text{akt.}} + t_w^{\text{inakt.}})$ kompensiert.

Der PControl Regler bildet die Dynamik $\Delta_{t^{\text{akt./inakt.}}} V$ des Batterieverhaltens aus Abschnitt 7.5.1 nach. Er modelliert die Kombination der auftretenden Spannungsgradienten in den Arbeits- und Ruhephasen. Mathematisch wird dies durch ein P-Regelungsgesetz $U(z) = K_p E(z)$ mit Verstärkung K_p beschrieben. Das Eingangssignal $E(z)$ des P-Reglers ist ein Spannungswert. Das Ausgangssignal $U(z)$ entspricht der Spannungsänderung ΔV, die an die Regelstrecke weitergegeben wird. Es wird das folgende Reglerverhalten modelliert:

Arbeitsphase: $K_p^{\text{akt.}} > 0$ verstärkt das Eingangssignal $E(z)$. Mit $E(z) < 0$ folgt $\Delta_{t^{\text{akt.}}} V < 0$ bei Prozessausführung.

Ruhephase: Mit $K_p^{\text{inakt.}} = 0$ wird die Dynamik der Batteriespannung ausgeschaltet. Es ist $\Delta_{t^{\text{inakt.}}} V = 0$. Dies entspricht dem Verhalten bei abgeschaltetem Rechnersystem.

Das Umschalten zwischen beiden Verstärkungen des P-Reglers geschieht durch einen Gain Scheduler [164]. Damit kann das nicht-lineare Verhalten der PControl Regelung mit Mitteln der linearen Regelungstheorie behandelt werden. Die Verschaltung

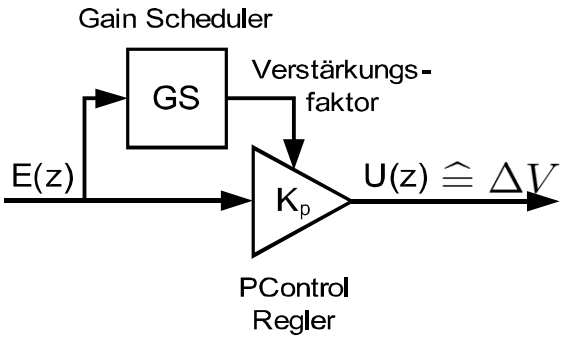


Abbildung 7.9: Der Gain Scheduler ändert in Abhängigkeit des Eingangssignals die Verstärkung des P-Reglers, um zwischen Arbeits- und Ruhephase zu wechseln.

beider Elemente ist in Abbildung 7.9 dargestellt. Ist die Spannungsminderung durch die Prozessausführung drastischer als von der Zielfunktion vorgegeben, zum Beispiel durch Hinzuschalten von Peripheriekomponenten, dann muss eine Ruhephase zur Kompensation erfolgen. Der Gain Scheduler entscheidet dies über die Abweichung vom Sollverlauf im Eingangssignal $E(w) = V_k^{\text{soll}}(t_w^{\text{akt.}}) - V_k(w)$. Ist $E(w) > 0$, dann wird eine Ruhephase eingeleitet, indem die Reglerverstärkung mit $K_p^{\text{inakt.}} = 0$ bestimmt wird. Das Rücksetzen erfolgt am Ende der Ruhephase. Die Transferfunktion des PControl Reglers mit Gain Scheduler in Z Transformation ist

$$P(z) = \frac{U(z)}{E(z)} = \begin{cases} K_p^{\text{inakt.}} = 0, & \text{wenn } E(w) > 0 \\ K_p^{\text{akt.}} > 0, & \text{sonst.} \end{cases} \tag{7.7}$$

Es ist nun zu zeigen, dass die Schaltzustände des Gain Schedulers mit der Dauer der Ruhephase $t_w^{\text{inakt.}}$ gekoppelt sind. Die Länge der Ruhephase kann aus Abbildung 7.8 wie folgt berechnet werden:

$$\begin{aligned} V_k(w) &= B_k\left(t + t_w^{\text{inakt.}}\right) + V_k(0) \\ &= B_k t_w^{\text{inakt.}} + \underbrace{B_k t + V_k(0)}_{=V_k^{\text{soll}}(t)} \\ &\text{gdw.} \\ t_w^{\text{inakt.}} &= -\frac{1}{B_k}\left(V_k^{\text{soll}}(t) - V_k(w)\right) \\ t_w^{\text{inakt.}} &= -\frac{1}{B_k}E(w) \end{aligned} \tag{7.8}$$

Mit Gleichung 7.8 kann nun folgender Zusammenhang zum Verhalten des P-Reglers aus Gleichung 7.7 etabliert werden.

- $\forall E(w) > 0 :\ t_w^{\text{inakt.}} > 0$. Unter dieser Bedingung bestimmt der Gain Scheduler in Gleichung 7.7 $K_p^{\text{inakt.}} = 0$ als Regelungsverstärkung. Das Rechnersystem ist in der Ruhephase.
- Für $E(w) = 0$ ist $t_w^{\text{inakt.}} = 0$. Gemäß Gleichung 7.7 schaltet der Gain Scheduler wieder auf $K_p^{\text{akt.}}$. Die Ruhephase ist abgeschlossen.

Die Ruhephasendauer $t_w^{\text{inakt.}}$ und die Schaltzustände des Gain Schedulers sind über die Abweichung $E(w)$ zum Sollverlauf miteinander gekoppelt. Beide haben identischen Einfluss auf die Regelstrecke. Dies hat folgende Konsequenzen:

1. Der P-Regler nimmt für die *gesamte Ruhephasendauer und nur für diese Dauer* den Wert $K_p^{\text{inakt.}} = 0$ an. Die Berechnung der Stellgröße $t_w^{\text{inakt.}}$ und der P-Regler können austauschbar verwendet werden.

2. In der Arbeitsphase ist $E(w) < 0$ und der P-Regler $K_p^{\text{akt.}} > 0$ erzeugt $\Delta_{t^{\text{akt.}}} V < 0$ (Eine Ausnahme bildet der Fall $E(w) = 0$, der ebenfalls zur Arbeitsphase gehört).

3. Die PControl Regelung kann mit dem Gain Scheduler gestützten P-Regler in Gleichung 7.7 beschrieben werden.

7.5.3 PControl Regelkreis

Damit ergibt sich der PControl Regelkreis wie in Abbildung 7.10 dargestellt. Der

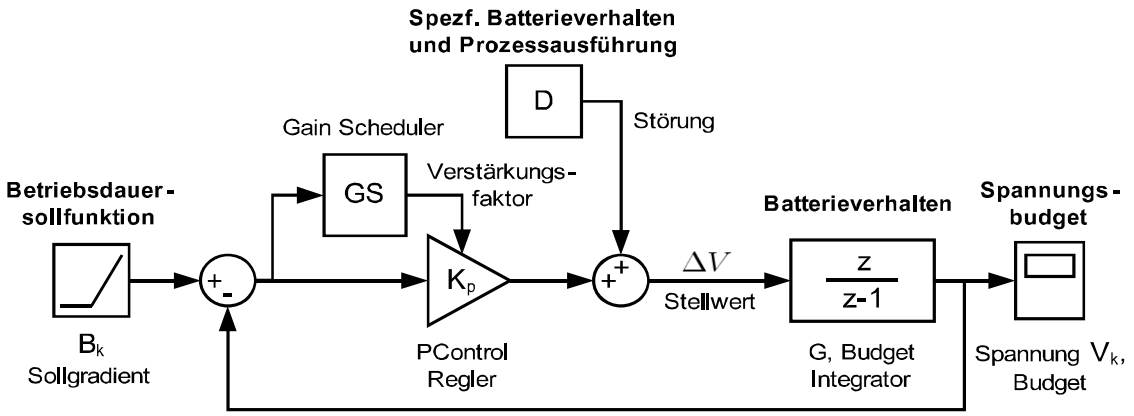


Abbildung 7.10: PControl Regelkreis als Simulink Modell

Regelkreis entspricht dem Budget/Kosten Modell. Die Transferfunktion des Regekreises für die PControl geregelte Prozessorganisation ist aus Abschnitt 3.4.3 sofort ableitbar. Es ist

$$F(z) = \frac{K_p z}{z(K_p + 1) - 1} \tag{7.9}$$

mit Gain Scheduler gestütztem K_p wie in Gleichung 7.7 definiert.

Die Betriebsdauersollfunktion ist eine stückweise lineare Anstiegsfunktion, die Betriebsdauer, Cut-off Voltage und Spannungsbudget in einen idealen Zusammenhang setzt. Sie erreicht nach vorgegebener Betriebsdauer die Cut-off Voltage. Der Spannungsverlauf im k-ten Zeitrahmen ist dann $V_k^{\text{soll}}(t) = B_k t + V_k(0)$. Der Gradient B_k setzt die Spannung zu Rahmenbeginn, die Cut-off Voltage V_{cut} und die Betriebsdauer miteinander in Beziehung. Es ist

$$B_k = \frac{V_k(0) - V_{\text{cut}}}{t - t^{\text{betr.}}}$$

$V_k(0)$ wird zu Beginn des Zeitrahmens gemessen. Es wird die Betriebsdauersollfunktion $V_k^{\text{soll}}(t)$ als Transferfunktion in Z Transformation wie folgt angegeben:

$$V_k^{\text{soll}}(z) = \frac{B_k}{(z-1)^2} \tag{7.10}$$

Gleichung 7.10 beschreibt das Eingangs- respektive Führungssignal des PControl Regelkreises $F(z)$. $F(z)$ überführt $V_k^{\text{soll}}(z)$ in das Spannungsbudget V_k.

Das Verhalten des Regelkreises wird weiterhin von der Störung D beeinflusst. D wird aus der Differenz der gemessenen Batteriespannung und der Betriebsdauersollfunktion ermittelt.

$$D(z) = \frac{V^{\text{batt}}}{(z-1)^2} - V_k^{\text{soll}}(z) \tag{7.11}$$

Gleichung 7.11 koppelt das sich unbekannt ändernde Batterieverhalten V^{batt} bei Prozessausführung in den Regelkreis ein.

7.5.4 Verständnis des Regelkreismodells

Es ist wichtig zu verstehen, wie die Zusammenhänge zwischen Prozessausführung, Energievorrat und gewünschter Betriebsdauer in einem kontrollierbaren, linearen Regelkreismodell erfasst werden. Die Konzeption soll im folgenden näher erläutert werden.

Der Regelkreis setzt das Spannungsbudget (= Ausgangssignal) in Relation zur geforderten Betriebsdauer (= Eingangssignal). Das Spannungsbudget hat keine physikalische Entsprechung. Es dient zur Bestimmung des *momentanen* Erfolges der Betriebsdauerregelung, d.h. ob der momentane Energievorrat für die bisherige Betriebsdauer ausreichend ist. Die Betriebsdauer ist als Parameter B_k des Eingangssignals V_k^{soll} in Gleichung 7.10 kodiert. Für das Erreichen der Betriebsdauer muss das Spannungsbudget dem Signal V_k^{soll} folgen.

Die Abweichung zwischen V_k^{soll} und *tatsächlichem* Spannungsmesswert wird als Störeinfluss D vor die Regelstrecke eingekoppelt. D umfasst alle unbekannten Einflüsse des Batterieverhaltens bei Prozessausführung. Ähnlich dem Spannungsverhalten einer Batterie hat jedes Auftreten einer Störung einen dauerhaften Einfluss auf das Spannungsbudget. Der Regelkreis bestimmt den Störeinfluss auf das Budget, kompensiert ihn, folgt dem Verlauf von V_k^{soll} und erreicht schließlich die geforderte Betriebsdauer.

Das Regelkreismodell spiegelt somit das tatsächliche Spannungsverhalten in Bezug zum Idealverlauf V_k^{soll} für das Erreichen der Betriebsdauer wider. Das Spannungsbudget ist Ausdruck des Momentanerfolges der Betriebsdauerregelung. Dieser Ansatz ermöglicht es, unbekanntes Prozess- und Batterieverhalten als Wirkungen auf ein lineares Regelungssystem zu beschreiben.

Ohne die Verwendung eines Energieprofils der Prozesse oder eines spezifischen Modells der Batteriekapazität formuliert der Regelkreis die Zusammenhänge zwischen Prozessausführungen, Energievorrat und gewünschter Betriebsdauer. Die Modellierung erlaubt eine mathematische Beschreibung für die folgende Analyse des kooperativen Verhaltens der PControl geregelten Prozessorganisation für das Energiemanagement.

7.6 Prozesskooperation

PControl regelt das Arbeits- und Ruhephasenverhältnis der Prozessausführungen gemäß der gemessenen Batteriespannung. Internes Prozesswissen, zum Beispiel über

die Ansteuerung peripherer Komponenten, oder Nutzungsinformationen aus einem Energieprofil, werden nicht verwendet. Alle Prozessausführungen wirken unabhängig voneinander auf den Energievorrat. Ziel der PControl geregelten Prozessorganisation ist es, eine vorgegebene Betriebsdauer bestmöglich zu erreichen. Nach Definition 3.2 entspricht dieses zielgerichtete Zusammenwirken einer Prozesskooperation.

Zur Bestimmung der Eigenschaften der Kooperation wird eine Fallunterscheidung gemäß der durch den Gain Scheduler bestimmten Reglerverstärkung K_p vorgenommen. Es werden Stabilität, Genauigkeit und Geschwindigkeit der PControl Regelung untersucht.

7.6.1 Verhalten während der Arbeitsphase

Das Verhalten in der Arbeitsphase wird durch das System $V_k(z) = V_k^{\text{soll}}(z) \cdot F(z)$ bestimmt. $V_k(z)$ bewertet den Erfolg der Betriebsdauerregelung. In der Arbeitsphase ist immer $K_p > 0$. Die Partialbruchzerlegung des Systems ergibt

$$V_k(z) = -\frac{B_k}{(z-1)K_p} + \frac{B_k(1+K_p)}{(z-1+K_pz)K_p} + \underbrace{\frac{B_k}{(z-1)^2}}_{V_k^{\text{soll}}(z)}$$

Die beiden ersten Terme beschreiben stabile Systeme. Die Impulsantworten konvergieren gegen $-\frac{B_k}{K_p}$ für den ersten Term und 0 für den zweiten Term. $V_k(z)$ folgt $V_k^{\text{soll}}(z)$ mit konstantem Abstand $-\frac{B_k}{K_p} > 0 (B_k < 0)$.

Der PControl Regelkreis besitzt ein Führungsverhalten bezüglich der Betriebsdauersollfunktion. Der positive Abstand garantiert, dass das Spannungsbudget größer als die Cut-off Voltage ist. Die vorgegebene Betriebsdauer wird erfolgreich erreicht.

Die Geschwindigkeit der Regelung gibt an wie schnell der Störeinfluss D einer Prozessausführung kompensiert wird. Der PControl Regler erzeugt dazu einen Spannungsgradienten $K_pE(z)$ aus dem Regelungsfehler $E(z)$. Die Dynamik des Stellwertsignals $\Delta_{t^{\text{akt.}}}V(z) = K_pE(z) + D(z)$ am Eingang der Regelstrecke $G(z)$ bestimmt die Geschwindigkeit der Störkompensation. Mit $E(z) = \frac{V^{\text{soll}}(z) - D(z)G(z)}{1+K_pG(z)}$ und $D(z)$ aus Gleichung 7.11 ergibt sich die Partialbruchzerlegung des Stellwertsignals zu

$$\begin{aligned}\Delta_{t^{\text{akt.}}}V(z) = &-\frac{(1+K_p)\left(-V_k^{\text{soll}}(z) + V^{\text{batt}} + V_k^{\text{soll}}(z)K_p\right)}{(z-1+zK_p)K_p} \\ &+ \frac{-V_k^{\text{soll}}(z) + V^{\text{batt}} + V_k^{\text{soll}}(z)K_p}{(z-1)K_p}\end{aligned}$$

Der erste Term beschreibt ein stabiles System mit Polstelle bei $\frac{1}{1+K_p}$. Die Impulsantwort konvergiert gegen 0 für alle $K_p > 0$. In der Arbeitsphase können die Prozesse die Batterie uneingeschränkt nutzen, d.h $\Delta_{t^{\text{akt.}}}V(z)$ ist maximal. Das System arbeitet am physikalischen Limit, festgelegt durch den tatsächlichen Verlauf der Batterie-

spannung V^{batt}. Mit diesem Zusammenhang kann die PControl-Reglerverstärkung K_p wie folgt synthetisiert werden:

$$\begin{aligned} \Delta_{t^{\text{akt.}}} V(z) &= \max \\ &\text{gdw.} \\ \frac{-V_k^{\text{soll}}(z) + V^{\text{batt}} + V_k^{\text{soll}}(z) K_p}{(z-1)K_p} &= \frac{V^{\text{batt}}}{(z-1)^2} \\ &\text{gdw.} \\ K_p &= 1 \end{aligned}$$

Für $K_p = 1$ operiert PControl in der Arbeitsphase *maximal schnell*. Die Geschwindigkeit der Störungskompensation ist ausschließlich durch den ersten Term der Zerlegung von $\Delta_{t^{\text{akt.}}} V(z)$ für $K_p = 1$ festgelegt. Der Störeinfluss ist nach $w \approx \frac{\ln 0.02}{\ln \frac{1}{K_p+1}} = 6$ Zeitschritten bis auf 2% kompensiert.

Der Länge des Zeitschritts des Regelkreises ist für alle Prozessausführungen konstant. Es wird die kleinste Prozessdauer angenommen. Die jeweils unterschiedliche Dauer der Prozessausführungen kann durch entsprechende Wahl von V^{batt} ausgedrückt werden.

7.6.2 Verhalten während der Ruhephase

Die Fähigkeit des Regelkreises, Störungen während der Arbeitsphase zu kompensieren, ist begrenzt. Tritt durch $D(z)$ im w-ten Duty Cycle ein Spannungsbudget $V_k < V_k^{\text{soll}}$ auf, bestimmt der Gain Scheduler aufgrund des nun auftretenden Regelungsfehlers $E(w) > 0$ die Reglerverstärkung zu $K_p = 0$. Das Rechnersystem tritt in die inaktive Ruhephase ein.

Für $K_p = 0$ ist der Regelkreis $F(z)$ unter allen Eingangsignalen konstant. In der Ruhephase ist das Störsignal $D(z) = 0$. Konstante Systeme sind inhärent stabil. Die Regelungsgeschwindigkeit gibt an, wie schnell der zu Beginn der Ruhephase im w-ten Duty Cycle aufgetretene maximale Regelungsfehler $E(w) > 0$ kompensiert wird. Der Ansatz zur Bestimmung der Geschwindigkeit führt über die Analyse des Fehlersignals $E(z)$. Dieses soll $E(w) > 0$ *maximal schnell* kompensieren. Dafür muss $E(z)$ maximale Amplituden aufweisen. Für dieses Optimierungsproblem ist das Regelungsverhalten zu bestimmen.

Das vom PControl Regler erzeugte Stellwertsignal $\Delta_{t^{\text{inakt.}}} V(z) = E(z) \cdot K_p$ muss der folgenden Bedingung genügen.

$$\text{Impulsantwort: } h\left(\Delta_{t^{\text{inakt.}}} V(z)\right) \leq 0 \tag{7.12}$$

Das bedeutet, Spannung kann nicht erzeugt werden. Das Stellwertsignal wird mit

$$\begin{aligned} \Delta_{t^{\text{inakt.}}} V(z) &= E(z) \cdot K_p \\ &= \frac{v(z)}{(z-1)^2} \end{aligned} \tag{7.13}$$

modelliert, wobei $v(z)$ zu bestimmen ist. Es wird nicht zwingend $v(z) = \text{konst.}$ gefordert. Mit Gleichung 7.12 folgt $h\left(v(z)\right) \leq 0$ für die Impulsanwort von $v(z)$. Schließlich

kann mit der Beziehung $E(z) = \frac{V_k^{\text{soll}}(z)}{1+K_p V(z)}$ und Gleichung 7.13 das Fehlersignal ohne Vorkommen von K_p formuliert werden:

$$E(z) = -\frac{(-B_k + v(z))z + B_k}{(z-1)^3} \tag{7.14}$$

Mit Gleichung 7.14 als Zielfunktion kann das folgende Optimierungsproblem für das Verhalten der PControl Regelung in der Ruhephase des Rechnersystems aufgestellt werden. Bestimme $v(z)$ so, dass

$$\begin{aligned} E(z) \quad &= \quad \max \\ &\text{mit} \quad h\left(v(z)\right) \leq 0, \text{ sonst beliebig} \end{aligned} \tag{7.15}$$

Anhang K leitet detailliert die Lösung von Gleichung 7.15 her. Für

$$\begin{aligned} v(z) \quad &= \quad 0 \quad \text{(Lösung gemäß Angang K)} \\ \text{respektive } K_p \quad &= \quad 0 \quad \text{(mit Gleichung 7.13)} \\ \text{ist } E(z) \quad &= \quad \frac{B_k}{(z-1)^2} \quad \text{maximal} \end{aligned}$$

Entsprechend ist in der Ruhephase $\Delta_{t^{\text{inakt.}}} V(z) = 0$ das betragsmäßig minimale Stellwertsignal. Das Ergebnis hat bedeutsame Konsequenzen, die in den folgenden Punkten erläutert werden.

- $v(z) = 0$ respektive $K_p = 0$ ist die *einzige* Lösung für das Optimierungsproblem 7.15.
- Die zentrale Erkenntnis ist: $K_p = 0$ ist zwingend und keine isolierte Entwurfsentscheidung.
- $E(w) > 0$ wird in der Dauer einer Ruhephase kompensiert. (Unter Berücksichtigung der minimalen Ansprechbarkeit ist die maximale Zeitdauer einer Ruhephase $t^{\text{rahm.}} - R^{\min}$.)
- Die Kompensation erfolgt mit der Dynamik und Geschwindigkeit der Betriebsdauersollfunktion, d.h. $V_k(z) = 0$.
- Das Signal der Sollfunktion existiert auch bei abgeschaltetem Rechnersystem, da die Betriebsdauer eine globale Zeitlinie außerhalb des Rechnersystems darstellt. Es ist das einzige Signal, für das das möglich ist.
- Die Gleichungen des PControl Regelkreises beschreiben die Ruhephase konsistent zum realen Ablauf des Rechnersystems.

PControl operiert in der Ruhephase *maximal schnell*. Im w-ten Duty Cycle ist nach $t_w^{\text{inakt.}}$ Zeiteinheiten gemäß Gleichung 7.8 der Fehler kompensiert und das Spannungsbudget erreicht die Sollfunktion $V_k^{\text{soll}}(z)$. Nachdem $E(w) \leq 0$ erreicht wurde, schaltet der Gain Scheduler auf $K_p = 1$. Es wird das Führungsverhalten in die Arbeitsphase aktiviert. Hin- und Herschalten zwischen den Phasen, das sogenannte Flattern, wird verhindert, wenn für $E(w) = 0$ festgelegt wird, dass sich das System in der Arbeitsphase befindet.

7.6.3 Eigenschaften der Prozesskooperation durch PControl

Durch maximieren respektive minimieren der Stellgröße $\Delta_{t^{\text{akt./inakt.}}} V(z)$ nähert sich das PControl geregelte Spannungsbudget sowohl in der Arbeitsphase wie auch in der Ruhephase der Betriebsdauersollfunktion V_k^{soll} mit höchster Geschwindigkeit. PControl realisiert eine *zeitoptimale* Regelung.

Die Eigenschaften der PControl geregelten kooperativen Prozessorganisation sind in der folgenden Tabelle 7.6 zusammengefasst.

