Methodik zur durchgängigen Entwicklung verteilter Systeme mit Echtzeitbedingungen für Rundrufnetze

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Marc Schanne

aus Rülzheim, Rheinland-Pfalz

Tag der mündlichen Prüfung:	8. November 2007
Erster Gutachter:	Prof. Dr. Walter F. Tichy
Zweiter Gutachter:	Prof. Dr. Uwe Brinkschulte

Kurzfassung

Moderne Sensoren und Aktoren verfügen oft über eigene Prozessoren zur Verarbeitung. Die Rechenleistung dieser Prozessoren wächst. Mit dem technischen Fortschritt wird es möglich, Arbeit von Applikationen auf solche Rechnerknoten zu verteilen. Verteilung ist deshalb auch für die Entwicklung von sicherheits- und geschäftskritischen (eingebetteten) Systemen interessant, um komplexe Applikationslogik effizient ausführen zu können.

Die Einhaltung von Echtzeitbedingungen ist für solche Systeme notwendig. Echtzeitfähige Kommunikationsnetze oder Feldbussysteme mit Rundruflogik werden dazu in der Praxis eingesetzt. Ihre Nutzung ist mit unterschiedlichen Betriebssystembibliotheken und -schnittstellen möglich. Diese Routinen werden auch bei der Entwicklung verteilter Applikationen in Hochsprachen und objektorientierten Laufzeitumgebungen verwendet. Für die Entwicklung werden Konzepte der physikalischen Kommunikation eingesetzt. Plattform- oder bibliotheksunabhängige Entwurfsabstraktionen für verteilte Systeme sind selten verfügbar. Echtzeitbedingungen in aktuellen Implementierungen werden außerdem meist durch die enge Kopplung innerhalb der verteilten Applikationen erreicht.

In dieser Dissertation wird eine unterstützende Entwicklungsmethodik vorgestellt, die beiden Problemen begegnet. Sie vereinfacht Entwurf, Implementierung und Analyse verteilter Systeme. Für die Entwicklung werden Erfahrungen objektorientierter Analyse und Designs genutzt, um die Logik zur Verarbeitung der Nachrichten von Programmcode für die Kommunikation und die Bereitstellung von Echtzeit zu trennen. Die Arbeit definiert dazu eine deskriptive Entwurfsmethode, die mittels einer Rahmenarchitektur leicht wieder verwendbare Komponenten bei der Verarbeitung erlaubt.

Voraussetzung für die Arbeit ist eine objektorientierte Laufzeitumgebung, die Kontrollfäden mit Echtzeitverhalten bereitstellt. Diese muss auf den einzelnen Rechnerknoten den Zugriff auf ein echtzeitfähiges Netz mit den

Funktionen einer Transportschicht[1] bieten. Die verwendete Rahmenarchitektur beschreibt einen asynchronen Publiziere-/Abonniere-Nachrichtendienst, der die lose Kopplung der Applikationskomponenten ermöglicht.

Mit dieser Rahmenarchitektur definiert die Arbeit ein Konzept von Nachrichtenkanälen zur applikations- und plattformunabhängigen Beschreibung verteilter Systeme. Ein Kanal beschreibt dabei das Verhalten beim Nachrichtenaustausch und Echtzeitbedingungen des verteilten Systems in der lokalen Verarbeitung. Die Verwaltung der erforderlichen Rahmenarchitektur ist dezentral und ebenfalls verteilt. Asynchron ist die Abstimmung in der Applikationslogik. Der Nachrichtenaustausch wird mit Warteschlangen organisiert, und die verteilte Applikation wird nicht durch das Warten auf Nachrichten blockiert.

Um den Einsatz der Rahmenarchitektur zu erleichtern, generiert die Methodik Programmcode für die Kommunikation in der Applikation direkt aus der Entwurfsbeschreibung. Bei Systemen mit harten Echtzeitbedingungen kann außerdem automatisch eine Modellbeschreibung für die statische Analyse von Ablaufplänen der Echtzeitkontrollfäden erzeugt werden. Dieses Modell repräsentiert die in der Rahmenarchitektur verwendeten Kontrollfäden. Für ein vollständiges Modell der einzelnen Knoten muss der Entwickler nur Anteile der Applikationslogik ergänzen. Er kann sich dabei auf die Funktionalität der Applikation konzentrieren.