Eigenschaft	**Systemverhalten**	**Interpretation**
Stabilität		
Arbeitsphase	$\forall K_p > 0$, insbesondere für $K_p = 1$ (optimale Wahl), liegen die Polstellen im Einheitskreis und der Regelkreis $F(z)$ ist stabil	Führungsverhalten bezüglich der Betriebsdauersollfunktion V_k^{soll}
Ruhephase	für $K_p = 0$ ist der Regelkreis $F(z) = 0$ konstant und somit stabil	PControl bildet das Spannungsverhalten bei ausgeschaltetem Rechnersystem nach
Schnelligkeit		
Arbeitsphase	konst. Störeinfluss D nach $w \approx \frac{\ln 0.02}{\ln \frac{1}{K_p+1}} = 6$ Duty Cycle Schritten bis auf 2% kompensiert	PControl operiert für $K_p = 1$ *maximal schnell*.
Ruhephase	mit $K_p = 0$ ist Fehler $E(w) > 0$ in $t_w^{\text{inakt.}} = -\frac{1}{B_k} E(w)$ Zeiteinheiten (siehe Gleichung 7.8) kompensiert	PControl operiert für $K_p = 0$ *maximal schnell*.
Genauigkeit		
Arbeitsphase	es existiert eine stationäre Regelabweichung $-\frac{B_k}{K_p} = -B_k > 0 (B_k < 0)$ von Betriebsdauersollfunktion V_k^{soll}	der ständigen Veränderung des Sollwertes kann mit gleicher Dynamik (wg. Stabilität), aber absolut bis auf den stationären Fehler $-B_k$ gefolgt werden
Ruhephase	ist $t_w^{\text{inakt.}}$ nicht ganzzahlig, dann existiert eine beschränkte Abweichung $B_k < E(t_w^{\text{inakt.}}) < 0$ von der Sollfunktion, sonst $E(t_w^{\text{inakt.}}) = 0$	$E(w) > 0$ wird *immer* vollständig kompensiert

Tabelle 7.6: Eigenschaften der PControl geregelten kooperativen Prozessorganisation für das Energiemanagement.

7.7 Optimalität von PControl

Im vorangegangenen Abschnitt wurde gezeigt, dass PControl zeitoptimal ist. In der Problembeschreibung des Energiemanagements in Abschnitt 7.3 wird gefordert, dass

PControl eine vorgegebene Betriebsdauer bestmöglich erreicht. Die Abweichung respektive der Regelungsfehler zwischen Betriebsdauersollfunktion und Spannungsbudget muss minimal werden. Dies kommt in folgendem Theorem zum Ausdruck:

Theorem 7.1. *Der PControl Regler minimiert die 1-Norm der Regeldifferenzen im PControl Regelkreis.*

Beweis. Es ist zu zeigen, dass PControl die Betragsregelfläche [165]

$$J_k(t) = \sum_{t=0}^{t^{\text{rahm.}}} |e_k(t)|$$

minimiert. Dabei bezeichnet $e_k(t)$ den Regelungsfehler zum Zeitpunkt $t \in \left[0, t^{\text{rahm.}}\right]$ im k-ten Zeitrahmen.

Während der Arbeitsphase arbeitet PControl schnellstmöglich und das Rechnersystem am physikalischen Limit. Mit $K_p = 1$ ist $\Delta_{t^{\text{akt.}}} V(z) = \frac{V^{\text{batt}}}{(z-1)^2}$. Das zeitoptimale Verhalten von PControl lässt somit das Spannungsbudget $V_k(z)$ dem *unbekannten* Störeinfluss des Batterie- und Prozessverhaltens folgen. In der Ruhephase bleibt das Spannungsbudget wie auch die Batteriespannung konstant.

Dieses Einflussverhalten des Batterie- und Prozessverhaltens auf die Regelstrecke erlaubt keine Prognose über den zukünftigen Verlauf des Spannungsbudgets. Als Konsequenz berechnet sich die Betragsregelfläche zu

$$\begin{aligned}
\min_{t=0\ldots t^{\text{rahm.}}} J_k(t) &= \\
&= \min_{t=0\ldots t^{\text{rahm.}}} \sum_{t=0}^{t^{\text{rahm.}}} |e_k(t)| \\
&= \min_{t=0\ldots t^{\text{rahm.}}} \sum_{t=0}^{t^{\text{rahm.}}} \left|V_k^{\text{soll}}(t) - V_k(t)\right| \\
&= \min_{t=0} \left|V_k^{\text{soll}}(t) - V_k(t)\right| \ldots \min_{t=t^{\text{rahm.}}} \left|V_k^{\text{soll}}(t) - V_k(t)\right| \qquad (7.16)
\end{aligned}$$

Der zeitoptimale Regler realisiert das Minimum für jeden Einzelterm in Gleichung 7.16, da er in Arbeits- und Ruhephase immer die maximale respektive minimale Stellgröße anlegt, um den Regelungsfehler mit maximaler Geschwindigkeit zu kompensieren. Die Behauptung folgt sofort. □

Das PControl geregelte Duty Cycling organisiert die Ausführung *aller Prozesse* so, dass eine gemeinsame, vorgegebene Betriebsdauer *bestmöglich* erreicht wird. Die Zeitrahmenstruktur garantiert eine Mindestansprechbarkeit des Rechnersystems. Kooperation durch PControl ist eine Lösung der Problemstellung des Energiemanagements ubiquitärer Rechnersysteme aus Abschnitt 7.3.

7.8 Simulationsstudien

Die Simulationsstudien illustrieren die PControl geregelte kooperative Prozessorganisation für das Energiemanagement ubiquitärer Rechnersysteme. Es wird das

Verhalten während der Arbeitsphase unter unterschiedlichen Störungen aufgrund der Batterieeigenschaften bei Prozessausführung veranschaulicht, sowie das Verhalten während der Ruhephase zur Kompensation bei Abweichung von der Betriebsdauersollfunktion. Für die Simulation wird das Simulinkmodell aus Abbildung 7.10 ausgeführt.

7.8.1 Batterieverhalten bei Prozessausführung

Das Modell in Abbildung 7.11 simuliert des Spannungsverhalten einer Batterie bei Prozessausführung. Konkret wird das System $V^{\text{batt}}\frac{1}{(z-1)^2}$ simuliert. V^{batt} ist ein

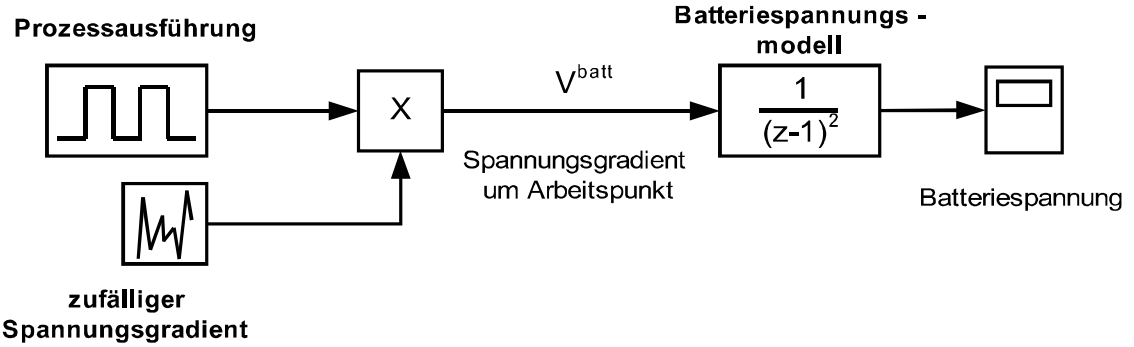


Abbildung 7.11: Simulinkmodell des Spannungsverhaltens einer Batterie bei Prozessausführung. Prozesse erzeugen impulsförmige Amplituden V^{batt}, die als Gradient im Batteriespannungsmodell $\frac{1}{(z-1)^2}$ wirken.

sich unbekannt, hier zufällig, änderndes Eingangssignal, das den Spannungsgradient durch Prozessausführung um einen Arbeitspunkt wiedergibt. Der Arbeitspunkt wird als der durchschnittliche Spannungsgradient verstanden. Positive respektive negative Abweichungen vom Arbeitspunkt demonstrieren größere respektive kleinere Spannungsgradienten. Die Amplituden der Gradienten müssen so gewählt werden, dass der Spannungsverlauf monoton fallend ist. Es wird ein instabiles System zweiter Ordnung gewählt, dass nach Impulsanregung divergiert, da neben den Prozessen die Komponenten des Rechnersystems eine Grundlast mit entsprechendem Spannungsgradienten erzeugen. Weitere sekundäre Effekte, zum Beispiel der Recovery-Effekt, werden gemäß der Betrachtungen in Abschnitt 7.2.3 nicht betrachtet. Die Ergebnisse der Simulation sind in Abbildung 7.12 zusammengefasst. Der Zeitverlauf der Spannung spiegelt ein realitätsnahes Verhalten wider und ist bis auf die Skalierung vergleichbar mit experimentellen Messungen der Betriebsspannung aus Abbildung 7.4. Für Untersuchung des Modells ist Skalierungstreue nicht relevant, da es sich um ein lineares Modell handelt.

7.8.2 PControl geregelte Prozesskooperation

Zur Veranschaulichung des Verhaltens von PControl werden Arbeits- und Ruhephase mit linearen Anstiegsfunktionen als Batteriespannung diskutiert. Im Anschluss wird das Verhalten über einen typischen Spannungsverlauf simuliert.

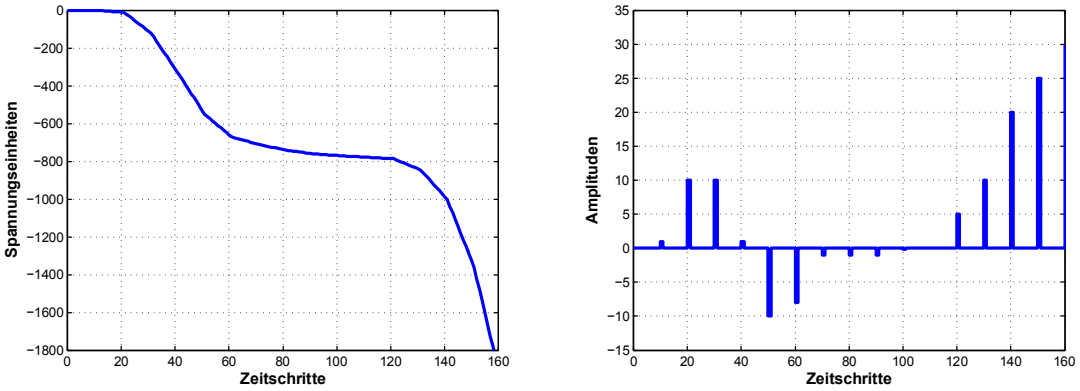


Abbildung 7.12: Links: Erzeugter Spannungsverlauf des Modells aus Abbildung 7.11. Rechts: Amplituden der Spannungsgradienten V^{batt} der Prozesse um den Arbeitspunkt.

Diskussion der Arbeits- und Ruhephase

Die Abbildung 7.13 zeigt Ausschnitte für eine Arbeits- und eine Ruhephase. Es werden Betriebsdauersollfunktion, Batteriespannung und Spannungsbudget im Vergleich zur Systemaktivität dargestellt. Dem PControl Regelkreis ist die Batteriespannung nur durch Messung bekannt und wird entsprechend der Gleichung 7.11 als Störung eingekoppelt. Die Batteriespannung als Störung des PControl Regelkreises

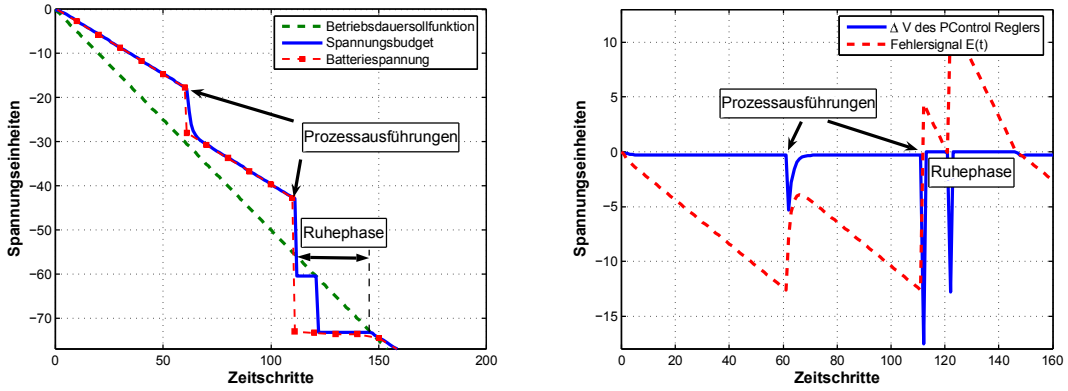


Abbildung 7.13: Ergebnisse der PControl Regelung des Simulinkmodells aus Abbildung 7.10. Links: Prozessausführungen mit Dauer 1 bei $t = \{60; 110\}$. Das Spannungsbudget folgt der Batteriespannung. Ruhephase im Zeitabschnitt $[111 \ldots 147)$ kompensiert den Fehler zur Sollfunktion. Rechts: Fehlerzeitsignal $E(t)$ und erzeugter Spannungsgradient ΔV des PControl Reglers. In der Ruhephase ist $\Delta_{t^{\text{inakt.}}} V = 0$.

ist nicht konstant, sondern ändert sich in jedem Schritt. Daher folgt im linken Diagramm der Abbildung 7.13 das Spannungsbudget der Batteriespannung. Da in der Arbeitphase die Batterie maximal genutzt wird, ist der Abstand zur Sollfunktion immer minimal. Das Budget erfährt gegenüber der Batteriespannung eine Messverzögerung. Dies verursacht das schwächerwerdende Nachlaufen des Spannungsbudgets. Im rechten Diagramm der Abbildung 7.13 ist gut zu erkennen, dass in der Arbeitsphase $\Delta_{t^{\text{akt.}}} V < 0$ maximiert wird, während in der Ruhephase das Fehlersignal $E(t)$ maximal wird und entsprechend schnell mit $\Delta_{t^{\text{inakt.}}} V = 0$ kompensiert wird. Die-

ses Verhalten erfüllt die Optimalitätsziele der PControl geregelten Kooperation aus Abschnitt 7.6.

Die Abbildung 7.14 zeigt die Bestimmung des PControl Reglers und die Zeitpunkte des Umschaltens des Gain Schedulers. In den aktiven Arbeitsphasen ist $K_p = 1$. In der Ruhephase im Zeitabschnitt $[111 \ldots 147)$ bestimmt der Gain Scheduler den Regler zu $K_p^{\text{inakt.}} = 0$. Die Messverzögerung des Spannungsbudgets gegenüber der

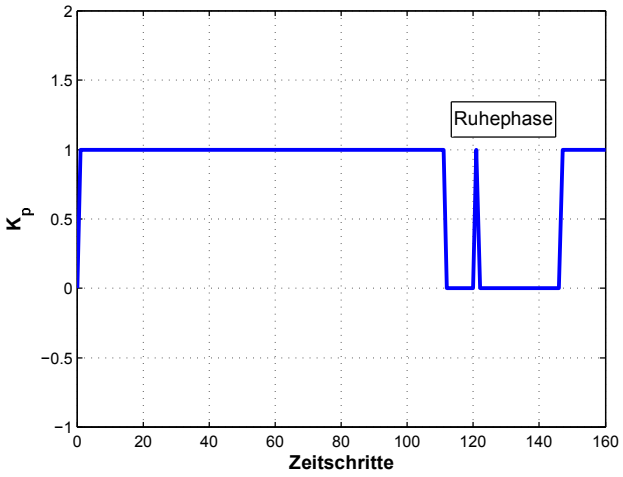


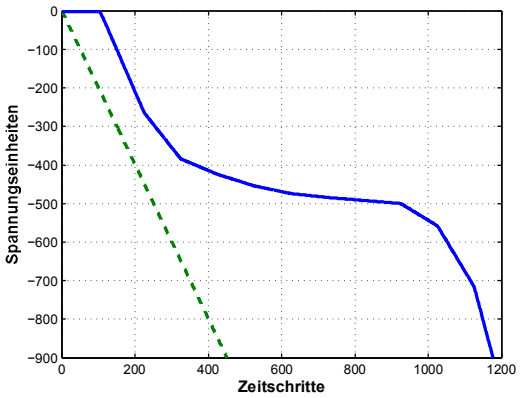
Abbildung 7.14: Umschalten der PControl Reglerverstärkung in der Ruhephase. Das Flattern zum Zeitpunkt $t = 121$ wird durch die Messverzögerung der tatsächlichen Batteriespannung verursacht.

tatsächlichen Batteriespannung lässt das Budget die Sollfunktion vorzeitig erreichen. Der Gain Scheduler aktiviert zum Zeitpunkt $t = 121$ das Rechnersystem wieder, um es sofort wieder zu deaktvieren. Das Flattern kann in einer Implementierung verhindert werden, indem vor dem Abschalten $V_k(t_w^{\text{inakt.}} = 0) = V^{\text{batt}}$ bestimmt wird.

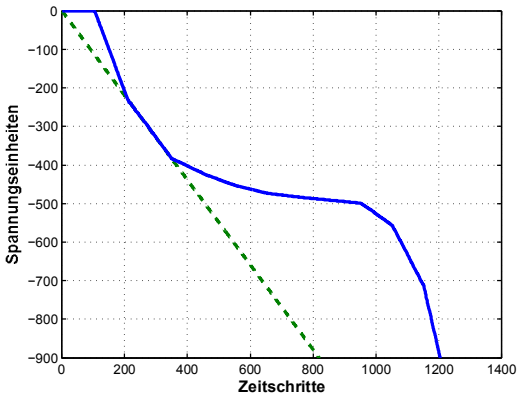
Simulation für einen typischen Spannungsverlauf

Zum Abschluss der Studien über das Verhalten der PControl geregelten Kooperation wird eine typische Spannungscharakteristik einer Batterie in ihrer Gesamtheit betrachtet. Die Untersuchung bietet einen Vorausblick auf das experimentell zu beobachtende Verhalten.

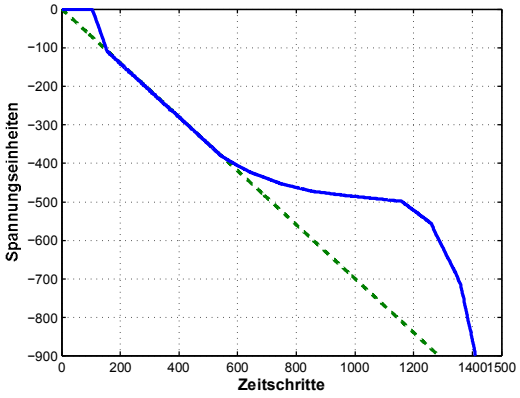
Es wird die simulierte Spannungscharakteristik aus Abbildung 7.12 als Störsignal in den PControl Regelkreis eingekoppelt. Die Cut-off Voltage beträgt $V_{\text{cut}} = -900$ Spannungseinheiten. Abbildung 7.15 stellt den Einfluss verschieden steiler Betriebsdauersollfunktionen und die damit verbundene Betriebsdauer dar. Arbeits- und Ruhephasen des Rechnersystems bilden größere Strukturen. So gibt es Bereiche, in denen PControl durch wechselnde Arbeits- und Ruhephasen die Spannungskurve entlang der Sollfunktion linearisiert. Die Linearisierung umfasst umso größere Bereiche, je flacher die Betriebsdauersollfunktion ist. Einhergehend ist damit auch eine Verlängerung der Betriebsdauer von anfänglich $t^{\text{betr.}} = 450$ bis $t^{\text{betr.}} = 1800$ zu beobachten. Idealerweise setzt PControl die Veränderung der Sollfunktionssteigung in die Veränderung der Betriebsdauer um.



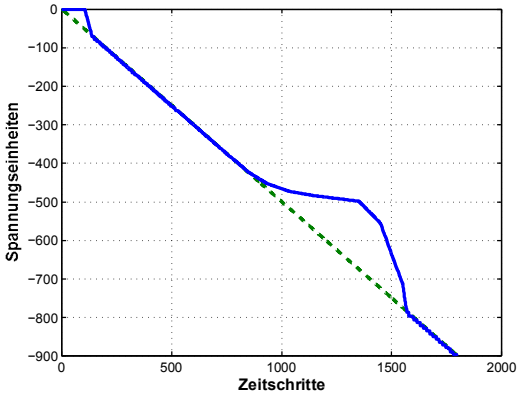
(a) $B_k = -2, t^{\text{betr.}} = 450$



(b) $B_k = -1.1, t^{\text{betr.}} = 819$



(c) $B_k = -0.7, t^{\text{betr.}} = 1287$



(d) $B_k = -0.5, t^{\text{betr.}} = 1800$

Abbildung 7.15: Spannungsbudget und Betriebsdauersollfunktionen von PControl Simulationen unter der Batteriespannungscharakteristik aus Abbildung 7.12. Die Fallbeispiele (a)-(d) durchlaufen immer flacher werdende Steigungen Bk. Durch Linearisierung des Spannungsbudgets entlang dieser Funktionen wird die Betriebsdauer $t^{\text{betr.}}$ bis zum Erreichen der Cut-off Voltage bei $V_{\text{cut}} = -900$ gedehnt. Idealerweise setzt PControl die Veränderung der Sollsteigung in die Veränderung der Betriebsdauer um.

Flacht die Spannungscharakteristik der Batterie ab, so sind Ruhephasen unnötig. In den Simulationen liegt dieser Bereich immer zwischen $V_k(t) = -400 \ldots -500$ Spannungseinheiten. Das Rechnersystem ist dann dauerhaft aktiv. NiMH Batterien weisen zum Beispiel langanhaltende flache Spannungscharakteristiken auf. Prozesse in PControl geregelten Ausführungsumgebungen erreichen in diesem Abschnitt eine hohe Ressourcennutzung und geringe Ausführungsverzögerung.

Theorem 7.1 weist nach, dass dieses Verhalten bei unbekannter Prozessausführung und Spannungscharakteristik eine vorgegebene Betriebsdauer bestmöglich erreicht.

7.8.3 Bestimmung der Betriebsdauersollfunktion $V_k^{\text{soll}}(z)$

Die Betriebsdauersollfunktion ist eine lineare Anstiegsfunktion, die Betriebsdauer, Cut-off Voltage und Spannungsbudget in einen idealen Zusammenhang setzt. In der Gleichung 7.10 dieser Funktion kodiert der Parameter $B_k < 0$ mit

$$B_k = \frac{V_k(0) - V_{\text{cut}}}{t - t^{\text{betr.}}}$$

diesen Zusammenhang als Gradient von $V_k^{\text{soll}}(z)$ im k-ten Zeitrahmen. Es ist sinnvoll, $V_k(0)$ mit dem Wert der gemessenen Versorgungsspannung des Rechnersystems zu initialisieren. Zwei Ansätze werden betrachtet:

1. $V_k(0) = V^{\text{batt}}(t = 0), k = 1$, d.h. die Betriebsdauersollfunktion wird beim Start des Rechnersystems festgelegt. Für alle Zeitrahmen k ist $B_k = \text{konst.}$.
2. $V_k(0) = V^{\text{batt}}\left(t = (k-1)t^{\text{rahm.}}\right), k > 1$, d.h. zu Beginn jedes Zeitrahmens k wird die Betriebsdauersollfunktion gemäß der aktuellen Versorgungsspannung des Rechnersystems nachgestellt.

Der Unterschied zeigt sich deutlich in Abbildung 7.16. Anstatt wie im linken Teil

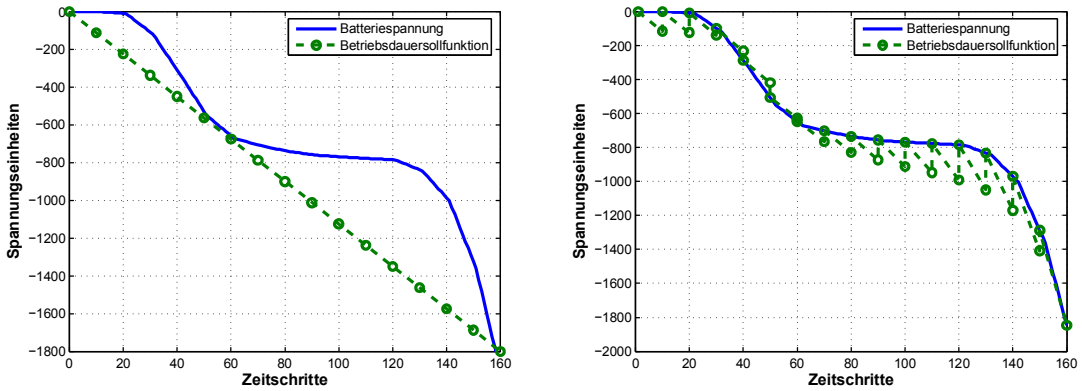


Abbildung 7.16: Betriebsdauersollfunktionen für $t^{\text{betr.}} = 160, t^{\text{rahm.}} = 10$. Links: Sollfunktion mit zu Beginn festgelegtem und unveränderlichen B_k (Ansatz 1). Rechts: Stückweise Linearisierung über Nachstellen von $V_k(0)$ zu Beginn jedes Zeitrahmens (Ansatz 2). Sollfunktion passt sich dem tatsächlichen Verlauf der Batterieversorgungsspannung besser an.

der Abbildung über alle k Zeitrahmen konstant zu linearisieren, realisiert die zweite Methode eine stückweise Linearisierung über die Dauer eines Zeitrahmens $t^{\text{rahm.}}$.

Vorteil ist eine Berücksichtigung der tatsächlichen Spannungscharakteristik der eingesetzten Batterie. Lange Phasen mit geringer Änderungscharakteristik wie sie für viele Batterietypen typisch sind, können durch die Betriebsdauersollfunktion akkurater erfasst werden. Kurzfristige und besonders ausgeprägte Spannungseigenheiten einer Batterie werden jedoch pessimistisch bewertet. Da erstere Charakterisierung in der Praxis häufig zu finden ist, wird der stückweisen Linearisierung der Vorzug gegeben.

Bei der stückweisen Linearisierung muss die verstrichene Betriebsdauer vom Rechnersystem erfasst und mitgeführt werden. Zusätzlicher Rechenaufwand ist notwendig, um B_k zu bestimmen. Beide Ansätze ändern nichts am optimalen Verhalten der PControl Regelung. Es bleibt $V_k^{\text{soll}}(z) - V_k(z) = \min$ erhalten.

7.9 Implementierung

Die PControl geregelte Prozessorganisation wurde auf der Particle Computer Plattform implementiert. Sie umfasst die PControl Regelung, die Messung der Batteriespannung zur Bestimmung des Störeinflusses und die Aktivierung der Ruhephase für die Dauer von $t_w^{\text{inakt.}}$. Zum Abschalten in den Ruhephasen wird das Particle Computer Rechnersystem in einen Power Down Modus versetzt. In diesem Modus sind alle peripheren Komponenten ausgeschaltet und im Mikrocontroller läuft nur noch der Zeitgeber. Die Leistungsaufnahme beträgt dann nur 1.32 μW.

Die PControl Regelung wurde in der vom Autor dieser Arbeit betreuten Studienarbeit von Steffen Melischko [166] implementiert. Es sind Zeitrahmenlänge, minimale Ansprechbarkeit und die Betriebsdauersollfunktion gemäß Gleichung 7.10 anzugeben. Zur Laufzeit wird mittels Spannungsmessung der Störeinfluss und der Regelungsfehler $E(w)$ bestimmt und gemäß Gleichung 7.8 das System für die Dauer von $t_w^{\text{inakt.}}$ in die Ruhephase versetzt. Die Laufzeitbibliothek (API) des Particle OS aus Abschnitt 4.6 unterstützt die Interaktion mit der Echtzeituhr des Rechnersystems.

Die Software wurde mit dem Small Devices C Compiler (SDCC) 2.6.1 [167] übersetzt. Die Rechenroutinen für die Gleichungen benötigt die Datentypen, die große Wertebereiche abbilden können, um eine ausreichende Genauigkeit der Berechnung zu ermöglichen. Der Gradient B_k der Sollfunktion liegt im Bereich von $-10^{-3}\left[\frac{\text{mV}}{\text{s}}\right]$. Der SDCC unterstützt 16 bit und 32 bit (long) Ganzzahlen wie auch 4 Byte Fließkommazahlen (IEEE Format mit einfacher Genauigkeit). Damit lassen sich Fest- und Fließkommaarithmetik realisieren. Die Speicherbelegung der PControl Implementierung ist in Tabelle 7.7 angegeben. RAM bezieht sich auf den Speicheraufwand für Variablen, ROM auf Berechnungsfunktionen von PControl.

	RAM[bytes]	**ROM**[bytes]
Fliesskomma		
Initialisierung	40	588
Laufzeit	57	780
Ganzzahl, 16 bit und 32 bit		
Initialisierung	34	652
Laufzeit	42	913

Tabelle 7.7: Speicherbelegung der PControl Implementierung. (Quelle: [166])

Die Ganzzahlimplementierung weist eine kleinere RAM Belegung auf, jedoch eine größere Belegung des ROM. Da die Particle Computer Plattform wie auch vergleichbaren typischen Geräteplattformen ubiquitärer Rechnersysteme wesentlich weniger RAM als ROM Speicher zur Verfügung steht, ist die Ganzzahlimplementierung zu bevorzugen.

7.10 Experimentelle Untersuchungen

Die Experimente wurden im Rahmen der vom Autor dieser Arbeit betreuten Studienarbeit von Steffen Melischko [166] durchgeführt. Den Untersuchungsgegenstand bildet die in Abbildung 7.17 gezeigte Bauvariante der Particle Computer Plattform. Sie wurde für den Einsatz in CoBIs Szenarien für die verteilte Gefahrendetektion in Abschnitt 6.1 vorgestellt. Regelmäßige Wartezyklen der Rechnersysteme zum Wech-

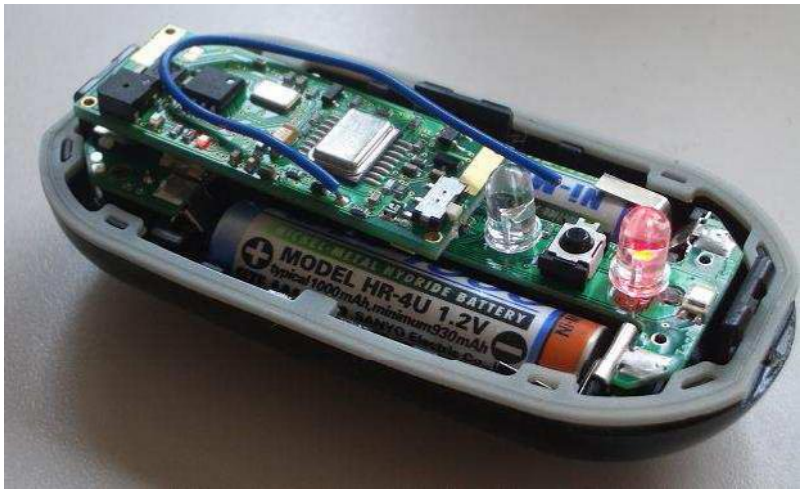


Abbildung 7.17: CoBIs Bauvariante der Particle Computer Plattform für die experimentellen Untersuchungen der PControl Regelung. (Quelle: [166])

sel der batteriebasierten Energieversorgung motivieren den Einsatz der PControl Regelung. Ziel dabei ist es, den gegebenen Wartungszeitpunkt zu erreichen. Versuchsaufbau und Ergebnisse sind in den folgenden Unterabschnitten erläutert.

7.10.1 Versuchsaufbau

Die experimentellen Versuche verfolgen einen hardware-in-the-loop Ansatz. Dabei wird die Particle Computer Plattform mit dem zyklengenauen PIC Hardwareemulator aus der Diplomarbeit von Yusuf Iskenderoglu [85] verbunden und der Code für die PControl Regelung aufgeteilt.

Der Particle Computer wird mit zwei 1.5 Volt (V) AAA NiMH Batterien betrieben und führt Prozesse aus. Der Particle Emulator auf einem Desktop System führt den für die Particle Plattform kompilierten Code der PControl Regelung aus. Dies erlaubt eine genaue Überwachung der Regelung und das Mitprotokollieren der Messwerte wie auch des Spannungsbudgets. Der Versuchsaufbau ist in Abbildung 7.18 dargestellt. Auf dem Particle Computer wird periodisch ein Prozess mit einer zufälligen, gleichförmig verteilten Ausführungsdauer ausgeführt. Der Erwartungswert beträgt 100 Millisekunden (ms). Der Prozess schaltet während der

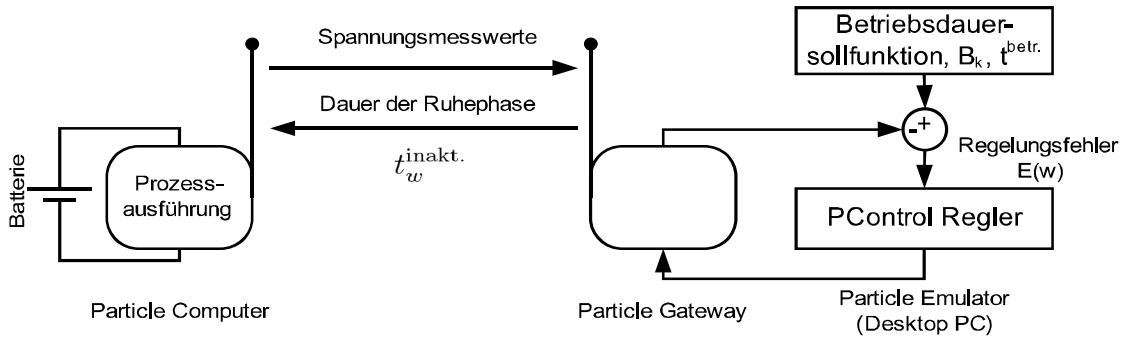


Abbildung 7.18: Versuchsaufbau für die Evaluierung der PControl Regelung mit der Particle Computer Plattform.

Ausführung eine externe Leuchtdiode mit einer Leistungsaufnahme von 100 Milliwatt (mW), die die Batterie belastet. Der gemessene Spannungsgradient wird drahtlos über ein Gateway an den Particle Simulator übertragen. Der Spannungsgradient, der durch die Leistungsaufnahme der Kommunikationsschnittstelle verursacht wurde, fließt bei der Messung in der darauffolgenden Periode ein.

Der Emulator nimmt die an den Gateway übertragenen Spannungsmesswerte des Particle Computers auf. Sie werden in die Implementierung des PControl Reglers eingekoppelt. Mit jeder Übertragung wird ein Duty Cycle identifiziert. Mittels Gleichung 7.8 kann der PControl Regler aus dem Regelungsfehler $E(w)$ die Dauer $t_w^{\text{inakt.}}$ der Ruhephase des w-ten Duty Cycle bestimmen.

Prozess- und Kommunikationsverhalten des Particle Computer bilden ein unbekanntes Verhalten der Batteriespannung ab. Nach jeder Prozessausführung entscheidet die PControl Regelung gemäß der Sollfunktion über die weitere Prozessausführung. Bleibt das Rechnersystem weiterhin in der aktiven Arbeitsphase, kann die Prozessausführung mit der nächsten Periode fortgesetzt werden. Bei abgeschaltetem Rechnersystem in der Ruhephase wird kein Prozess ausgeführt, auch dann nicht, wenn seine Periode erneut beginnt.

7.10.2 Durchführung, Ergebnisse und Bewertung

Es wird sowohl das Verhalten bei konstant linearer wie auch bei stückweiser linearer Betriebsdauersollfunktion untersucht. Die beobachtete Größe ist die Batteriespannung des Particle Computer Systems. PControl ist erfolgreich, wenn nach Ablauf der Betriebsdauer die Batteriespannung größer der Cut-off Voltage ist. PControl wurde mit folgenden Parametern initialisiert.

$t^{\text{rahm.}}$	500 Sekunden
$R^{\min}$	1 Prozess pro $t^{\text{rahm.}}$
V_{cut}	600 Millivolt
$t^{\text{betr.}}$	120000 Sekunden

Alle Versuche wurden mehrmals durchgeführt, Dabei wurden jeweils andere NiMH Batterien verwendet, um spezifische Effekte zu vermeiden. Die Ergebnisdiagramme sind exemplarisch herausgegriffen. Die gemessenen Spannungswerte wurden mit einem gleitenden Durchschnitt gefiltert, um Rauschen zu eliminieren. Jedoch wurden alle aufgetretenen Ruhephasen berücksichtigt. Die Rauschfilterung führt somit nicht zu inkonsistenten Darstellungen.

Konstant lineare Betriebsdauersollfunktion

Zum Vergleich wurde eine Spannungskurve über den Zeitverlauf ohne Eingriff der PControl Regelung aufgenommen. Der geregelte und ungeregelte Spannungsverlauf bei konstant linearer Sollfunktion wird in Abbildung 7.19 gezeigt. Das Einfügen von

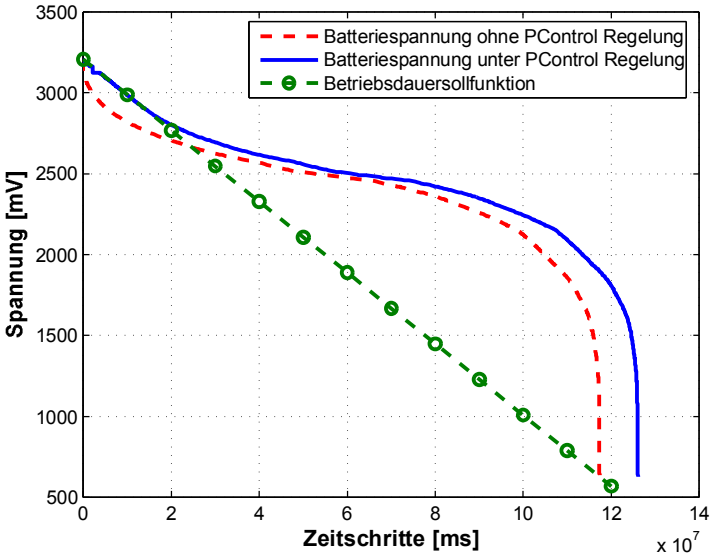


Abbildung 7.19: Die konstante Sollfunktion dehnt den Spannungsverlauf durch Linearisierung am Beginn der Kurve. Der Spannungsverlauf ohne PControl Regelung dient als Referenz.

Ruhephasen aufgrund der Linearisierung zu Beginn des Spannungsverlaufs verlängert die Betriebsdauer. Das experimentelle Ergebnis bestätigt die Simulationsergebnisse in der Abbildung 7.15.

Stückweise linearisierte Betriebsdauersollfunktion

Wie in Abschnitt 7.8.3 vorgeschlagen, wird die Sollfunktion stückweise über die Dauer $t^{\text{rahm.}}$ linearisiert. Die Anpassung wirkt sich somit auf den gesamten Verlauf aus. Um den typischen Verlauf besser zu erfassen, wurde die Betriebsdauer gegenüber den vorherigen Experimenten erhöht. Es ist nun

V_{cut}	1100 Millivolt
$t^{\text{betr.}}$	140000 Sekunden

Die experimentellen Ergebnisse sind in Abbildung 7.20 gezeigt. Erläuterungen sind in Tabelle 7.8 zusammengefasst. Die PControl Regelung mit stückweise linearisierter Betriebsdauersollfunktion verlängert die Betriebsdauer des Rechnersystems gegenüber dem ungeregelten Fall. Der Effekt ist wesentlich ausgeprägter als bei konstant linearer Sollfunktion. So ist die Cut-off Voltage wesentlich größer gewählt, d.h. es besteht noch weiteres Potential zum Erreichen längerer Betriebsdauern. Erreicht wird dies durch mehrere kurze Ruhephasen in der letzten Phase des Spannungsverlaufes (Ausschnitt (c)). Der starke Spannungsabfall wird gebremst und der Verlauf zeitlich gedehnt. Die Ergebnisse in Tabelle 7.8 bestätigen die Erwartungen der PControl Simulationen.

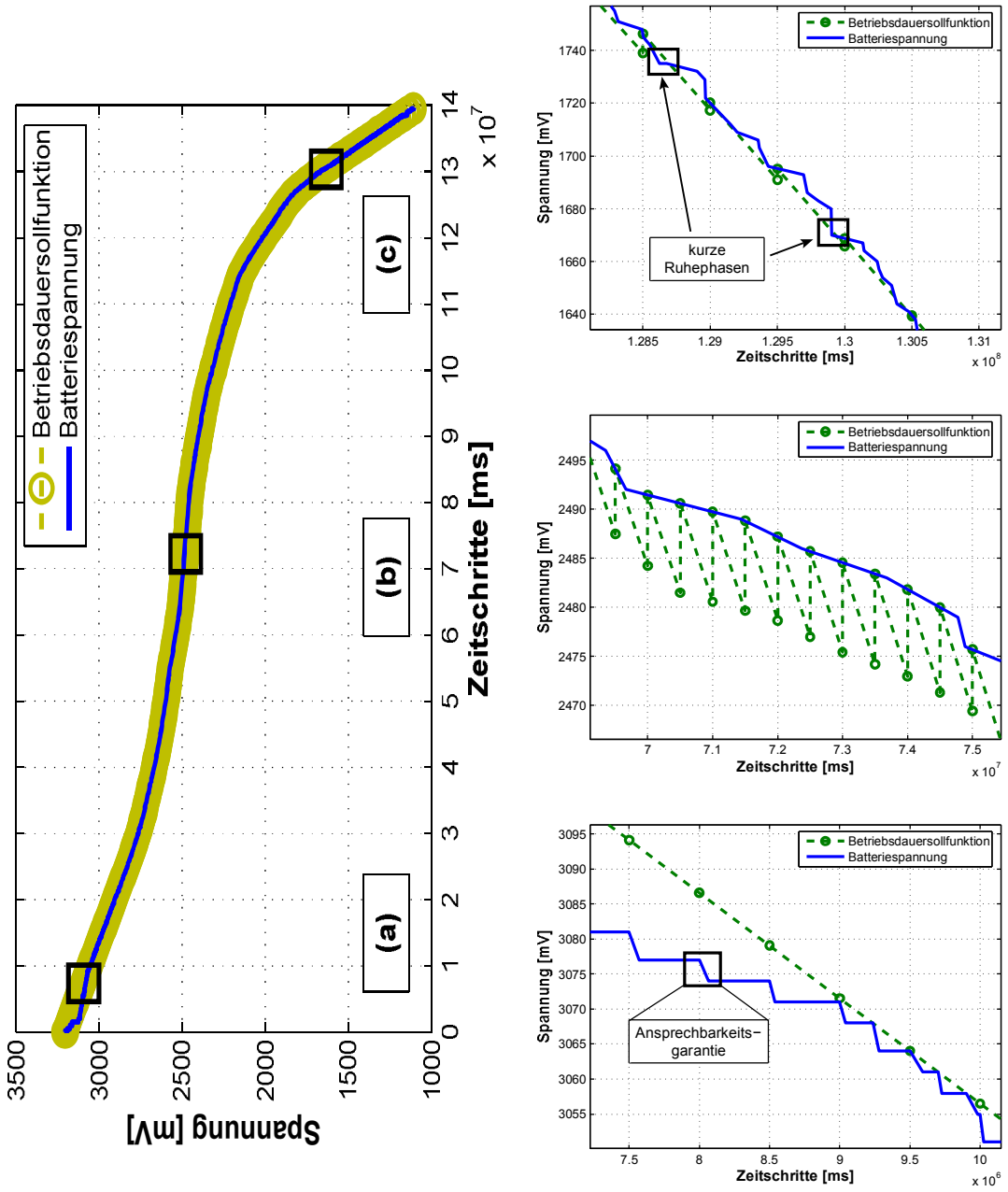


Abbildung 7.20: Links: Spannungsverlauf mit stückweiser linearer Betriebsdauersollfunktion. Die Sollfunktion wird auf dieser Skala nicht deutlicher aufgelöst. Rechts (von unten nach oben): Die vergrößerten Ausschnitte (a), (b) und (c) illustrieren das Verhalten zu Beginn (rechts unten), während (rechts mitte) und am Ende (rechts oben) der Betriebsdauer. Ergebnisse sind in Tabelle 7.8 erläutert.

Ausschnitt	Spannungscharakteristik	Verhalten PControl
(a)	starker Abfall der Batteriespannung, stets unter Sollfunktion.	Arbeitsphase besteht nur aus der Ansprechbarkeitsgarantie, im übrigen Zeitrahmen lange Ruhephase.
(b)	flache Batteriespannungscharakteristik	Sollfunktion wird nicht unterschritten, *kontinuierliche* Arbeitsphase über gesamten Zeitrahmen.
(c)	starke Variation der Batteriespannung	Einfügen kurze Ruhephasen.

Tabelle 7.8: Batterieverhalten und Verhalten der PControl Regelung in den extrahierten Ausschnitten der Abbildung 7.20.

Zusammenfassend lassen sich folgende Ergebnisse aus Theorie und experimenteller Untersuchung formulieren: Die PControl geregelte kooperative Prozessorganisation erreicht bestmöglich eine vorgegebene Betriebsdauer. Sie erzielt eine deutliche Verbesserung gegenüber der ungeregelten Organisation.

7.11 Auswirkungen des Energiemanagements auf die Prozessklassen

Das Abschalten des ubiquitären Rechnersystems für die Dauer $t^{\text{inakt.}}$ verhindert die Ausführung aller Prozesse. Die Auswirkung auf deren Laufzeiteigenschaften werden vom kooperativen Energiemanagement nicht berücksichtigt. Der Umgang mit diesem Nachteil muss diskutiert werden. Es ist in den Vordergrund gestellt und als Einsicht in den kooperativen und kollaborativen Ansatz durch Regelung zu bewerten, dass die Auswirkungen des Duty Cycling *inhärent* behandelt werden.

Der regelkreisbasierte Entwurf der kooperativen und kollaborativen Prozessorganisation trennt zwischen Regelstrecke und Störung. Dadurch stellt er *Orthogonalität* zwischen den Prozessen her. Das bedeutet, die Auswirkungen der Regelung einer Prozessklasse auf andere Prozessklassen wird vom jeweiligen Regelkreis kompensiert. Auf weitere Zusatzannahmen wird verzichtet. Die Kompensation erfolgt somit inhärent. Im folgenden wird dies für die Prozessklassen im Kontext des kooperativen Energiemanagements durch Duty Cycling diskutiert.

Periodische und aperiodische Prozesse

Mögliche Nachteile des Duty Cycling für die periodische und aperiodische Prozessausführung umfassen:

Ausführung von Teilfunktionalitäten. Durch das serviceorientierte Design der Prozesse im Prozessgraphen wird ein Prozess niemals nur zum Teil ausgeführt. In jedem Zeitrahmen kann ein wertvolles Ergebnis erzeugt werden.

Gefahr des Verhungerns von Prozessen durch regelmäßige Ruhephasen des Rechnersystems. Die periodischen Prozessorganisation kann die Ruhephasen

nicht von einer verlängerten Ausführungsdauer eines vorangegangenen Prozesses unterscheiden. Im Regelkreis ist dies eine Störung des Systemausgangs, die durch die Regelung kompensiert wird.

Verminderte Ereignisbehandlung. In den Ruhephasen erfolgt keine Ereignisbehandlung. Die Ansprechbarkeit des Systems ist im schlimmsten Fall auf das wählbare $R^{\min}$ garantiert. Die periodische Behandlung des Ereignispuffers ist garantiert, wenn sie in $R^{\min}$ oder seinem Vielfachen eingeplant wird. Werden alle Serverperioden mit $\forall j : T_j(w+1) = \min_j T_j(w+1)$ identifiziert, wobei $T_j(w+1)$ wie in Gleichung 4.15 angegeben, und wird die Zeitrahmendauer mit $t^{\text{rahm.}} = \frac{1}{n} T_j(w+1),\ n \in \mathbb{N}, n \geq 1$ gewählt, erreicht die aperiodische Prozessorganisation gleiche oder für einzelne Prozesse kleinere Verlustwahrscheinlichkeiten. Diese Behandlung ist nicht mehr fair, aber kein Prozess wird durch das Energiemanagement benachteiligt. Die Auslastungsgrenze U_b muss dafür entsprechend groß gewählt werden. Die Möglichkeiten sind ausgeschöpft, wenn sich $U_b > 1$ ergibt. Der Entwickler muss dann größere Verlustwahrscheinlichkeiten akzeptieren.

Die kooperativen und kollaborativen Organisationsmechanismen für periodische und aperiodische Prozesse können die Auswirkungen des Energiemanagements durch Duty Cycling ohne Nachteile für die Prozesse behandeln.

Echtzeitprozesse

Die Echtzeitbedingungen bleiben garantiert erhalten, wenn die Periode der Echtzeitservices auf die Zeitrahmendauer festgesetzt wird. Die Modellierung des BFS Regelkreises in Abschnitt 5.5.1 setzt maximale Prozessorauslastung $U = 1$ voraus. Eine Formulierung für $U < 1$ ist möglich, jedoch ist der Duty Cycle unbekannt und veränderlich. Ist die Batteriecharakteristik phasenweise bekannt, so kann das Modell angenähert werden. Im allgemeinen Fall kann jedoch kein kollaboratives Verhalten zwischen Echtzeitservices und datengetriebenen Services etabliert werden.

Verteilte Prozesse

Die kooperative Energieregelung der verteilten Prozesse wird durch das kooperative Energiemanagement mit Duty Cycling ersetzt. Alle Stationen erhalten die gleiche Betriebsdauer vorgegeben. Wird in $R^{\min}$ von PControl die regelmäßige Ausführung des FCUP-MAC Regelkreises priorisiert, dann erhält die kollaborative Organisation verteilter Prozesse den Verbund kommunizierender Stationen auch unter dem wechselnden Duty Cycle des Energiemanagements. Alternativ können mittels einer Reglersynthese zur Laufzeit wie in Abschnitt 6.11.1 vorgeschlagen, durch Ruhephasen gedehnte Kommunikationsperioden berücksichtigt werden.

7.12 Was wurde erreicht?

Dieses Kapitel zeigte, wie die in Kapitel 3 eingeführten rückgekoppelten Regelkreise zur Organisation von Prozessen erfolgreich für die Betriebsdauerregelung ubiquitärer Rechnersysteme eingesetzt werden können. Ziel ist, eine vorgegebene Betriebsdauer ohne internes Prozesswissen in unbekannten Einsatzumgebungen bestmöglich zu erreichen.

In Abschnitt 7.4 wurde mit PControl ein Verfahren vorgestellt, das die Prozessausführung kooperativ in aktive Arbeits- und inaktive Ruhephasen organisiert. Unter Einbeziehung der Batterieversorgungsspannung regelt dieser Duty Cycle Mechanismus die Nutzung der Batterie zum Erreichen einer gemeinsamen Betriebsdauer. Das regelmäßige Einfügen einer minimalen Arbeitsphase garantiert eine minimale Systemleistung respektive Ansprechbarkeit des Rechnersystems unter allen Einsatzbedingungen. Mit diesen technischen Merkmalen erschließt PControl ein sehr breites Spektrum an eingebetteten Systemen für ubiquitäre Rechnerumgebungen.

Mit dem PControl Regelkreismodell in Abschnitt 7.5 ist es gelungen, die Zusammenhänge zwischen Prozessausführungen, Energievorrat und gewünschter Betriebsdauer zu formulieren. Ein Energieprofil der Prozesse oder ein Modell der Batteriekapazität sind für die Anwendung des Regelkreises nicht notwendig. Aus diesen Gründen ist PControl für das Energiemanagement von batteriebetriebenen ubiquitären Rechnersystemen in unbekannten Einsatzumgebungen geeignet.

Theorie, Simulation und Experiment mit der Implementierung in Abschnitt 7.9 befinden sich in konsistenter Übereinstimmung. Langanhaltende, flache Spannungscharakteristiken von Batterien, zum Beispiel NiMH Batterien, werden von PControl besonders nutzbar gemacht. PControl geregelte Prozesse erreichen in diesem Abschnitt eine hohe Ressourcennutzung und geringe Ausführungsverzögerung. Der theoretische Aufwand dieses Kapitels führt in Abschnitt 7.7 schließlich zum Optimalitätstheorem (Theorem 7.1). Es zeigt, dass die PControl geregelte kooperative Prozessorganisation eine *vorgegebene Betriebsdauer bestmöglich erreicht*. Sie erzielt eine deutliche Verbesserung gegenüber der ungeregelten Organisation.

Es ist in den Vordergrund gestellt und als Einsicht in den kooperativen und kollaborativen Ansatz durch Regelung zu bewerten, dass die Auswirkungen des Duty Cycling auf die Prozessklassen inhärent behandelt werden. Die kooperative und kollaborative Ausführung periodischer, aperiodischer und verteilter Prozesse bleibt ohne weitere Zusatzannahmen auch unter Duty Cycling erhalten. Kollaboration in Gegenwart von Echtzeitservices kann jedoch nicht unter allen Bedingungen etabliert werden.

8. Zusammenfassung und Ausblick

Eingebettet in die alltägliche Umgebung erbringen ubiquitäre Rechnersysteme unterschiedliche und jeweils spezifische Anwendungsfunktionalitäten (Appliances) und Dienste. Integrierte Sensoren erfassen Umgebungsparameter, die in-situ verarbeitet, ausgewertet und mit anderen Rechnersystemen in der Umgebung drahtlos ausgetauscht werden. Zahlreiche Realisierungen finden sich in den Szenarien von ParcTab [3], AwareOffice [4], AwareHome [5], CoBIs [6] und Remembrance Camera [10]. Darin erbringen ubiquitäre Rechnersysteme Anwendungsfunktionalitäten einzeln und im Verbund miteinander.

Ubiquitäre Rechnersysteme in dieser Arbeit sind ressourcenbeschränkte Miniatursensorsysteme. Zentrale Recheneinheit ist ein Mikrocontroller, der von Peripheriekomponenten wie Sensoren, Speicher und einer drahtlosen Kommunikationsschnittstelle umgeben ist. Prozesse implementieren die Funktionalität der Rechnersysteme als ausführbare Programmteile auf dem Mikrocontroller.

Eine besondere Herausforderung besteht im Umgang mit unbekannten Einsatzbedingungen. Die Anzahl umgebender Systeme, Laufzeiteigenschaften und das Datenaufkommen von informationsverarbeitenden Prozessen sind zur Entwurfszeit nicht oder nur ungenau bekannt und unterliegen während des Betriebs ständiger Veränderung. Diensterbringung und Anwendungsfunktionalitäten sind gefährdet, da Prozesse ungeordnet und unsystematisch ablaufen. Das Verhalten ubiquitärer Rechnerumgebungen ist unverständlich und nicht mehr nachvollziehbar. Die Forderung nach einem geringen Wartungs- und Administrationsaufwand verschärft diese Problematik zusätzlich.

Die vorliegende Arbeit stellt einen neuartigen Ansatz der Prozessorganisation für eingebettete ubiquitäre Rechnersysteme vor. Ziel ist, die Prozessausführung für die Erbringung von Diensten und Anwendungsfunktionalitäten unter unbekannten Ausführungsbedingungen beherrschbar zu halten. Kooperative und kollaborative Organisationsmechanismen erfassen, bewerten und regeln die Prozessausführung, um den Ablauf der Informationsverarbeitung in ubiquitären Rechnerumgebungen zu verbessern. Kooperation und Kollaboration werden auf vier grundlegende Ausprägungen von Prozessen eines Rechnersystems angewendet. Sie unterteilen sich in die Klassen der *periodischen, aperiodischen, echtzeitfähigen und verteilten* Prozesse. Zudem

wird die Verwendung der Energieressourcen des Rechnersystems organisiert. In einem Laufzeitsystem zusammengeführt sind die Mechanismen für alle Prozesse eines ubiquitären Rechnersystems nutzbar. Die These destilliert die gewonnnen Einsichten dieses Ansatzes.

These: Kooperation und Kollaboration sind Mechanismen, die in einem verallgemeinerten Konzept auf alle Prozesse eingebetteter, ubiquitärer Rechnersysteme angewendet werden können. Kooperation und Kollaboration verbessern die Prozessausführung in unbekannten und veränderlichen Einsatzumgebungen. Die Kombination von Kollaboration und Kooperation in einer Kaskade maximiert den Nutzen für ubiquitäre Rechnersysteme.

Diese Behauptung wurde mit den Einzelbeiträgen der Arbeit nachgewiesen. Die Ergebnisse dieser Arbeit tragen dazu bei, die Gestaltung komplexer Appliances für ubiquitäre Rechnerumgebungen zu ermöglichen.

8.1 Einzelbeiträge

Die Analyse ubiquitärer Rechnerumgebungen in Kapitel 2 stellt die *zeitnahe Verarbeitung von Informationen* als grundlegende Anforderung an die Arbeitsweise eines Rechnersystems heraus. Obwohl zunächst axiomatisch festgelegt, zeigt die vorliegende Arbeit, dass damit wichtige Problemstellungen periodischer, aperiodischer, echtzeitfähiger und verteilter Prozessausführung sowie beim Energiemanagement des Rechnersystems zielgerichtet gelöst werden können.

Einsatzumgebungen mit unbekannten und zeitlich veränderlichen Variablen wie Ausführungsdauer, zu verarbeitendes Datenaufkommen und Kommunikationsbeziehungen lassen ein hochkomplexes und schwer verständliches und schwer nachvollziehbares Verhalten von ubiquitären Rechnersystemen entstehen. Die Analyse formuliert das Problem, dass komplexe und verteilte ubiquitäre Rechnerumgebungen ohne die vollständige Erfassung der Ausführungsparameter aller Beteiligten kontrollierbar bleiben sollen.

Dieser Herausforderung soll mit neuen Instrumenten der kooperativen und kollaborativen Organisationsmechanismen der Prozessausführung begegnet werden.

Im Mittelpunkt von Kapitel 3 steht das Rückkopplungsprinzip. In der ersten Hälfte wird ein *neuartiges Verständnis von Kooperation und Kollaboration* als das zielgerichtete Zusammenwirken (= Kooperation) und die zielgerichtete Zusammenarbeit (= Kollaboration) von Prozessen unter Einbeziehung rückgekoppelter Ablaufinformationen geschaffen.

Mit dem *Budget/Kosten Regelkreismodell* wird in der zweiten Hälfte des Kapitels 3 ein mathematisch beschreibbares und in allen Prozessklassen wiederkehrendes Funktionsmuster der rückgekoppelten Prozessorganisation gefunden. Der Entwurf separiert bekanntes Systemverhalten und unbekannte Einflüsse voneinander. Der Regelkreis kompensiert schwankendes Verhalten der Prozessausführung. Eine kompakte Implementierung auf ressourcenbeschränkten Geräten erlaubt ubiquitären Rechnersystemen, ein deterministisches Ausführungsverhalten in unbekannten, zeitlich veränderlichen Einsatzumgebungen zu erreichen.

Zentraler Beitrag des Kapitels 3 ist der Ausbau der Modellierung zur *Theorie der kooperativen und kollaborativen Prozessorganisation* eingebetteter, ubiquitärer Rechnersysteme. Sie ist für periodische, aperiodische, echtzeitfähige und verteilte Prozesse anwendbar und leistet Beiträge für das Energiemanagement. Abschnitt 3.4.4 gibt ein methodisches Vorgehen zur Übertragung auf die Prozessklassen an. Schließlich bedeutet dies, dass Kooperation und Kollaboration ein durchdringendes Konzept für die Datenverarbeitung ubiquitärer Rechnersysteme in unbekannten Einsatzszenarien sind.

In den Kapiteln 4 bis 7 wurde die Theorie der kooperativen und kollaborativen Prozessorganisation auf die vier Prozessklassen und das Energiemanagement angewendet. Neue kooperative und kollaborative Organisationsverfahren wurden entwickelt. Sie wurden in dem *neuartigen Laufzeitsystem Particle OS* zusammengeführt und auf der Particle Computer Plattform experimentell evaluiert. Tabelle 8.1 zeigt die Ergebnisse im Überblick. Entwurf und Modellierung mit dem Budget/Kosten Regelkreismodell macht Prozessklassen voneinander unabhängig. Es genügt nur die Prozesse einer Klasse zu betrachten. Die Auswirkungen, die durch die Ausführung von Prozessen anderer Klassen entstehen, werden einheitlich als unbekannte Störung im Regelkreis modelliert und durch die Regelung behandelt. Diese als *Orthogonalität* bezeichnete Eigenschaft erlaubte es in Kapitel 7 ein schlankes und effektives Energiemanagement zu realisieren, wobei die Ziele der Prozessausführung weiterhin garantiert werden.

8.2 Nachweis der These

Zum Nachweis werden die Aussagen der These mit den wichtigsten Erkenntnissen der Arbeit in Beziehung gesetzt.

Erste Aussage der These: Kooperation und Kollaboration sind Mechanismen, die in einem verallgemeinerten Konzept auf alle Prozesse eingebetteter, ubiquitärer Rechnersysteme angewendet werden können.

Erkenntnis der Arbeit: Trotz der Verschiedenartigkeit der vier Prozessklassen und des Energiemanagements bilden Kooperation und Kollaboration durch das Budget/Kosten Regelkreismodell eine gemeinsame Mechanik zur Regelung des Prozessausführungsverhaltens für alle Prozesse. Dies ist die Kernaussage der Theorie der kooperativen und kollaborativen Prozessausführung aus Kapitel 3. Der Regelungseingriff, d.h. die festgelegte Stellgröße, und Regelungsziel ist für jede Prozessart verschieden. Das Modell ermöglicht jedoch die Synthese eines spezifischen Reglers zur Steuerung des Eingriffs und zum Erreichen der Ziele.

Das hat bedeutende Konsequenzen. Unabhängig von der konkreten Prozessaufgabe können unterschiedliche und gegenseitig unbekannte Anwendungen unter unbekannten und zeitlich veränderlichen Ausführungsbedingungen geordnet miteinander interagieren. Konkret, kann das Ausführungsverhalten komplexer und verteilter ubiquitärer Rechnerumgebungen zielorientiert erfolgen und somit verständlich und nachvollziehbar gestaltet werden.

Bemerkenswert ist, dass die Beschreibung der kooperativen und kollaborativen Organisationsmechanismen als *lineares, zeitinvariantes* Regelkreissystem gelingt. Das

Prozess-klasse	**Verfahren**	**Prozess-organi-sation**	**Ergebnisse**
periodisch	FQS	kooperativ	Prozesse werden *datenechtzeitfähig*, d.h. die Informationsverarbeitung erfolgt zeitnah. Prozesse mit Bezug zum aktuellen Kontext liefern Informationen mit hohem Grad an Aktualität. Realisierung im Laufzeitsystem Particle OS.
		kollaborativ	*Alle* Prozesse verhalten sich datenechzeitfähig ohne vorheriges Wissen über Prozessaufgaben und Laufzeitverhalten. Sie kooperieren *maximal* lang.
aperiodisch	gesteuerte M/D/k/k	kooperativ	Faire Ereignisverarbeitung aller Prozesse ohne Überlast bei beschränkter Systemauslastung und Speichergröße.
echtzeitfähig	BFS	kollaborativ	*Zeitnahe* Informationsverarbeitung zwischen Echtzeit- und datengetrieben Teilen eines Prozesses unter unbekannten Ausführungsbedingungen, Ressourcennutzung ist maximal effektiv.
verteilt	FCUP-MAC	kollaborativ	Stationen kommunizieren im Verbund unter unbekannten Kommunikations- und Netzwerkbedingungen, Latenz: um 27% bis 32% kleiner, Durchsatz: Faktor 8 bis 10 mehr Nachrichten im Vgl. zum unkoordinierten Medienzugriff.
		kooperativ	Alle Stationen egalisieren ihren Energievorrat und erhalten Kollaborationsverhalten *maximal* lange aufrecht.
Energie-management	PControl	kooperativ	Rechnersystem erreicht *bestmöglich* eine vorgegebene Betriebsdauer

Tabelle 8.1: Ergebnisse der kooperativen und kollaborativen Prozessorganisation

macht es nicht nur geschlossen analytisch behandelbar, sondern es lässt sich auch kompakt auf ressourcenbeschränkten Geräten implementieren.

Mit der Theorie der kooperativen und kollaborativen Prozessorganisation ist die erste Aussage der These nachgewiesen.

Zweite Aussage der These: Kooperation und Kollaboration verbessern die Prozessausführung in unbekannten und veränderlichen Einsatzumgebungen.

Erkenntnis der Arbeit: Kooperation und Kollaboration können zur Laufzeit Einflüsse wie unbekanntes Datenaufkommen, Ausführungsdauer und Kommunikationsverhalten zielorientiert behandeln. Im Vordergrund steht dabei die zeitnahe Ausführung. Dabei werden aktuelle Informationen als wichtiger bewertet. Insbesondere Prozesse mit Bezug zum aktuellen Umgebungskontext stellen ubiquitären Anwendungen Informationen mit hohem Grad an Aktualität bereit. Die gesteigerte Umgebungssensitivität erhöht die Nützlichkeit von ubiquitären Applikationen. Tabelle 8.1 zeigt für die vier Prozessklassen und das Energiemanagement, welche Verbesserungen im Einzelnen erreicht wurden. Alle Verbesserungen sind aus einem realen Anwendungsumfeld motiviert.

Das konsistente Bild aus Theorie, Simulation und experimenteller Implementierung bildet den Nachweis der zweiten Aussage der These.

Dritte Aussage der These: Die Kombination von Kollaboration und Kooperation in einer Kaskade maximiert den Nutzen für ubiquitäre Rechnersysteme.

Erkenntnis der Arbeit: Periodische und verteilte Prozesse kombinieren Kooperation und Kollaboration in einer Kaskade. Dabei justiert der äußere Regelkreis, das Regelungsziel des inneren entsprechend weiteren Informationen über die Ausführungsbedingungen nach. Die Dauer des kooperativen respektive kollaborativen Prozessverhaltens wird maximiert. Die Reihenfolge der Mechanismen in der Kaskade wird nur vom Anwendungsfall bestimmt.

Mit der Maximierung der Kooperations- respektive Kollaborationsdauer geht als Folge der bestätigten zweiten Aussage der These eine Maximierung des Nutzens einher. Damit ist die dritte Aussage der These nachgewiesen.

8.3 Ausblick

Die kooperative und kollaborative Prozessorganisation ermöglicht, dass die Erbringung von Diensten und Anwendungsfunktionalitäten ubiquitärer Rechnersysteme in Umgebungen mit unbekannten Störeinflüssen realisiert werden kann. Zwei Erkenntnisse dieser Arbeit bilden dafür das Fundament.

1. Die Verwendung von Rückkopplungsinformationen ermöglicht das Zusammenwirken und die Zusammenarbeit von Prozessen, um in Einsatzumgebungen unbekannte und veränderliche Störeinflüsse auf das Prozessausführungsverhaltens zu kompensieren.

2. Das Budget/Kosten Regelkreismodell *garantiert* durch mathematischen Nachweis, dass sich ein gewünschtes kooperatives respektive kollaboratives Prozessverhalten einstellt.

Die komplexen Interaktionen durch Rückkopplungen zwischen ubiquitären Rechnersysteme bilden ein riesiges Forschungsfeld zur Erkundung von Mechanismen der Zusammenarbeit der Systeme. Das Ziel zukünftiger Forschungen sollte sein, die Regelungsmechanismen breiter anzuwenden als für sehr spezielle Fragestellungen in einzelnen Domainen wie Netzwerksteuerung oder Energiemanagement.

Gegenstand zukünftiger Forschung sollten daher Techniken der Identifikation des Regelstreckenmodells für Prozesse in Einsatzumgebungen sein. Modelle, die mehr Variablen als das Budget/Kosten Modell enthalten, erlauben eine feingranulare Beschreibung der Prozessabläufe und von Regelungseingriffen. Prozessscheduler können mit detaillierten Informationen bessere Strategien für eine schnelle Datenverarbeitung realisieren.

Eine Alternative bieten Online-Verfahren der Modellidentifikation, die direkt auf den Rechnersystemen in der Einsatzumgebung eingesetzt werden. Der höhere Rechenaufwand lässt sich beispielsweise mit parametrisierbaren Klassen von Regelkreisen reduzieren. Robuste Regelungsverfahren [168] sind unempfindlicher gegenüber sich änderndem oder nur ungenau bekanntem Verhalten der Regelstrecke.

Der höhere Abstraktionsgrad kooperativer und kollaborativer Prozessorganisationsmechanismen und deren mathematische Modellierung sind zudem attraktiv für die Integration in modellgetriebene Softwareentwicklungsprozesse. Die in Abschnitt 3.4.4 vorgestellte Methode zur Übertragung auf konkrete Prozessklassen kann als Ausgangspunkt für eine entsprechende Abbildung auf das Zielsystem mit seinen Prozessen dienen. Für beide Modellierungsansätze steht eine breite Werkzeugunterstützung zur Verfügung.

Die steigende Komplexität durch den sich ausweitenden Einsatz ubiquitärer Rechnersysteme wird die Motivation für kooperative und kollaborative Organisationsverfahren erhöhen. In zukünftigen Beiträgen werden Forscher mit den Mechanismen der Kooperation und Kollaboration sicherlich noch viele, neue Anwendungen mit verblüffenden Eigenschaften aufzeigen. Das in dieser Arbeit gebildete Verständnis der Mechanismen und die gezeigten Experimente bereiten diese Weiterentwicklung vor.

Anhang

A. Herleitung der Transferfunktion des Regelkreises

Die Transferfunktion des Regelkreises wird auf Basis der Signale aus Abbildung A.1 hergeleitet. Die Transferfunktion des Regelkreises $F(z)$ setzt den Sollwert $R(z)$ zum

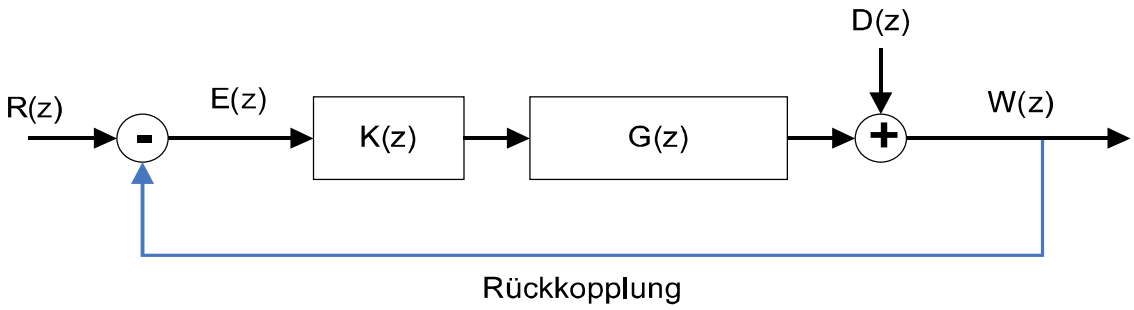


Abbildung A.1: Blockschaltbild eines allgemeinen Regelkreises mit Signalbezeichnungen

Ausgang $W(z)$ in Beziehung. Die Störgröße wird mit $D(z) = 0$ angenommen. Damit entspricht $W(z)$ dem Ausgang der Regelstrecke $G(z)$. Die Störung ist somit nicht Teil der Transferfunktion. Sie wird im allgemeinen als Unbekannte angenommen, deren Einfluss durch die Regelung aufgehoben werden soll. Die Herleitung beginnt mit

dem Aufstellen der Transferfunktion der konkatenierten Blöcke und wird sukzessive in die Form $F(z) = \frac{W(z)}{R(z)}$ umgestellt.

$$\begin{aligned}
W(z) &= E(z)K(z)G(z) \\
E(z) &= R(z) - W(z) \\
&= R(z) - E(z)K(z)G(z) \\
R(z) &= [1 + K(z)G(z)]\, E(z) \\
E(z) &= \frac{R(z)}{1 + K(z)G(z)} \\
W(z) &= E(z)K(z)G(z) \\
&= R(z)\frac{K(z)G(z)}{1 + K(z)G(z)} \\
F(z) &= \frac{W(z)}{R(z)} \\
&= \frac{K(z)G(z)}{1 + K(z)G(z)}
\end{aligned} \tag{A.1}$$

B. Impulsfunktion des Budget/Kosten Modells

Die Transferfunktion des Budget/Kosten-Modells ist

$$F(z) = \frac{Kz}{z(K+1)-1} \tag{B.1}$$

Es wird nun die Impulsfunktion, also die Antwort des Systems auf ein impulsartiges Eingangssignal, berechnet. Es gilt $I(z) = F(z)1 = F(z)$. Damit entspricht die Impulsfunktion im Zeitbereich einer Z Rücktransformation der Transferfunktion. Die Transferfunktion wird entsprechend umgeformt

$$\begin{aligned} F(z) &= \frac{Kz}{z(K+1)-1} \\ &= \frac{K}{K+1}\frac{z}{z-\frac{1}{K+1}} \end{aligned}$$

bevor die Rücktransformation zur Impulsfunktion durchgeführt wird.

$$\begin{aligned} I(k) = Z^{-1}[F(z)] &= Z^{-1}\left[\frac{z}{z-\frac{1}{K+1}}\right]\frac{K}{K+1} \\ &= \left(\frac{1}{K+1}\right)^k \frac{K}{K+1} \end{aligned} \tag{B.2}$$

Gleichung B.2 ist der Impulsfunktion des Budget/Kosten-Modells.

C. Beispiel der Graphentransformation

Es wird beispielhaft die Transformation des Servicegraphen in Abbildung C.1 (rechts) in den Baum (linke Seite der Abbildung) gezeigt.

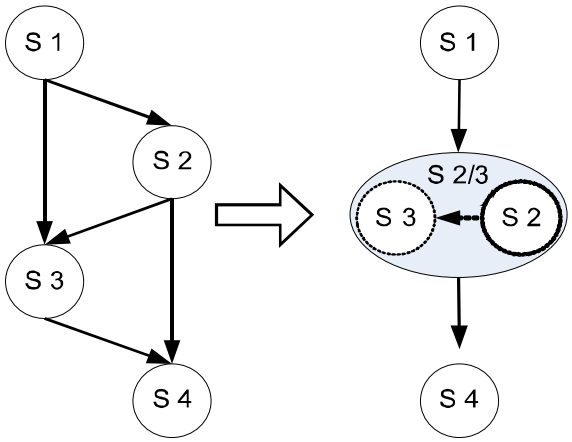


Abbildung C.1: Beispielhafte Transformation des Servicegraphen in einen Baum

1. **Identifikation:** Adjazenzmatrix V_{ij} zum Graph

$$V_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{C.1}$$

   Wobei V_{4j} und V_{i1} entsprechend das Blatt und die Wurzel repräsentieren. Identifiziere vom Blatt ausgehend den nächsten Service, der mehr als einen Vorgänger hat. Es wird mit $Pred(4) = 3, 2$ Service 4 identifiziert.

2. **Verschmelzung:** Service 4 ist Blatt und hat zwei Vorgänger $Pred(4) = 3, 2$. Es entsteht der neue Service 2/3, der die Services 2 und 3 in dieser Reihenfolge

ausführt, da $v_{23} = 1$. Die Addition von Spalte V_{i3} auf V_{i2} und Elimination von V_{i3} und V_{3j} lässt die neue Adjazenzmatrix mit dem Service 2/3 in Spalte und Zeile 2 entstehen.

3.
$$V_{ij} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \tag{C.2}$$

Der Algorithmus ist beendet, da $\forall i : Pred(i) = \emptyset$ ist, d.h es gibt keinen Service mit mehr als einem Vorgänger.

D. Jitterkorrektur

Jitter verändert die Aktivierungszeitpunkte periodischer Prozesse. Jitter tritt zufällig auf, ist aber auf einen maximalen Wert begrenzt. Es entsteht eine Periodenabweichung als Folge von ungleichmäßigen Aktivierungszeitpunkten. Die Jitterkorrektur modifiziert die nächste Aktivierungszeit des Wurzelservice s_k eines Prozesses zu $a_{k,n} = a_{k,n-1} + T_k + Jit_{k,n-1}$, wobei $Jit_{k,n-1}$ die Abweichung von der Aktivierungszeit der $n-1$-ten Jobausführung von s_k bezeichnet. Das folgende Theorem trifft eine Aussage über die Grenzen der Abweichung bei Anwendung der Jitterkorrektur.

Theorem D.1. *Die Jitterkorrektur $a_{k,n} = a_{k,n-1} + T_k + Jit_{k,n-1}$ begrenzt die Periodenabweichung T_k auf $\pm \max_n Jit_k$.*

Beweis. Zuerst wird die Periode eines Prozesses mit Hilfe der Differenz zwischen den Startzeiten der $(n-2)$-ten Ausführungsinstanz und der $(n-1)$-ten Instanz berechnet. Es wird angenommen, das die $(n-1)$ Instanz durch den Jitter $Jit_{k,n-2}$ verzögert wurde. Offensichtlich ist die Differenz $a_{k,n-1} + Jit_{k,n-2} - a_{k,n-2} = T_k + Jit_{k,n-2}$. Im zweiten Schritt wird die zeitliche Differenz zwischen der n-ten Ausführungsinstanz und der $(n-1)$-ten Instanz berechnet. Die $(n-1)$ Instanz startet an $a_{k,n-1} = a_{k,n-2} + T_i + Jit_{k,n-2}$, weil sie verzögert wurde. Wegen der Jitterkorrektur beginnt die n-te Instanz von $a_{k,n} = a_{k,n-1} + T_k$. Dennoch kann $a_{k,n}$ einen zusätzlichem Jitter $Jit_{k,n-1}$ erfahren. Die Differenz ist $a_{k,n} - a_{k,n-1} = T_k + Jit_{k,n-1}$. Die Periodenabweichung ΔT ist jetzt die Differenz zwischen zwei hintereinanderfolgenden Perioden von Prozessen: $\Delta T = |Jit_{k,n-2} - Jit_{k,n-1}| \leq \max_n Jit_{k,n}$. Im allgemeinen Fall kann Jitter nicht nur verzögern, sondern auch einen verfrühten Ausführungsbeginn verursachen. Die Periodenabweichung ist daher $\Delta T = \pm \max_n Jit_{k,n}$. □

Die Abbildung D.1 illustriert die Schwankungen der Ankunftszeit mit und ohne Jitterkorrektur. Die Simulation bezieht Jitterabweichungen in beide Richtungen ausgehend von der Aktivierungszeit ein. Die Ausführung der Prozesse wird zufällig mit Jitter aus dem Bereich $[-1; 1]$ gestört. Die Abbildung D.1 stellt grafisch die Verteilung der Periodenabweichungen für Simulationen mit und ohne Jitterkorrektur dar. Wie oben bewiesen bleibt die Abweichung zwischen ± 1 für die Prozesse mit Korrektur. Prozesse ohne Jitterkorrektur erfahren eine Abweichung zwischen ± 2. In diesem Fall ist die Abweichung breiter verteilt als in dem Fall mit Korrektur.

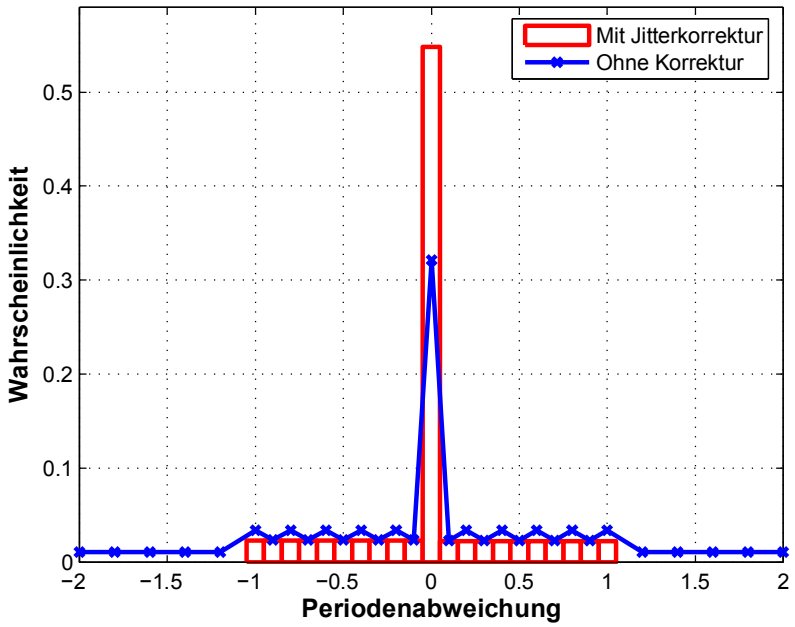


Abbildung D.1: Periodenänderung periodischer Prozesse bei maximalen Jitter $Jit_{\max} = 1$. Simulation mit und ohne Anwendung der Jitterkorrektur.

E. Least Quality First (LQF)

LQF soll Prozesspfade so einplanen, dass kein Pfad über längere Zeit größere Qualitätsminderungen erfährt. Daher wird der Pfad mit der kleinsten Qualität im nächsten Schedulingschritt als erster eingeplant. In der Abbildung E.1 sind über 30 Schedulingschritte hinweg die Prozessqualitäten von fünf Prozessen aus einem Servicegraphen aufgezeichnet. Kein Prozess erfährt dauerhafte Qualitätsminderungen. Al-

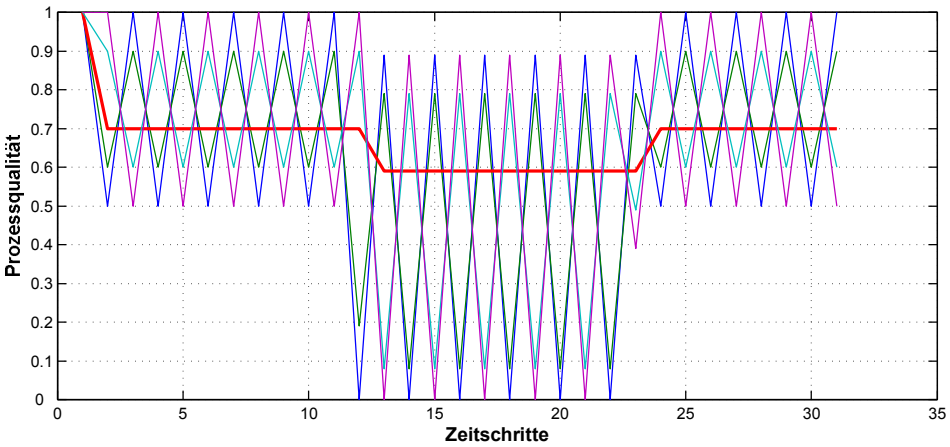


Abbildung E.1: Alternierende Ausführungsqualitäten von Prozesspfaden unter LQF. Im Zeitabschnitt $t = 11 \ldots 22$ wird die Ausführungsdauer eines Prozesses stark erhöht. (Quelle: [78])

lerdings unterliegen die einzelnen Qualitäten starken und jeweils unterschiedlichen Schwankungen, da die Reihenfolge der Prozesse stark verändert wird. Im Zeitabschnitt $t = 11 \ldots 22$ wird die Ausführungsdauer eines Prozesses stark erhöht und anschließend wieder auf den ursprünglichen Wert zurückgesetzt. Nach Korollar 4.1 bewirkt dies eine allgemeine Qualitätsminderung der Qualitätensumme aller Prozesse. Die Oszillation der Qualitäten folgt der allgemeinen Qualitätsminderung. Es ist zu erkennen, dass ein Prozess einen verzögerungsneutralen Platz in der Schedulingreihenfolge einnimmt. Seine Qualität ist stabil und oszilliert nicht. Fairness

im Sinne einer ausgeglichenen und gleichbleibenden Ausführungsqualität wird nicht erreicht.

F. Herleitung der Serverperioden für die kooperative aperiodische Prozessorganisation

Der Kooperationsmechanismus für aperiodische Prozesse aus Abschnitt 4.5.5 soll eine vorgegebene Systemlast U_b zur Verarbeitung der aperiodischen Ereignisse nicht überschreiten, um Überlast zu vermeiden. Dazu werden die Serverperioden der Prozesse angepasst.

F.1 Ansatz

Es werden zwei Mengen von Prozessen I, $\mathcal{J}$ mit $I \cup \mathcal{J} = \mathcal{P}^{\mathrm{aP}}$ und $I \cap \mathcal{J} = \emptyset$ definiert. Die Verlustwahrscheinlichkeiten der Prozesse der Menge I bleiben unverändert, während die Prozesse aus $\mathcal{J}$ durch Periodenverlangsamung die erhöhte Serverauslastung von I kompensieren. Die Serverauslastung durch die Prozesse I muss kleiner als die gegebene Auslastung U_b sein. Die Zugehörigkeiten zu den beiden Mengen kann in jeder einzelnen Serverperiode neu bestimmt werden.

Es sei $U_1(w)$ die Serverauslastung im Zeitintervall w für einen ausgewählten Prozess aus der Menge $\mathcal{J}$, zum Beispiel den Prozess mit der größten Serverauslastung. Im Verhältnis zu $U_1(w)$ kann man alle weiteren Prozesse $j \in \mathcal{J}$ darstellen. Um die gegebene Auslastungsgrenze U_b nicht zu überschreiten, soll dieses Verhältnis auch nach der Anpassung im Zeitintervall $w+1$ erhalten bleiben. Es gilt die Bilanzgleichung F.1

$$g_j(w) = \frac{U_j(w)}{U_1(w)} = \frac{U_j(w+1)}{U_1(w+1)} \tag{F.1}$$

Die Serverauslastung für einen Prozess j nach der Anpassung berechnet sich folglich zu

$$U_j(w+1) = g_j(w) U_1(w+1) \tag{F.2}$$

F.2 Aufteilung von U_b im Zeitintervall $w+1$

Im ersten Schritt werden die Prozesse aus I behandelt. Dazu werden die Serverperioden durch das jeweilige Steuerungsglied gemäß der Gleichung 4.14 berechnet. Die Prozesse in I erzeugen die neue Serverauslastung $U_I(w+1) = \sum_{i \in I} U_i(w+1)$. Im nächsten Schritt müssen die Prozesse der Menge $\mathcal{J}$ auf die verbliebene Auslastung aufgeteilt werden.

$$\begin{aligned} U_b - \sum_{i \in I} U_i(w+1) &= \sum_{j \in \mathcal{J}} U_j(w+1) \\ &= \sum_{j \in \mathcal{J}} g_j(w) U_1(w+1) \text{ m.Gl. F.2} \\ U_1(w+1) &= \frac{U_b - \sum_{i \in I} U_i(w+1)}{\sum_{j \in \mathcal{J}} g_j(w)} \end{aligned}$$

Das Ergebnis für $U_1(w+1)$ wird in Gleichung F.2 eingesetzt.

$$\begin{aligned} U_j(w+1) &= g_j(w) \frac{U_b - \sum_{i \in I} U_i(w+1)}{\sum_{j \in \mathcal{J}} g_j(w)} \\ &= \frac{g_j(w)}{\sum_{j \in \mathcal{J}} g_j(w)} \left(U_b - \sum_{i \in I} U_i(w+1) \right) \\ &= \frac{U_j(w)}{\sum_{j \in \mathcal{J}} U_j(w)} \left(U_b - \sum_{i \in I} U_i(w+1) \right) \text{ m.Gl. F.1} \\ &= \frac{U_b - \sum_{i \in I} U_i(w+1)}{\sum_{j \in \mathcal{J}} U_j(w)} U_j(w) \end{aligned} \tag{F.3}$$

Mit $U_j(w) = \frac{C_j}{T_j(w)}$ folgt für die Berechnung der Serverperioden

$$T_j(w+1) = \frac{\sum_{j \in \mathcal{J}} U_j(w)}{U_b - \sum_{i \in I} U_i(w+1)} T_j(w) \tag{F.4}$$

Werden die Auslastungen U_j und U_i in Gleichung F.4 durch die Ausdrücke der Steuerungsglieder aus Gleichung 4.14 ersetzt, so folgt direkt die Berechnung der Serverperioden durch die erfasste Ereignisrate

$$T_j(w+1) = \frac{\sum_{j \in \mathcal{J}} C_j \lambda_j(w)}{RU_b - \sum_{i \in I} C_i \lambda_i(w+1)} T_j(w) \tag{F.5}$$

Die Gleichung F.5 ist nicht kausal, da zur Berechnung von $T_j(w+1)$ gemessene Informationen über $\lambda_i(w+1)$ verwendet werden. Sie gilt nur, wenn Steuerbarkeit des Systems nachgewiesen wird. Dies wird in Abschnitt 4.5.6 diskutiert.

G. Herleitung der mittleren Wartezeit von Ereignissen in der M/D/k/k Schlange

Die mittlere Wartezeit $E[T_w]$ ist der Erwartungswert der Aufenthaltsdauer von Ereignisdaten zwischen Einbringen in die Warteschlange und Verarbeitung durch den periodischen Serverprozess. Letzterer *verarbeitet alle Ereignisse in einem Schritt.* Daher kommt die Herleitung von $E[T_w]$ ohne ein Markov-Modell aus.

Die mittlere Anzahl an Ereignissen ist $\bar{N} = \lambda T$. Man kann erwarten, dass der Puffer im Mittel bis zu dieser Größe gefüllt ist. Für die Durchschnittsbetrachtung ist es ausreichend, daß die Ereignisse gleichmäßig mit der Rate λ ankommen. Der zeitliche Abstand zweier Ereignisse ist im Mittel $\frac{1}{\lambda}$. Abbildung G.1 zeigt den Ereignispuffer. Es wird zuerst der Fall $\bar{N} = \lambda T$, $\bar{N}$ ist ganzzahlig, betrachtet. Die mittlere Wartezeit

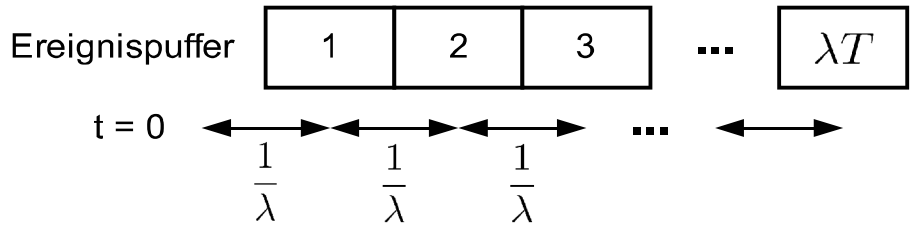


Abbildung G.1: Darstellung des Puffers und der Ereigniszeiten für eine Durchschnittsbetrachtung.

berechnet sich wie folgt

$$E[T_w]^{\text{ganzz.}} = \frac{\sum_{i=1}^{\lambda T} \frac{1}{\lambda} i}{\lambda T} \tag{G.1}$$

Gebrochen rationale $\bar{N}$ werden als Anteile der ganzzahligen Grenzen, zwischen denen sie liegen, dargestellt. Es ist

$$\bar{N} = \lfloor \bar{N} \rfloor \left(\lceil \bar{N} \rceil - \bar{N} \right) + \lceil \bar{N} \rceil \left(\bar{N} - \lfloor \bar{N} \rfloor \right)$$

Diese Darstellung wird auf Gleichung G.1 angewendet. Es folgt mit

$$E[T_w] = \frac{(\lceil \lambda T \rceil - \lambda T) \sum_{i=1}^{\lfloor \lambda T \rfloor} \frac{1}{\lambda} i + (\lambda T - \lfloor \lambda T \rfloor) \sum_{i=1}^{\lceil \lambda T \rceil} \frac{1}{\lambda} i}{\lambda T}$$

sofort Gleichung für die mittlere Wartezeit von Ereignissen in der M/D/k/k Schlange.

H. Systemmodellierung und Regelung von Echtzeitprozessen

Der Anhang gibt detaillierte Zusatzinformationen zur Systemmodellierung von Echtzeitprozessen. In Abschnitt H.2 wird ein tieferes Verständnis für die Regelinitialisierung $\forall i: \; T_i = \max_i T_i$ gegeben.

H.1 Herleitung der Transferfunktion der Prozessausführung

Es wird die Übertragungsfunktion des offenen Systems der Prozessausführung hergeleitet. Es ist $S^{\mathrm{rt}} = \{s_i^{\mathrm{rt}}\}$ die Menge der Echtzeitservices. Sie bildet eine Ablaufgruppe (engl. gang). Es ist $s(w)$ Anzahl der Ausführungen der Servicegruppe S^{rt}, d.h. $s(w) = 1$ alle Echtzeitservices werden einmal ausgeführt. Es sind S^{nrt} und $j(w)$ die analogen Entsprechungen für datengetriebene Services. Die jeweiligen Ausführungszeiten der Ablaufgruppen sind $C^{\mathrm{rt}} = \sum_i C_i^{\mathrm{rt}}$ für die Echtzeitservices und $C^{\mathrm{nrt}}(w) = \sum_j C_j^{\mathrm{nrt}}(w)$ für die datengetriebenen.

In der Modellierung wird die Ausführung in einem Zeitintervall w mit der Länge I beobachtet. Die durchschnittliche Periode $\bar{T}$ der Echtzeitservices in I ist

$$\begin{aligned} \bar{T} &= \frac{I}{s(w)} \\ I &= \bar{T} s(w) \end{aligned} \tag{H.1}$$

Die Datenverarbeitungsleistung im w-ten Intervall ist

$$P(w) = s(w) - j(w) \tag{H.2}$$

Im Fall von $P(w) = 0$ gibt es zu jeder Echtzeitserviceausführung eine entsprechende Ausführung der datengetriebenen Verarbeitung. $P(w) > 0$ bezeichnet einen Datenverlust, während $P(w) < 0$ Leerlauf angibt. Die Datenverarbeitungsleistung $P(w)$

spiegelt also sowohl den Verlustzähler wie auch den Leerlaufzähler des Entwurfs der Datenverarbeitung aus Abschnitt 5.4.1 wieder. Die Ausführung verursacht eine Prozessorauslastung im w-ten Beobachtungsintervall. Mit Gleichung H.1 und $j(w) = s(w) - P(w)$ ist

$$\begin{aligned} U(w) = \frac{C(w)}{I} &= \frac{s(w)C^{\mathrm{rt}} + j(w)C^{\mathrm{nrt}}(w)}{\bar{T}s(w)} \\ &= \frac{s(w)C^{\mathrm{rt}} + [s(w) - P(w)]\,C^{\mathrm{nrt}}(w)}{s(w)T(w)} \end{aligned} \tag{H.3}$$

Die Rechenzeitressourcen des ubiquitären Rechnersystems sollen bestmöglich genutzt werden. Mit $U(w) = 1$ folgt

$$\begin{aligned} 1 &= \frac{C^{\mathrm{rt}}}{T(w)} + \frac{C^{\mathrm{nrt}}(w)}{T(w)} - \frac{P(w)C^{\mathrm{nrt}}(w)}{s(w)T(w)} \\ s(w)T(w) &= s(w)C^{\mathrm{rt}} + s(w)C^{\mathrm{nrt}}(w) - P(w)C^{\mathrm{nrt}}(w) \end{aligned}$$

Daraus folgt sofort die Beziehung zur Berechnung der Datenverarbeitungsleistung.

$$P(w) = K_f(w)\,[D(w) - T(w)] \tag{H.4}$$

mit $K_f(w) = \frac{s(w)}{C^{\mathrm{nrt}}(w)}$ und $D(w) = C^{\mathrm{rt}} + C^{\mathrm{nrt}}(w)$. Im Idealfall ist $K_f(w) = K_f = \frac{1}{C^{\mathrm{nrt}}} = \text{konst.}$. Mit dieser Annahme wird die Modellierung fortgeführt. Die Veränderung dieses Parameters wird separat bei der Analyse des BFS Regelkreises untersucht. Die Ausführung der Prozesse ist ein dynamisches System. Es wird die Änderung $\Delta P(w)$ untersucht.

$$\begin{aligned} \Delta P(w) &= K_f\,[D(w) - T(w)] - K_f\,[D(w-1) - T(w-1)] \\ &= K_f\,[\Delta D(w) - \Delta T(w)] \end{aligned} \tag{H.5}$$

$D(w)$ ist die Veränderung der Ausführungsdauer der datengetriebenen Services. Das ist die Störgröße des Systems und wird aus der Modellierung des offenen Systems ausgelassen. Der geschlossene Regelkreis hat die Aufgabe, unbekannte Störungen zu kompensieren. Es ist

$$P(w) = P(w-1) + \Delta P(w) = P(w-1) - K_f \Delta T(w) \tag{H.6}$$

die Differenzengleichung des Systems der Prozessausführung von Echtzeitprozessen. Durch Z Transformation und Umformen von Gleichung H.6 ergibt sich die Transferfunktion des offenen Systems

$$P(z) - P(z)z^{-1} = -K_f \Delta T(z) \tag{H.7}$$

$$P(z)(1 - z^{-1}) = -K_f \Delta T(z) \tag{H.8}$$

$$G(z) = \frac{P(z)}{\Delta T(z)} = -K_f \frac{z}{z-1} \tag{H.9}$$

H.2 Reglerinitialisierung

Zur Vermeidung beliebig großer Amplituden initialisiert der Regler alle Ausführungsperioden auf $\max_i T_i$. Die Motivation dieser Entscheidung wird in den folgenden Unterabschnitten gegeben.

H.2.1 Einzelfallbeispiel

Es wird zuerst der Fall mit nur einem Echtzeitservice und einem datengetriebenen Service betrachtet. Datenverarbeitungsleistung $P(w) = 0$ wird erreicht, wenn nach jeder Ausführung des Echtzeitservice die Verarbeitung erfolgt. Es muss eine Synchronisation zwischen beiden Services bestehen. Andernfalls wird nach mehreren Perioden Verlust oder Leerlauf auftreten. Echtzeit- wie auch datengetriebener Service werden als Integrationsglieder $G^{\mathrm{rt/nrt}}(z) = \pm\frac{z}{z-1}$ jeweils mit unterschiedlichem Vorzeichen modelliert. Abbildung H.1 zeigt die Schrittantwort. Beide Systeme sind

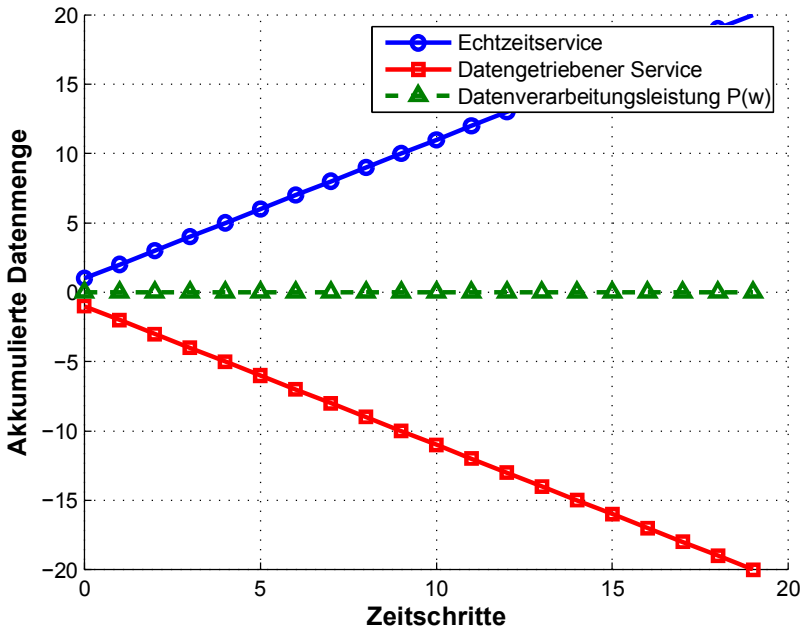


Abbildung H.1: Schrittantwort und Datenverarbeitungsleistung $P(w)$ synchronisierter Echtzeit- und datengetriebener Services.

nicht stabil, trotzdem ist $P(z) = G^{\mathrm{rt}}(z) - G^{\mathrm{nrt}}(z) = 0$ konstant, da die Systemantworten sich *nur* durch unterschiedliche Vorzeichen annihilieren.

In der Regelungstechnik sind solche Systeme mathematisch korrekt, aber praktisch sehr fragil. Ändert sich die Verarbeitungsfrequenz nur minimal, d.h $G^{\mathrm{nrt},*}(z) = \delta G^{\mathrm{nrt}}(z)$, dann ist das entstehende System

$$P(z) = G^{\mathrm{rt}}(z) - G^{\mathrm{nrt},*}(z) = (1-\delta)\frac{z}{z-1}$$

nicht stabil für jedes $\delta \neq 1$.

Im gezeigten Beispiel in Abbildung H.1 werden Echtzeit- und datengetriebener Service *synchronisiert* ausgeführt. Dabei ist es ausreichend, dass die Synchronisation bis auf einen konstanten Jitter genau ist, damit $\delta \equiv 1$. Unter dieser Voraussetzung ist die Frequenzantwort des Gesamtsystems begrenzt: In jedem Schritt, wird nur *genau ein* Datum bereitgestellt und verarbeitet. Das Gesamtsystem arbeitet stabil.

H.2.2 Unterschiedliche Serviceperioden

Im Fall von mehreren Echtzeitservices mit verschiedenen Ausführungsperioden wird eine Durchschnittsperiode der Datenverarbeitung eingeführt.

O.B.d.A. seien $G_1(z), G_2(z)$ zwei Echtzeitservices, die mit Perioden T_1 und T_2 Daten akquirieren und bereitstellen. Es ist $C(z)$

$$\begin{aligned} C_1(z) &= \frac{1}{T_1}\frac{z}{z-1} \\ C_2(z) &= \frac{1}{T_2}\frac{z}{z-1} \\ C(z) &= \frac{1}{2}\left(\frac{1}{T_1}+\frac{1}{T_2}\right)\frac{z}{z-1} \end{aligned} \tag{H.10}$$

das Modell aller datengetriebenen Services, die mit Durchschnittsperiode die bereitgestellten Daten verarbeiten. Mit Gleichung H.10 kann das folgende Gesamtsystem der Datenverarbeitung hergeleitet werden.

$$\begin{aligned} G_1(z) - C(z) &= G_1(z) - \left(\frac{1}{2}C_1(z) + C_2(z)\right) \\ G_2(z) - C(z) &= G_2(z) - \left(\frac{1}{2}C_1(z) + C_2(z)\right) \\ G_1(z) - C(z) + G_2(z) - C(z) &= G_1(z) - C_1(z) + G_2(z) - C_2(z) \end{aligned} \tag{H.11}$$

Gleichung H.11 besagt, dass bei Datenverarbeitung mit Durchschnittsperiode gleichzeitig mit Verarbeitungsperiode T_1 *und* T_2 gearbeitet werden muss. Da alle datengetriebenen Services eine gemeinsame Periode haben, kann ein Puffer nicht mit einer höheren, zum Beispiel T_1, und ein anderer mit einer niedrigeren Periode, zum Beispiel T_2, verarbeitet werden. In diesem Fall können die datengetriebenen Services *zu keiner Zeit synchronisiert* mit den Echtzeitservices ausgeführt werden. Das Gesamtsystem kann nicht stabilisiert werden.

Daher wird bei Reglerinitialisierung für alle Echtzeitservices ebenfalls eine gemeinsame Periode vereinbart. Mit der Initialisierung $\forall i : T_i = \max_i T_i$ bleiben gemäß Korollar 5.2 die Services echtzeitfähig einplanbar. Datengetriebene Services und Echtzeitservices können synchronisiert werden und das Gesamtsystem kann wie im Beispiel aus Abbildung H.1 stabilisiert werden.

I. Beweise der Schedulinganalyse für Echtzeitprozesse

Dieser Anhang enthält die Beweise zu den Korollaren aus Abschnitt 5.5.4.

I.1 Beweis zu Korollar 5.1

Das Korollar 5.1 wird im folgenden wiederholt.

Korollar I.1. *Ist die Menge der Echtzeitservices $S = \{(C_1, T_1), \ldots, (C_n, T_n)\}$ einplanbar gemäß Theorem 5.1 ist, dann ist die neue Menge $S^* = \{(C_i, T_i + k)\}$, in der jede Periode um die Konstante $k > 0$ vergrößert wurde, ebenfalls unter Einhaltung aller Zeitschranken einplanbar.*

Um das Korollar zu beweisen, wird zuerst das folgende Lemma formuliert und bewiesen.

Lemma I.1. *Wenn $g(k), h(k)$ zwei lineare Funktionen mit $\frac{\partial g}{\partial k} = \frac{\partial h}{\partial k}$ sind und $\forall k, k > 0$, dann ist $\frac{g(k)}{h(k)}$ monoton.*

Beweis. Lemma I.1 wird konstruktiv mit der Untersuchung folgender Ableitung $\frac{\partial}{\partial k}\frac{g(k)}{h(k)}$ bewiesen. Seien $g(k) = k + x$ und $h(k) = k + y$, dann $\frac{\partial}{\partial k}\frac{g(k)}{h(k)} = \frac{y-x}{(x+k)^2}$. Für $x, y > 0$ und $x < y$ ist $\frac{\partial}{\partial k}\frac{g(k)}{h(k)} > 0$. Daher ist $\frac{g(k)}{h(k)}$ monoton steigend. Für $x, y > 0$ und $x > y$ ist $\frac{\partial}{\partial k}\frac{g(k)}{h(k)} < 0$. Daher ist $\frac{g(k)}{h(k)}$ monoton fallend. Für $x, y > 0$ und $x = y$ ist $\frac{\partial}{\partial k}\frac{g(k)}{h(k)} = 0$. Daher ist $\frac{g(k)}{h(k)}$ gleichzeitig monoton fallend und steigend. □

Es folgt der Beweis zu Korollar 5.1.

Beweis zu Korollar 5.1. Das Theorem 5.1 von Jeffay et al. nennt zwei Bedingungen zur Einhaltung hart Zeitschranken in dynamischen Schedulingverfahren.

(1) $\sum_i \frac{C_i}{T_i} \leq 1$

(2) $\forall i, 1 < i \leq n; \forall L, T_1 < L < T_i : L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_j} \right\rfloor C_j$

Die Bedingungen sind notwendig und hinreichend. Dieser Beweis untersucht die Einhaltung der Bedingungen für Korollar 5.1.

Zuerst wird Bedingung (1) des Jeffay Theorems untersucht. Aus $T_i + k > T_i$ folgt direkt, dass $\frac{C_i}{T_i} > \frac{C_i}{T_i+k} \forall k, k > 0$. Daher ist $\sum_i \frac{C_i}{T_i+k} < \sum_i \frac{C_i}{T_i} \leq 1$.

Es wird im nächsten Schritt Bedingung (2) untersucht. Es ist zu beachten, dass in dieser Bedingung k ebenso auf L angewendet wird, so dass sich der Ausdruck wie folgt ändert: $\forall L, T_1 + k < L + k < T_i + k$. Für die folgende Fälle ist es ausreichend, den Ausdruck $\left\lfloor \frac{L-1}{T_j} \right\rfloor$ in Bedingung (2) von Jeffays Theorem für diese Veränderung zu untersuchen:

Fall 1:$(L - 1 < T_j)$ Die Vergrößerung der Perioden T_j um eine konstante Zeit k führt zu: $\forall k, k > 0 : \lim_{k\to\infty} \frac{L-1+k}{T_j+k} = 1^-$, d.h. für alle k mit $k > 0$, konvergiert der Ausdruck $\frac{L-1+k}{T_j+k}$ gegen 1 von der linken Seite. Anhand von Lemma I.1, konvergiert er monoton steigend. Das Ergebnis ist $\left\lfloor \frac{L-1+k}{T_j+k} \right\rfloor = 0$ für alle $k > 0$.

Fall 2:$(L - 1 > T_j)$ Die Vergrößerung der Perioden T_j um eine konstante Zeit k führt zu: $\forall k, k > 0 : \lim_{k\to\infty} \frac{L-1+k}{T_j+k} = 1^+$, d.h. für alle k mit $k > 0$, konvergiert der Ausdruck $\frac{L-1+k}{T_j+k}$ gegen 1 von der rechten Seite. Anhand von Lemma I.1, konvergiert er monoton fallend. Das Ergebnis ist $\left\lfloor \frac{L-1+k}{T_j+k} \right\rfloor < \left\lfloor \frac{L-1}{T_j} \right\rfloor$ für alle $k > 0$.

Fall 3:$(L - 1 = T_j)$ Die Vergrößerung der Perioden T_j um eine konstante Zeit k führt zu $\left\lfloor \frac{L-1+k}{T_j+k} \right\rfloor = \left\lfloor \frac{L-1}{T_j} \right\rfloor$ für alle $k > 0$.

Als Ergebnis ergibt sich (Eigenschaften von S, i, L wie in Bedingung (2) festgesetzt): $L \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_j} \right\rfloor C_j \geq C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1+k}{T_j+k} \right\rfloor C_j, \forall k, k > 0$. S^* ist unter Echtzeitschranken einplanbar □

I.2 Beweis zu Korollar 5.2

Das Korollar 5.2 wird im folgenden wiederholt.

Korollar I.2. *Ist die Menge der Echtzeitservices $S = \{(C_1, T_1), \ldots, (C_n, T_n)\}$ einplanbar gemäß Theorem 5.1 ist, dann ist die neue Menge $S^{*,max} = \{(C_i, T^{max})\}$ mit $T^{max} = \max_i T_i$ ebenfalls unter Einhaltung aller Zeitschranken einplanbar.*

Beweis zu Korollar 5.2. Es ist zu zeigen, dass Bedingung (1) und (2) des Theorems von Jeffay et al. 5.1 erfüllt sind. Die Ausführungsdauern C_i der Echtzeitservices sind konstant.

Bedingung (1). Es ist $\sum_i \frac{C_i}{T_i} \leq 1$. $\forall i : T^{\max} = \max_i T_i \geq T_i$. Es folgt, dass $\forall i : \frac{C_i}{T^{\max}} \leq \frac{C_i}{T_i}$. Die Behauptung über die Summe folgt unmittelbar.

Bedingung (2). Es wird $\forall j : T_j = L - 1$ und $T_i = L + 1$ definiert. Es folgt, dass $T_j < L < T_i$. Damit kann die Auslastung im Intervall L berechnet werden. Es ist

$$\begin{aligned} T_i &> L \geq C_i + \sum_{j=1}^{i-1} C_j \\ T_i &> T_j + 1 \geq C_i + \sum_{j=1}^{i-1} C_j \end{aligned} \tag{I.1}$$

Abbildung I.1 zeigt den Zusammenhang der Gleichung I.1 in der Übersicht. Da alle

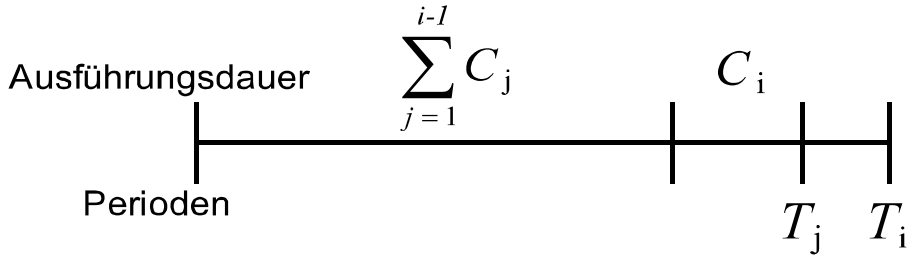


Abbildung I.1: Auslastung aus Gleichung I.1 zeitlich aufgetragen.

Echtzeitservices bereits in T_j abgearbeitet sind, verletzt die zeitliche Dehnung $T_j \rightarrow T_i$ keine Zeitschranken. Die Behauptung von Korollar 5.2 folgt unmittelbar. □

J. Systemmodellierung des FCUP-MAC FIFO Puffers

Im FCUP-MAC FIFO Puffer in Abbildung J.1 werden unmittelbare und mittelbare Nachrichtenpakete unterschieden. Lokale Prozesse legen unmittelbar Nachrichten an entfernte Prozesse in den Puffer ab und entnehmen die an sie datenorientiert adressierten Nachrichten. Das mittelbare Nachrichtenaufkommen von entfernten und für entfernte Prozesse umfasst alle Nachrichten, die für die Weiterleitung bestimmt sind. In der Systemgleichung wird der Zu- und Abfluss unmittelbarer und mittelbarer Nachrichten modelliert.

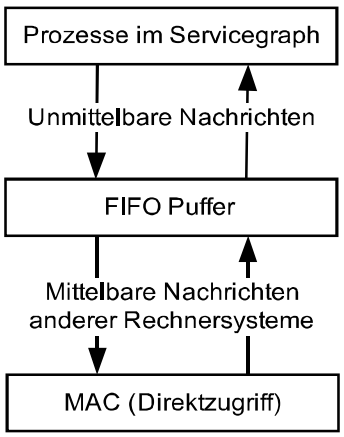


Abbildung J.1: Unmittelbare und mittelbare Nachrichten im FCUP-MAC FIFO Puffer.

Das Kommunikationsintervall wird in Zeitabschnitte gemäß Abbildung J.2 eingeteilt. Dadurch werden die Zugriffe auf den FIFO Puffer geordnet. Jeder Zeitabschnitt wird in Sende- und Empfangsdauer unterteilt, um Zugriffsaktivitäten des Nachrichtenzufluss und -abfluss zu beschreiben. Eine konkrete Reihung, also zum Beispiel zuerst Senden, dann Empfangen, ist nicht notwendig. Die Zeitabschnitte beschreiben lediglich eine Dauer des Vorgangs im Kommunikationsintervall. Die Anzahl der Nachrichten, die in $C^{\text{comm.}}$ ausgetauscht werden können, wird als Bandbreite W bezeichnet.

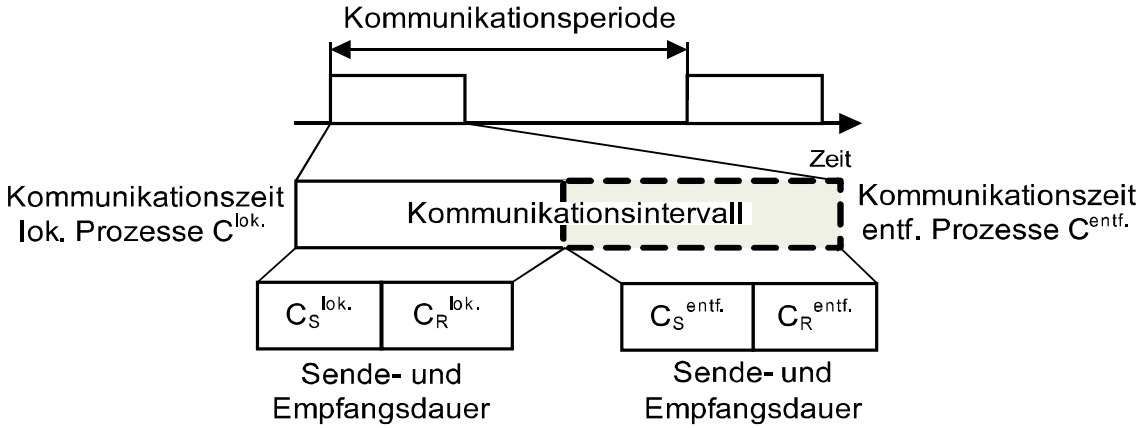


Abbildung J.2: Beschreibung des Kommunikationsintervalls für die Zeitdauer von Zugriffaktivitäten.

Unmittelbare Nachrichten. Es sind $N_S^{\text{lok.}}$ respektive $N_R^{\text{lok.}}$, die Anzahl der Nachrichten, die dem Puffer in der Zeitspanne $C^{\text{lok.}}$ zugeführt respektive aus ihm abgeführt werden.

$$\begin{aligned} N_S^{\text{lok.}} &= C^{\text{lok.}} S \\ N_R^{\text{lok.}} &= C^{\text{lok.}} R \end{aligned}$$

S und R sind Raten, mit denen lokale Prozesse Nachrichten erzeugen respektive verarbeiten. Alle Nachrichten werden über den Kanal kommuniziert.

Mittelbare Nachrichten. Ist die Bandbreite noch nicht erschöpft, d.h. $N_S^{\text{lok.}} + N_R^{\text{lok.}} < WC^{\text{comm.}}$, so ist

$$N^{\text{entf.}} = WC^{\text{comm.}} - (N_S^{\text{lok.}} + N_R^{\text{lok.}}) \tag{J.1}$$

die Anzahl der mittelbaren Nachrichten, die in der verbleibenden Kommunikationsdauer $C^{\text{entf.}}$ maximal den Puffer durchlaufen können. Folglich ergibt sich die zu regelnde Kommunikationsdauer

$$\begin{aligned} C^{\text{entf.}} &= \frac{N^{\text{entf.}}}{W} \\ &= C_S^{\text{entf.}} + C_R^{\text{entf.}} \end{aligned} \tag{J.2}$$

die ähnlich der Kommunikationszeit für unmittelbare Nachrichten in Sende- und Empfangsdauer aufgeteilt wird. Die Gleichungen J.1 und J.2 stellen den Zusammenhang für Kernidee des Modellierungsansatzes her: Die Dynamik des Nachrichtenaustauschs ΔW spiegelt sich direkt in der Pufferdynamik ΔN wider. Es muss also nicht ΔW modelliert werden, sondern für $W = \text{konst.}$ wird die Dynamik mittels $\Delta C^{\text{entf.}}$ ausgedrückt.

Im Kommunikationszeitintervall $[kw; (k+1)w] = C^{\text{comm.}}$ wird die gesamte Dynamik in die zweite Phase $C^{\text{entf.}}(w)$ verschoben. Die Dynamik des Puffers, d.h. die Veränderung des Pufferfüllstandes, ist die Differenz aus gesendeten und empfangenen Nachrichten. Sie berechnet sich wie folgt:

$$\Delta N(w) = \underbrace{(C_R^{\text{entf.}}(w) - C_S^{\text{entf.}}(w))}_{=u(w)} W \tag{J.3}$$

Mit Gleichung J.3 wird die Systemgleichung des FCUP-MAC FIFO Puffers aufgestellt.

$$B(w+1) = B(w) + Wu(w)$$

Die Transferfunktion des Systems in Z Transformation ist

$$G(z) = \frac{B(z)}{U(z)} = \frac{W}{z-1} \tag{J.4}$$

K. Transientes Verhalten von PControl in der Ruhephase

Veränderungen der Batteriespannung unter den gegebenen Sollwert gefährden das Erreichen der vorgegebenen Betriebsdauer. In der Ruhephase des Rechnersystems kompensiert PControl den zu Beginn des w-ten Duty Cycle aufgetretenen Regelungsfehler $E(w) > 0$. Das Spannungsbudget wird an die Betriebsdauersollfunktion angeglichen. Das transiente Verhalten des PControl Regelkreises in dieser Phase soll optimal sein, d.h. $E(w)$ soll schnellstmöglich kompensiert werden.

Ausgehend von der Formulierung des Optimierungsproblems in Abschnitt 7.6.2 ist das Fehlersignal

$$E(z) = -\frac{(-B_k + v(z))z + B_k}{(z-1)^3} \quad \text{mit} \quad h\left(v(z)\right) \leq 0, \text{ sonst beliebig} \tag{K.1}$$

zu untersuchen. $E(z)$ lässt sich als IIR Filter (engl. Infinite Impulse Response) dritter Ordnung beschreiben

$$E(z) = \frac{(B_k - v)z^{-2} - Bz^{-3}}{1 - 3z^{-1} + 3z^{-2} - 1z^{-3}}$$

und anschliessend in die Filter $E_1(z)$ und $E_2(z)$ wie in Abbildung K kaskadieren. Dabei ist

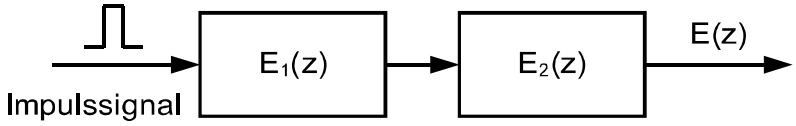


Abbildung K.1: Kaskade des $E(z)$ Filters

$$\begin{aligned} E(z) &= E_1(z) \cdot E_2(z) \text{ mit} \\ E_1(z) &= (B_k - v(z))z^{-2} - B_k z^{-3} \quad \text{FIR Filter} \\ E_2(z) &= \frac{1}{1 - 3z^{-1} + 3z^{-2} - 1z^{-3}} \quad \text{IIR Filter} \end{aligned}$$

Der FIR Filter $E_1(z)$ (engl. Finite Impulse Response) ist stabil mit der Nullstelle bei $z_N = \frac{B_k}{B_k - v(t)}$. Seine Impulsantwort besteht aus den Koeffizienten $B_k - v(z)$ und $-B_k$. Der IIR Filter $E_2(z)$ ist instabil mit einer dreifachen Polstelle bei $z_P = 1$. $E_2(z)$ oszilliert nicht und divergiert je nach Eingangssignal (= Ausgang von $E_1(z)$) gegen $\pm\infty$. Die Koeffizienten von $E_1(z)$ sind die Anregesignale von $E_2(z)$ und bestimmen dessen Amplituden.

Für die Lösung des Optimierungsproblems der schnellsten Kompensation von $E(w) > 0$ muss ein *negatives Fehlersignal mit maximaler Amplitude* erzeugt werden. Da $B_k < 0, \text{konst.}$, ist der zweite Koeffizient von $E_1(z)$ konstant und positiv. Der erste Koeffizient ist abhängig von $v(z)$ und vorzeichenveränderlich. Die Impulsantwort von $E_1(z)$ regt $E_2(z)$ zuerst mit dem Signal $B_k - v(z)$ an und dann mit dem konstant positiven Signal B_k. $v(z)$ bestimmt somit die Amplitude von $E_2(z)$ und somit die Amplitude des Fehlersignals $E(z)$. Tabelle K.1 führt die verschiedenen Kombinationen der Wahl von $v(z)$ und das resultierende Systemverhalten von $E_2(z)$ auf. Der

$h\,(v(z))$	Nullstelle z_N	Koeff. $E_1(z)$	Antwort $E_2(z)$
$v > 0$	$z_N < 1$	$\{b; -B_k\}$, mit $b < B_k$	$-\infty$
$B_k < v < 0$	$z_N > 1$	$\{b; -B_k\}$, mit $B_k < b < 0$	zuerst < 0, dann $+\infty$
$v < B_k$	$z_N > 1$	$\{b; -B_k\}$, mit $b > 0$	$+\infty$
$v = 0$	$z_N = 1$	$\{B_k; -B_k\}$	$-\infty$

Tabelle K.1: Filterkoeffizienten von $E_1(z)$ und resultierendes Systemverhalten von $E_2(z)$.

erste Fall, $v > 0$, verletzt die Optimierungsbedingung und ist nicht anwendbar. In den beiden darauffolgenden Fällen dominiert entweder der größere Betrag des positiven zweiten Koeffizienten oder es sind beide Koeffizienten positiv. Deshalb divergiert $E_2(z)$ gegen $+\infty$. Im letzten Fall, $v = 0$, sind die Koeffizienten antisymmetrisch und regen $E_2(z)$ nacheinander mit dem selben Betrag $|B_k|$ aber unterschiedlichen Vorzeichen an. Kann sich dadurch eine endliche Impulsantwort von $E_2(z)$ ausbilden?

Zur Beantwortung werden die Pole von $E_2(z)$ untersucht. Die Nullstelle $z_N = 1$ von $E_1(z)$ in diesem Fall löscht eine der drei Polstellen $z_P = 1$ von $E_2(z)$. Jedoch ist auch die Impulsantwort eines Systems mit doppelter Polstelle bei $z_P = 1$ nicht endlich. Unter diesen Bedingungen ergibt sich der FIR Filter zu

$$\begin{aligned} E_1(z) &= B_k z - B_k \\ &= B_k(z-1) \\ &= B_k \quad \text{(mit Polauslöschung)} \end{aligned}$$

Da $B_k < 0$, divergiert $E_2(z)$ gegen $-\infty$.

Nur der Fall $v(z) = 0$ erzeugt maximale negative Amplituden. $v(z) = 0$ ist daher die Lösung des Optimierungsproblems. Abbildung K.2 zeigt die Impulsantwort von $E_1(z)$ (links) und von $E_2(z)$ (rechts) in der Filterkaskade aus Abbildung K. Zur Vervollständigung des Bildes werden in den folgenden kurzen Abschnitten weitere Aspekte der Filterkaskade und das optimale Verhalten des Fehlersignals $E(z)$ diskutiert.

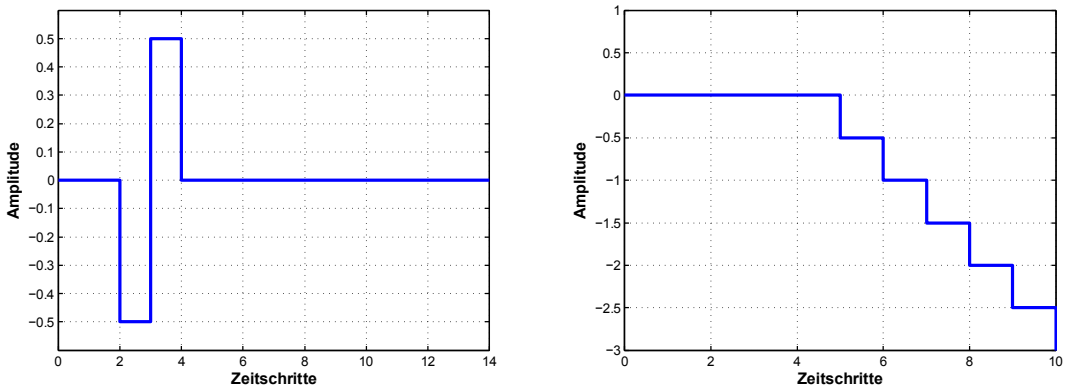


Abbildung K.2: Links: Impulsantwort des $E_1(z)$ FIR Filters für $B_k = -0.5$, $v(z) = 0$. Rechts: Impulsantwort des $E_2(z)$ IIR Filters in der Filterkaskade mit $E_1(z)$ als Eingang. Es werden maximale negative Amplituden erzeugt.

K.1 Dynamik von $v(z)$

Bisher wurde $v(z) =$ konst. vorausgesetzt. Die Tabelle K.1 ist auch für dynamische $v(z)$ erfüllt. Da $E_1(z)$ ein FIR Filter ist, besteht die Impulsantwort immer aus den Koeffizienten. Die Dynamik von $v(z)$ spiegelt sich somit identisch in der Impulsantwort von $E_1(z)$ wieder. Abbildung K.3 illustriert eine sinusoidale Dynamik von $v(z)$ und die zugehörige Impulsantwort von $E_1(z)$. Ab dem dritten Zeitschritt ist die Fil-

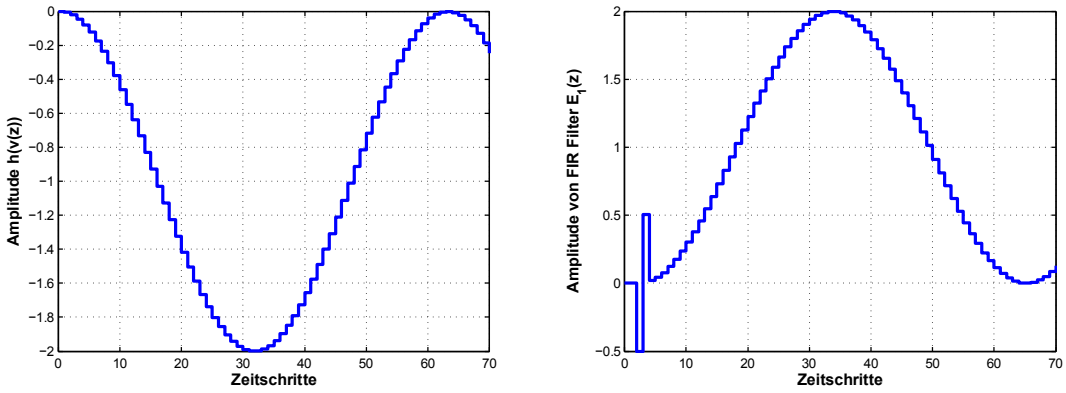


Abbildung K.3: Links: Sinusoidale Dynamik von $v(z)$. Um die Optimierungsbedingung zu erfüllen ist $h(v(z)) \leq 0$. Rechts: Impulsantwort des FIR Filters $E_1(z)$ mit sinusiodalen Koeffizienten ($B_k = -0.5$). Ab Zeitschritt 3 ist sie immer positiv.

terantwort immer positiv. $E_2(z)$ wird in der Kaskade somit positiv angeregt und divergiert gegen $+\infty$.

K.2 Zur Instabilität von $E_2(z)$

Mit der dreifachen Polstelle bei $z_P = 1$ ist $E_2(z)$ instabil. Da $z_P = 1 = \exp(-i\theta)$ einen Phasenwinkel von $\theta = 0$ hat, oszilliert die Impulsantwort jedoch nicht. Deshalb bestimmt allein das Vorzeichen des Eingangssignals, ob $E_2(z)$ gegen $+\infty$ oder $-\infty$ divergiert.

Die Auswirkungen der Divergenz auf das Rechnersystem werden durch den Gain Scheduler unter Kontrolle gehalten. Ein negativ divergierendes Signal reduziert den Regelungsfehler $E(w) > 0$. Der Gain Scheduler schaltet auf das stabile Führungsverhalten sobald $E(w) \leq 0$ erreicht wird. Die unendlichen Filterantworten von $E_2(z)$ werden somit auf endliche Werte begrenzt. Zum Umschaltzeitpunkt ist der Regelungsfehler *vollständig* kompensiert. Das Rechnersystem beendet die Ruhephase und tritt in die Arbeitsphase ein.

K.3 Optimales Fehlersignal $E(z)$

$v(z) = 0$ ist die einzige und optimale Lösung. Mit $D(z) = 0$ in der Ruhephase erfährt $v(z) = 0$ auch keine weitere Störung. In Gleichung K.1 wird daher eine Polstelle des Nennerterms *perfekt* ausgelöscht. Dies bezeichnet man als Zero-Pole-Cancellation. Damit berechnet sich das Fehlersignal zu

$$\begin{aligned} E(z) &= -\frac{(-B_k + v(z))z + B_k}{(z-1)^3} \\ &= \frac{B_k(z-1)}{(z-1)^3} \\ &= \frac{B_k}{(z-1)^2} \ \left(= V_k^{\text{soll}}\right) \end{aligned}$$

Das optimale Verhalten der Fehlerkompensation entspricht dem Verhalten der Betriebsdauersollfunktion.

In vielen praktischen Systemen unterliegt die Dynamik kleinen Schwankungen, die durch die Systemgleichungen nicht erfasst werden. Eine perfekte Auslöschung ist daher oftmals nicht möglich. Das Rechnersystem in der Ruhephase unterliegt Dynamikschwankungen durch Selbstentladungs-, Temperatur- oder Lagerungseffekte, die in der Systemgleichung des PControl Regelkreises vernachlässigt werden. Sie integrieren sich jedoch viel langsamer auf als die Betriebsdauersollfunktion. Wäre das Gegenteil der Fall, dann wäre die geforderte Betriebsdauer größer als die Zeit bis zur vollständigen Selbstentladung. Diese Konfiguration macht das Rechnersystem nicht mehr praktikabel einsetzbar. Es ist daher akzeptabel, das Verhalten des Rechnersystems für die Zeitdauer der Ruhephase als *konstant* mit $v(z) = 0$ zu betrachten.

Literaturverzeichnis

[1] Weiser, M.: The computer for the 21st century. Scientific American **265**(3) (September 1991) 66–75

[2] Weiser, M., Brown, J.S.: Designing calm technology. PowerGrid Journal **1**(2) (1996) 94–110

[3] Want, R., Schilit, W.N., Adams, N.I., Gold, R., Petersen, K., Goldberg, D., Ellis, J.R., Weiser, M.: The parctab ubiquitous computing experiment. Mobile Computing (1996) 45–102

[4] Matt, J., Kohlbecker, M.: Artefakte für das AwareOffice. Studienarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2005)

[5] Kidd, C., Orr, R., Abowd, G., Atkeson, C., Essa, I., Macintyre, B., Mynatt, E., Starner, T., Newstetter, W.: The aware home: A living laboratory for ubiquitous computing research. Cooperative Buildings. Integrating Information, Organizations and Architecture (1999) 191–198

[6] Decker, C., Riedel, T., Beigl, M., Sa de Souza, L., Spiess, P., Muller, J., Haller, S.: Collaborative business items. Intelligent Environments, 2007. IE 07. 3rd IET International Conference on (Sept. 2007) 40–47

[7] Decker, C., Beigl, M., Krohn, A., Robinson, P., Kubach, U.: eseal - a system for enhanced electronic assertion of authenticity and integrity. (2004) 254–268

[8] Beigl, M.: Kommunikation in interaktiven Räumen. Dissertation, Shaker Verlag (2001)

[9] Weiser, M., Brown, J.S.: The coming age of calm technolgy. (1997) 75–85

[10] Fischer, M., Kroehl, M.: Remembrance camera. Studienarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH (March 2006)

[11] Tolmie, P., Pycock, J., Diggins, T., MacLean, A., Karsenty, A.: Unremarkable computing. In: CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM (2002) 399–406

[12] Norman, D.A.: The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution. Volume 1 of MIT Press Books. The MIT Press (January 1999)

[13] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGPLAN Not. **35**(11) (2000) 93–104

[14] Horton, M., Culler, D., Pister, K., Hill, J., Szewczyk, R., Woo, A.: Mica: the commercialization of microsensor motes. Sensors **19**(4) (2002) 40–48

[15] Beutel, J., Kasten, O., Ringwald, M.: Btnodes - applications and architecture compared. in Holger Karl, Adam Wolisz (Ed.), TKN Technical Report TKN-03-012, 1. GI/ITG KuVS Fachgespräch Sensornetze, Technical University Berlin, Telecommunication Networks Group (July 2003)

[16] Baar, M., Will, H., Blywis, B., Hillebrandt, T., Liers, A., Wittenburg, G., Schiller, J.: The scatterweb msb-a2 platform for wireless sensor networks. Technical Report TR-B-08-15, Freie Universität Berlin, Department of Mathematics and Computer Science, Institute for Computer Science, Telematics and Computer Systems group, Takustraße 9, 14195 Berlin, Germany (09 2008)

[17] Polastre, J., Szewczyk, R., Culler, D.: Telos: enabling ultra-low power wireless research. In: IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks, Piscataway, NJ, USA, IEEE Press (2005)

[18] Beigl, M., Gellersen, H.: Smart-its: An embedded platform for smart objects. In: Smart Objects Conference (SOC 2003). (2003) 15–17

[19] Decker, C., Krohn, A., Beigl, M., Zimmer, T.: The particle computer system. In: IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks, Piscataway, NJ, USA, IEEE Press (2005) 62

[20] Beigl, M., Krohn, A., Riedel, T., Zimmer, T., Decker, C., Isomura, M.: The upart experience: The upart experience. In: IPSN '06: Proceedings of the 5th international conference on Information processing in sensor networks, New York, NY, USA, ACM (2006) 366–373

[21] Tuulari, E., Ylisaukko-oja, A.: Soapbox: A platform for ubiquitous computing research and applications. In: Pervasive '02: Proceedings of the First International Conference on Pervasive Computing, London, UK, Springer-Verlag (2002) 125–138

[22] Can Filibeli, M., Ozkasap, O., Reha Civanlar, M.: Embedded web server-based home appliance networks. J. Netw. Comput. Appl. **30**(2) (2007) 499–514

[23] Gumstix, I.: gumstix - dream, design, deliver. http://www.gumstix.com/ (2009) [Online; Zugriff 12.Mai 2009].

[24] Lifton, J., Feldmeier, M., Ono, Y., Lewis, C., Paradiso, J.A.: A platform for ubiquitous sensor deployment in occupational and domestic environments. In Abdelzaher, T.F., Guibas, L.J., Welsh, M., eds.: IPSN, ACM (2007) 119–127

[25] Beigl, M., Krohn, A., Zimmer, T., Decker, C.: Typical sensors needed in ubiquitous and pervasive computing. In: in Proceedings of the First International Workshop on Networked Sensing Systems (INSS '04). (2004) 153–158

[26] Winograd, T.: Architectures for context. Human-Computer Interaction **16** (2001) 401–419

[27] Hartmann, B., Doorley, S., Klemmer, S.R.: Hacking, mashing, gluing: Understanding opportunistic design. IEEE Pervasive Computing **7**(3) (2008) 46–54

[28] Tanenbaum, A.S., van Steen, M.: Verteilte Systeme. 1. auflage edn. PEARSON STUDIUM (2003)

[29] Microsystems, S.: Nfs: Network file system protocol specification (1989)

[30] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C.: Coda: A highly available file system for a distributed workstation environment. IEEE Transactions on Computers **39**(4) (1990) 447–459

[31] Cohen, B.: Incentives build robustness in bittorrent (2003)

[32] Moore, K.: The lotus notes storage system. In: SIGMOD Conference. (1995) 427–428

[33] McKinley, P.K., **S. Masoud Sadjadi**, Kasten, E.P., Cheng, B.H.C.: A taxonomy of compositional adaptation. Technical Report MSU-CSE-04-17, Department of Computer Science, Michigan State University, East Lansing, Michigan (May 2004)

[34] Sadjadi, S.M., McKinley, P.K.: A survey of adaptive middleware. Technical Report MSU-CSE-03-35, Computer Science and Engineering, Michigan State University, East Lansing, Michigan (December 2003)

[35] Schmeck, H.: Organic computing - a new vision for distributed embedded systems. Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on (18-20 Mai 2005) 201–203

[36] Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1) (2003) 41–50

[37] Picioroaga, F., Bechina, A., Brinkschulte, U., Schneider, E.: Osa+ real-time middleware, results and perspectives. Object-Oriented Real-Time Distributed Computing, 2004. Proceedings. Seventh IEEE International Symposium on (12-14 May 2004) 147–154

[38] Pacher, M., von Renteln, A., Brinkschulte, U.: Towards an organic middleware for real-time applications. In: ISORC '06: Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), Washington, DC, USA, IEEE Computer Society (2006) 400–407

[39] Uwe Brinkschulte, Mathias Pacher, A.v.R.: Selbst-organisierende middleware für eingebettete echtzeitsysteme. In: doIT Forschungstag, dPunkt Verlag (2006)

[40] Uwe Brinkschulte, Alexander von Renteln, M.P.: A scheduling strategy for a real-time dependable organic middleware. In: Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2006) 339–348

[41] Li, K., Shor, M., Walpole, J., Pu, C., Steere, D.: Modeling the effect of short-term rate variations on tcp-friendly congestion control behavior. In: Proceedings of the 2001 American Control Conference, Arlington, Virginia. (June 2001)

[42] Benmohamed, L., Meerkov, S.M.: Feedback control of congestion in packet switching networks: the case of a single congested node. IEEE/ACM Trans. Netw. **1**(6) (1993) 693–708

[43] Hollot, C., Misra, V., Towsley, D., Gong, W.B.: A control theoretic analysis of red. INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE **3** (2001) 1510–1519 vol.3

[44] Sha, L., Liu, X., Lu, Y., Abdelzaher, T.: Queueing model based network server performance control. In: RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02), Washington, DC, USA, IEEE Computer Society (2002) 81

[45] Parekh, S., Gandhi, N., Hellerstein, J., Tilbury, D., Jayram, T., Bigus, J.: Using control theory to achieve service level objectives in performance management. Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on (2001) 841–854

[46] Diao, Y., Hellerstein, J.L., Parekh, S.: Optimizing quality of service using fuzzy control. In: DSOM '02: Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, London, UK, Springer-Verlag (2002) 42–53

[47] Zhang, R., Lu, C., Abdelzaher, T.F., Stankovic, J.A.: Controlware: A middleware architecture for feedback control of software performance. In: ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02), Washington, DC, USA, IEEE Computer Society (2002) 301

[48] Stankovic, J.A., He, T., Abdelzaher, T., Marley, M., Tao, G., Son, S., Lu, C.: Feedback control scheduling in distributed real-time systems. In: RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01), Washington, DC, USA, IEEE Computer Society (2001) 59

[49] P. Albertos, M.V., Crespo, A.: Embedded control systems: the control kernel. In: 10th International Conference on Computer Aided Systems Theory, Universidad de las Palmas de Gran Canaria. (June 2005)

[50] Unbehauen, R.: Systemtheorie 1. 8.auflage edn. Oldenbourg Wissenschaftsverlag GmbH (1997)

[51] Ljung, L.: System identification (2nd ed.): theory for the user. Prentice Hall PTR, Upper Saddle River, NJ, USA (1999)

[52] Decker, C., Riedel, T., Peev, E., Beigl, M.: Adaptation of on-line scheduling strategies for sensor network platforms. Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference on (Oktober 2006) 534–537

[53] Berchtold, M., Riedel, T., Decker, C., Beigl, M., Bittel, C.: Quality of location: Estimation, system integration and application. INSS 2008. Fifth International Conference on Networked Sensing Systems (17-19 June 2008)

[54] Stankovic, J.A., Ramamritham, K., Spuri, M.: Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms. Kluwer Academic Publishers, Norwell, MA, USA (1998)

[55] Buttazzo, G.C.: Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Kluwer Academic Publishers, Norwell, MA, USA (1997)

[56] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)

[57] Leung, J., Kelly, L., Anderson, J.H.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Inc., Boca Raton, FL, USA (2004)

[58] Jensen, E., Locke, C., Tokuda, H.: A time driven scheduling model for real-time operating systems. In: In Proceedings IEEE Real-Time Sytems Symposium, pages 112–122, 1985. (1985)

[59] Locke, C.D.: Best-effort decision-making for real-time scheduling. PhD thesis, Pittsburgh, PA, USA (1986)

[60] Levis, P., Sharp, C.: Tep106: Schedulers and tasks. Online: http://www.tinyos.net/tinyos-2.x/doc/html/tep106.html, Zugriff: April 2008

[61] Beigl, M., Krohn, A., Zimmer, T., Decker, C., Robinson, P.: Awarecon: Situation aware context communication (2003)

[62] Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM **20**(1) (1973) 46–61

[63] Hofmeijer, T., Dulman, S., Jansen, P., Havinga, P.: Ambientrt - real time system software support for data centric sensor networks. Intelligent Sensors, Sensor Networks and Information Processing Conference, 2004. Proceedings of the 2004 (14-17 Dec. 2004) 61–66

[64] Barry, R.: FreeRTOS - a free RTOS for small embedded real time systems. http://www.freertos.org/ (April 2006)

[65] Shift-Right Technologies: eXtreme Minimal Kernel (xmk) - a free real time operating system for microcontrollers. http://www.shift-right.com/xmk/ (April 2006)

[66] Stajano, F., Anderson, R.: The grenade timer: Fortifying the watchdog timer against malicious mobile code. In: 7th International Workshop on Mobile Multimedia Communications (MoMuC 2000), Waseda, Tokyo, Japan (2000)

[67] Han, C.C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A dynamic operating system for sensor nodes. In: Proceedings of MobiSys '05, New York, NY, USA, ACM Press (2005) 163–176

[68] Dunkels, A., Gronvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: First IEEE Workshop on Embedded Networked Sensors. (2004)

[69] Combaz, J., Fernandez, J.C., Lepley, T., Sifakis, J.: Qos control for optimality and safety. In: EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software, New York, NY, USA, ACM (2005) 90–99

[70] Wust, C.C., Steffens, L., Bril, R.J., Verhaegh, W.F.J.: Qos control strategies for high-quality video processing. In: ECRTS '04: Proceedings of the 16th Euromicro Conference on Real-Time Systems, Washington, DC, USA, IEEE Computer Society (2004) 3–12

[71] Lu, C., Stankovic, J.A., Son, S.H., Tao, G.: Feedback control real-time scheduling: Framework, modeling, and algorithms*. Real-Time Syst. **23**(1-2) (2002) 85–126

[72] Jeffay, K., Stanat, D., Martel, C.: On non-preemptive scheduling of period and sporadic tasks. Real-Time Systems Symposium, 1991. Proceedings., Twelfth (4-6 Dec 1991) 129–139

[73] Buttazzo, G.C., Sensini, F.: Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. IEEE Trans. Comput. **48**(10) (1999) 1035–1052

[74] Tia, T.S., Liu, J.W.S., Shankar, M.: Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. Real-Time Syst. **10**(1) (1996) 23–43

[75] Berchtold, M., Decker, C., Riedel, T., Zimmer, T., Beigl, M.: Using a context quality measure for improving smart appliances. In: ICDCSW '07: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops, Washington, DC, USA, IEEE Computer Society (2007) 52

[76] Berchtold, M., Riedel, T., Beigl, M., Decker, C.: Awarepen - classification probability and fuzziness in a context aware application. In: 5th International Conference on Ubiquitous Intelligence and Computing (UIC 2008). (2008) 647–661

[77] Beigl, M., Zimmer, T., Krohn, A., Decker, C., Robinson, P.: Creating ad-hoc pervasive computing environments. In: Pervasive 2004 Advances in Pervasive Computing, Vienna, Austria (Arpil 2004) 377–381

[78] Sayer, A.: Service-orientiertes Betriebssystem für ubiquitäre Sensorsysteme. Diplomarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2007)

[79] Tijms, H.C.: A First Course in Stochastic Models. Wiley, Chichester (2003)

[80] Hamann, C.J.: Real-time scheduling, abschnitt 2-1 probabilistische modelle. Vorlesung (Juli 2008)

[81] Marrón, P.J., Gauger, M., Lachenmann, A., Minder, D., Saukh, O., Rothermel, K.: Flexcup: A flexible and efficient code update mechanism for sensor networks. In Römer, K., Karl, H., Mattern, F., eds.: European Conference on Wireless Sensor Networks (EWSN). Volume 3868 of Lecture Notes in Computer Science., Springer (2006) 212–227

[82] Reijers, N., Langendoen, K.: Efficient code distribution in wireless sensor networks. In: WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications, New York, NY, USA, ACM (2003) 60–67

[83] Arnold, A.: Ein objektorientierter Ansatz zur Programmierung von Sensorknoten. Diplomarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2007)

[84] AT&T Research Labs (Initial), O.S.: Graphviz, homepage. http://www.graphviz.org/ (2008) [Online; Zugriff 15.Juli 2008].

[85] Iskenderoglu, Y.E.: Rapid Prototyping in Sensornetzen durch Simulationsverfeinerung. Diplomarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2006)

[86] Notheis, Simon: Kontextsensitive Anwendungen im Büroumfeld (2004)

[87] Decker, C., Beigl, M., Riedel, T., Krohn, A., Zimmer, T.: Buffer feedback scheduling: Runtime adaptation of ubicomp applications. In: Ubiquitous Computing Symposium (UCS). (2006) 254–269

[88] Lehoczky, J., Sha, L., Ding, Y.: The rate monotonic scheduling algorithm: exact characterization and average case behavior. Real Time Systems Symposium, 1989., Proceedings. (Dec 1989) 166–171

[89] Park, M.: Non-preemptive fixed priority scheduling of hard real-time periodic tasks. In: International Conference on Computational Science (4). (2007) 881–888

[90] Sathaye, S.S., Katcher, D.I., Strosnider, J.K.: Fixed priority scheduling with limited priority levels (1995)

[91] Wind River Systems: Vxworks. Online: http://www.windriver.com/products/vxworks/ (August 2008)

[92] Sun Microsystems: Chorusos. Online: http://www.sun.com/software/chorusos/index.xml, Zugriff: 08/2008

[93] Wind River Systems: Wind river real-time core. Online: http://www.windriver.com (August 2008)

[94] Wind River Systems: Rtlinuxfree. Online: http://www.rtlinuxfree.com (August 2008)

[95] Open Software Foundation, C.: Real-time mach (rt-mach) ntt release home page. Online: http://www.rtmach.org/ (August 2008)

[96] Tokuda, H., Kotera, M.: A real-time tool set for the arts kernel. Real-Time Systems Symposium, 1988., Proceedings. (Dec 1988) 289–299

[97] Tokuda, H., Mercer, C.W.: Arts: a distributed real-time kernel. SIGOPS Oper. Syst. Rev. **23**(3) (1989) 29–53

[98] Buttazzo, G., Di Natale, M.: Hartik: a hard real-time kernel for programming robot tasks with explicit time constraints and guaranteed execution. Robotics and Automation, 1993. Proceedings., 1993 IEEE International Conference on (May 1993) 404–409 vol.2

[99] Stankovic, J., Ramamritham, K.: The spring kernel: a new paradigm for real-time systems. Software, IEEE **8**(3) (May 1991) 62–72

[100] Stankovic, J.A., Ramamritham, K.: The spring kernel: a new paradigm for real-time operating systems. SIGOPS Oper. Syst. Rev. **23**(3) (1989) 54–71

[101] Günter, A.: Minimalistische kerne für sensorknoten. Buch ISSN: 1432-7864, Telecooperation Office, Fakultät für Informatik, Universität Karlsruhe (2006)

[102] Kuschel, D., Moczarski, S.: pico]os - rt operating system. Online: http://picoos.sourceforge.net, Zugriff: August 2008

[103] Olsonet Communications Corporation.: Picos. Online: http://www.olsonet.com/Documents/PicOS201.pdf, Zugriff: August 2008 (Dezember 2006)

[104] Han, C.C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: Sos: A dynamic operating system for sensor nodes. In: MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services, New York, NY, USA, ACM (2005) 163–176

[105] Dunkels, A., Grönvall, B., Voigt, T.: Contiki - a lightweight and flexible operating system for tiny networked sensors. In: 29th Annual IEEE International Conference on Local Computer Networks (LCN'04). (2004) 455–462

[106] Zürich., E.: Btnodes - a distributed environment for prototyping ad hoc networks. Online: http://www.btnode.ethz.ch/,Zugriff:August2008

[107] Jansen, P.G., Mullender, S.J., Havinga, P.J.M., Scholten, J.: Lightweight edf scheduling with deadline inheritance. Technical Report TR-CTIT-03-23, University of Twente, Enschede (May 2003)

[108] Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., Zheng, H.: Heterogeneous concurrent modeling and design in java (volume 1: Introduction to ptolemy ii). Technical report, Electrical Engineering and Computer Sciences University of California at Berkeley, Technical Report No. UCB/EECS-2007-7 (January 2007)

[109] Sayer, A.: Serviceorientiertes Prozessframework für Sensorknoten. Studienarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2006)

[110] Reed, D.P.: That sneaky exponential— beyond metcalfe's law to the power of community building. Context Magazine (Spring 1999)

[111] Decker, C., Spiess, P., Souza, L.M.S.D., Beigl, M.: Coupling enterprise systems with wireless sensor nodes: Analysis, implementation, experiences and guidelines. In: In Pervasive Technology Applied @ PERVASIVE. (2006)

[112] Culler, D.E., Hill, J., Buonadonna, P., Szewczyk, R., Woo, A.: A network-centric approach to embedded software for tiny devices. In: EMSOFT '01: Proceedings of the First International Workshop on Embedded Software, London, UK, Springer-Verlag (2001) 114–130

[113] Krohn, A., Beigl, M., Decker, C., Zimmer, T.: ConCom - A language and Protocol for Communication of Context. ISSN 1432-7864 2004/19, Institut für Telematik, Universität Karlsruhe (TH) (2004)

[114] Raghunathan, V., Srivastava, M.B., Shaw, B.: Energy-aware wireless microsensor networks. In: IEEE Signal Processing Magazine. Volume 19. (Mrz. 2002) 40–50

[115] Cristian, F.: Probabilistic clock synchronization. Distributed Computing **3**(3) (1989) 146–158

[116] Gusella, R., Zatti, S.: The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd. Software Engineering, IEEE Transactions on **15**(7) (Jul 1989) 847–853

[117] Langendoen, K.: Medium access control in wireless sensor networks. In Wu, H., Pan, Y., eds.: Medium Access Control in Wireless Networks, Volume II: Practice and Standards. Nova Science Publishers, Inc. (2007)

[118] Ye, W., Heidemann, J., Estrin, D.: An energy-efficient mac protocol for wireless sensor networks. In: Proceedings 21st International Annual Joint Conference of the IEEE Computer and Communications Societies, New York, New York, USA (2002)

[119] Ye, W., Heidemann, J., Estrin, D.: Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Trans. Netw. **12**(3) (2004) 493–506

[120] van Dam, T., Langendoen, K.: An adaptive energy-efficient mac protocol for wireless sensor networks. In: SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems, New York, NY, USA, ACM (2003) 171–180

[121] Ye, W., Silva, F., Heidemann, J.: Ultra-low duty cycle mac with scheduled channel polling. In: SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems, New York, NY, USA, ACM (2006) 321–334

[122] Hoesel, L.v., Havinga, P.: A lightweight medium access protocol (LMAC) for wireless sensor networks. In: INSS04, Tokyo, Japan (June 2004)

[123] Coleri-Ergen, S., Varaiya, P.: Pedamacs: Power efficient and delay aware medium access protocol for sensor networks. IEEE Trans. on Mobile Computing **5**(7) (July 2006) 920–930

[124] Rajendran, V., Obraczka, K., Garcia-Luna-Aceves, J.: Energy-efficient, collision-free medium access control for wireless sensor networks. In: SenSys03, Los Angeles, CA (November 2003) 181–192

[125] Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, New York, NY, USA, ACM (2004) 95–107

[126] El-Hoiydi, A., Decotignie, J.D.: Wisemac: an ultra low power mac protocol for the downlink of infrastructure wireless sensor networks. In: ISCC '04: Proceedings of the Ninth International Symposium on Computers and Communications 2004 Volume 2 (ISCC"04), Washington, DC, USA, IEEE Computer Society (2004) 244–251

[127] El-Hoiydi, A.: Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In: IEEE International Conference on Communications (ICC), New York (April 2002)

[128] Rhee, I., Warrier, A., Aia, M., Min, J.: Z-MAC: a hybrid MAC for wireless sensor networks. In: SenSys05, San Diego, CA (2005) 90–101

[129] Zheng, T., Radhakrishnan, S., Sarangan, V.: Pmac: an adaptive energy-efficient mac protocol for wireless sensor networks. In: IPDPS05, Denver, CO (April 2005)

[130] Halkes, G., Langendoen, K.: Crankshaft: An energy-efficient MAC-protocol for dense wireless sensor networks. In: EWSN07, Delft, The Netherlands (January 2007)

[131] Peev, E.: Verteilte Koordination in eingebetteten Ubicomp Systemen. Studienarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2007)

[132] Krohn, A., Zimmer, T., Beigl, M., Decker, C., (teco, T.O., Karlsruhe, U.: Collaborative sensing in a retail store using synchronous distributed jam signalling. In: 3rd International Conference on Pervasive Computing, volume 3468 of Lecture Notes in Computer Science, Springer-Verlag (2005) 237–254

[133] Nath, S., Gibbons, P.B., Seshan, S., Anderson, Z.R.: Synopsis diffusion for robust aggregation in sensor networks. SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems (2004) 250–262

[134] Zimmer, T.: Verbesserung der Kontexterkennung in Ubiquitären Informationsumgebungen. Dissertation, Universität Braunschweig (2007)

[135] Fichter, P.D.K.: Zukunftsmarkt energieeffiziente rechenzentren. Technical report, Borderstep Institut für Innovation und Nachhaltigkeit, Berlin, Postfach 1406, 06844 Dessau-Roßlau (Dezember 2007)

[136] Bellosa, F., Weissel, A., Waitz, M., Kellner, S.: Event-driven energy accounting for dynamic thermal management. In: Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'03), New Orleans, LA (September 2003)

[137] Microchip: Pic18f8722 family data sheet. Online: http://www.microchip.com, Zugriff: 08/2008

[138] Kellner, S., Bellosa, F.: Energy accounting support in tinyos. In: GI/ITG KuVS Fachgespräch Systemsoftware und Energiebewusste Systeme. Number 2007-20, Karlsruhe, Germany, Fakultät für Informatik, Universität Karlsruhe (TH) (October 2007) 17–20 Interner Bericht.

[139] Doyle, M., Fuller, T.F., Newman, J.S.: Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. Journal of the Electrochemical Society **140**(6) (1993) 1526–1533

[140] LaFollette, R.M., Bennion, D.: Design fundamentals of high power density, pulsed discharge, lead-acid batteries. ii modeling. Journal of the Electrochemical Society **137**(12) (1990) 3701–3707

[141] LaFollette, R.M., Harb, J.N.: Mathematical model of the discharge behavior of a spirally wound lead-acid cell. Journal of the Electrochemical Society **146**(3) (1999) 809–818

[142] Newman, J.S.: Fortran programs for simulation of electrochemical systems. http://www.cchem.berkeley.edu/~jsngrp/ (1999) [Online; Zugriff 23.Oktober 2008].

[143] Linden, D., Reddy, T.B., eds.: Handbook of Batteries. 3. ed edn. McGraw-Hill (2002)

[144] Pedram, M., Wu, Q.: Battery-powered digital cmos design. In: DATE '99: Proceedings of the conference on Design, automation and test in Europe, New York, NY, USA, ACM (1999) 17

[145] Benini, L., Bruni, D., Macii, A., Macii, E., Poncino, M.: Discharge current steering for battery lifetime optimization. IEEE Trans. Comput. **52**(8) (2003) 985–995

[146] Manwell, J., McGowan, J.: Lead acid battery storage model for hybrid energy systems. Journal of Solar Energy Science and Engineering **50**(5) (1993) 399–405

[147] Hageman, S.: Simple pspice models let you simulate common battery types. EDN (1993) 117–129

[148] Hyman, E., Spindeler, W., F., F.J.: Phenomenological discharge voltage model for lead-acid batteries. In: Proceedings of the AIChE Meeting. (November 1986) 78–86

[149] Benini, L., Castelli, G., Macii, A., Macii, B., Scarai, R.: Battery-driven dynamic power management of portable systems. System Synthesis, 2000. Proceedings. The 13th International Symposium on (2000) 25–30

[150] Martin, T.L.: Balancing batteries, power, and performance: system issues in cpu speed-setting for mobile computing. PhD thesis, Pittsburgh, PA, USA (1999) Adviser-Daniel P. Siewiorek.

[151] Zinniker, R.: Zinnikers batterie und akku seiten, die ideale batterie (2008) [Online; Zugriff 23.Oktober 2008].

[152] DALLAS Semiconductor: High-precision li+ battery monitor with 2-wire interface. http://datasheets.maxim-ic.com/en/ds/DS2764.pdf (2008) [Online; Zugriff 22.Oktober 2008].

[153] DALLAS Semiconductor: 1-cell or 2-cell stand-alone fuel gauge ic. http://datasheets.maxim-ic.com/en/ds/DS2781.pdf (2008) [Online; Zugriff 22.Oktober 2008].

[154] Chiasserini, C.F., Rao, R.R.: Pulsed battery discharge in communication devices. In: MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, New York, NY, USA, ACM (1999) 88–95

[155] Benini, L., Castelli, G., Macii, A., Scarsi, R.: Battery-driven dynamic power management. IEEE Des. Test **18**(2) (2001) 53–60

[156] Benini, L., Castelli, G., Macii, A., Macii, E., Poncino, M., Scarsi, R.: Extending lifetime of portable systems by battery scheduling. In: DATE '01: Proceedings of the conference on Design, automation and test in Europe, Piscataway, NJ, USA, IEEE Press (2001) 197–203

[157] Chiasserini, C.F., Rao, R.: Energy efficient battery management. Selected Areas in Communications, IEEE Journal on **19**(7) (Jul 2001) 1235–1245

[158] Li, T., John, L.K.: Run-time modeling and estimation of operating system power consumption. In: SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, New York, NY, USA, ACM (2003) 160–171

[159] Rakhmatov, D., Vrudhula, S., Chakrabarti, C.: Battery-conscious task sequencing for portable devices including voltage/clock scaling. In: DAC '02: Proceedings of the 39th conference on Design automation, New York, NY, USA, ACM (2002) 189–194

[160] Liu, J., Chou, P., Bagherzadeh, N., Kurdahi, F.: Power-aware scheduling under timing constraints for mission-critical embedded systems. Design Automation Conference, 2001. Proceedings (2001) 840–845

[161] Subramanian, R., Fekri, F.: Sleep scheduling and lifetime maximization in sensor networks: fundamental limits and optimal solutions. Information Processing in Sensor Networks, 2006. IPSN 2006. The Fifth International Conference on (April 2006) 218–225

[162] Cao, Q., Abdelzaher, T., He, T., Stankovic, J.: Towards optimal sleep scheduling in sensor networks for rare-event detection. In: IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks, Piscataway, NJ, USA, IEEE Press (2005) 4

[163] Degesys, J., Rose, I., Patel, A., Nagpal, R.: Desync: Self-organizing desynchronization and tdma on wireless sensor networks. Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on (April 2007) 11–20

[164] Astrom, K.J., Wittenmark, B.: Adaptive Control. Addison-Wesley Publishing Company Inc., Second Edition. Reading (1995)

[165] Tröster, F.: Steuerungs-und Regelungstechnik für Ingenieure. Oldenbourg, München (2005)

[166] Melischko, S.: Adaptive Regelung des Einsatzes von Energieressourcen auf Sensorknoten. Studienarbeit, TecO, Institut für Telematik, Universität Karlsruhe (TH) (2008)

[167] Sandeep Dutta (Initial), O.S.: Sdcc small devices c compiler, homepage. http://sdcc.sourceforge.net/ (2009) [Online; Zugriff 06. Mai 2009].

[168] Green, M., Limebeer, D.J.N.: Linear robust control. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1995)

[169] Flajolet, P., Martin, G.N.: Probabilistic counting algorithms for data base applications. J. Comput. Syst. Sci. **31**(2) (1985) 182–209

[170] Sadjadi, S.M.: Transparent shaping of existing software to support pervasive and autonomic computing. PhD thesis, East Lansing, MI, USA (2004) Adviser-Philip K. Mckinley.

[171] Uwe Brinkschulte, Alexander von Renteln, M.P.: Towards an artificial hormone system for self-organizing real-time task allocation. In: Lecture Notes in Computer Science, Springer Berlin / Heidelberg (2007) 339–347

[172] Lachenmann, A., Marrón, P.J., Minder, D., Rothermel, K.: Meeting lifetime goals with energy levels. In: 5th International Conference on Embedded Networked Sensor Systems (SenSys). (November 2007) 131–144

Lebenslauf

Persönliche Daten	
Name	Decker
Vorname	Christian
Geburtsdatum	20. Oktober 1977
Geburtsort	Zeulenroda
Staatsangehörigkeit	deutsch

Ausbildung	
von 1984 bis 1990	Polytechnische Oberschule „Friedrich Schiller“
von 1990 bis 06/1996	Friedrich-Schiller-Gymnasium Zeulenroda
von 07/1996 bis 07/1997	Zivildienst in der Katholischen Gemeinde Zeulenroda
von 10/1997 bis 09/2002	Studium der Informatik an der Universität Karlsruhe

Beruflicher Werdegang	
von 10/1997 bis 09/1999	wissenschaftliche Hilfskraft am Rechenzentrum der Universität Karlsruhe, Aufgabe: universitätsweiter LaTeX Support
von 10/1999 bis 03/2001	wissenschaftliche Hilfskraft am Telecooperation Office (TecO), Universität Karlsruhe(TH)
von 04/2001 bis 06/2001	Internship bei STARLAB, Brüssel
von 07/2001 bis 03/2002	wissenschaftliche Hilfskraft am Telecooperation Office (TecO) Universität Karlsruhe(TH)
von 04/2002 bis 09/2002	Zeitangestellter bei SAP Corporate Research in Karlsruhe
seit 10/2002	Wissenschaftlicher Mitarbeiter am Telecooperation Office (TecO), Universität Karlsruhe(TH)
seit 03/2006	Gruppenleitung der Abteilung Telecooperation Office (TecO), Universität Karlsruhe(TH)

Abschlüsse	
06/1996	Allgemeine Hochschulreife
09/2002	Informatik Diplom (Dipl.Inform.) an der Universität Karlsruhe

Karlsruhe, den 18. Mai 2009

Christian Decker