Die in der Arbeit entwickelte Methodik mit Beschreibung, Generierung und Analyse unterstützt die Entwicklung verteilter Systeme mit Echtzeitbedingungen. Da die mit der Entwicklungsmethodik angestrebte Prozessverbesserung schwierig zu messen ist, wird mit der Implementierung eines Prototyps die Eignung von Rahmenarchitektur und Methodik an repräsentativen Beispielszenarien untersucht und die Machbarkeit nachgewiesen. Die Evidenz des Ansatzes wird sowohl für weiche als auch für harte Echtzeitbedingungen dargestellt. Zur Evaluierung werden Prototypimplementierungen mit Echtzeit-Java für die plattformunabhängige Rahmenarchitektur eingeführt. Mit dem Einsatz von UDP/IP-Gruppenruf für weiche Echtzeitbedingungen und mit der Anbindung des Modellierungs- und Analysewerkzeugs MAST zur statischen Ablaufplananalyse bei harten Echtzeitbedingungen wird die Eignung der Entwicklungsmethodik für beide Implementierungen belegt.

Das Ergebnis der Arbeit ist eine Methodik, die die Entwicklung verteilter Systeme mit Echtzeitbedingungen unterstützt. Ausgehend von einer echtzeitfähigen objektorientierten Laufzeitumgebung und einem Kommuni-

[1] Schicht 4 des ISO/OSI-Referenzmodells für Kommunikation

kationsnetz mit Rundruf, vereinfacht sie den Entwicklungsprozess durchgängig.

Danksagung

Der Prozess beim Schreiben einer Dissertation ähnelt einem Softwareentwicklungsprozess. Durch frühzeitige Modellanalysen und Codedurchsichten kann die Ausarbeitung an Qualität gewinnen. Ohne Kollegen, ohne gemeinsame Forschung im FZI oder in Forschungsprojekten, ohne Unterstützung durch Studenten als Hiwis oder in Studien- und Diplomarbeiten ist diese Qualitätsprüfung nicht möglich!

Die Anforderungsanalyse begann noch als Diskussion meiner FZI-Arbeit mit meinem Kollegen Dr.-Ing. James Hunt. Seine Überlegungen zu einem asynchronen Nachrichtendienst in eingebetteten Echtzeitsystemen habe ich hier umgesetzt und erweitert. Durch seinen Einsatz im EU-Projekt HIDOORS wurde es möglich, dass ich den Nachrichtendienst auch im 2. EU-Forschungsprojekt zu High Integrity Java (HIIJA) weiterentwickeln und weitergehende Erfahrungen sammeln konnte. Ich habe von seinen Erfahrungen profitiert und die Ausgestaltung meines Arbeitsbereichs als Dissertation wurde so erst möglich. Für die Unterstützung meiner Tätigkeit durch die Europäische Kommission möchte ich allen Kollegen und Studenten in Karlsruhe und Europa danken. Ergebnisse sind erfolgreiche Projektentwicklungen — EU-Forschungsprojekte und eben diese Dissertation.

Meinem Referenten Prof. Dr. rer. nat. Walter F. Tichy danke ich für die Betreuung des gesamten Verfahrens. Er und sein Fachbereich am FZI haben mir die nötige Freiheit gegeben meine Arbeit erfolgreich abzuschließen. Geholfen dabei haben auch mein Korreferent Prof. Dr. rer. nat. Uwe Brinkschulte und meine beiden sachverständigen Prüfer Prof. Dr.-Ing. Frank Bellosa und Prof. Dr. rer. nat. Peter H. Schmitt, ihnen gilt mein aufrichtiger Dank.

Bei ständig verschobenen Projektabschlussfristen erleben außer Kollegen auch Freunde und Familie den Alltag der Entwicklung. Für ihr Verständnis in den Jahren auf dem Weg zum Projekt “Doktor” bedanke ich mich bei jedem und jeder herzlich.

Eine will ich aber ganz besonders hervorheben, das ist Eka, sie war und ist mir jedes der drei: Kollegin als Studentin am FZI, Freundin in gemeinsamen Spiele- oder Filmabenden, Familie als meine Frau und Mutter unserer Tochter Lea.

Danke allen, die mich unterstützt und bis zum erfolgreichen Projektabschluss begleitet haben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Bei sicherheits- oder geschäftskritischen Systemen wächst die Notwendigkeit zur Verteilung der Verarbeitung auf mehrere Rechnerknoten. Durch Ausnutzung der Rechenleistung moderner Sensoren und Aktoren wird es möglich, Daten näher an der Quelle bzw. der Senke zu verarbeiten. Die Entwicklung komplexerer, verstärkt fehlerresistenter und skalierbarer verteilter Applikationen wird möglich. Durch objektorientierte Techniken möchte man diese Vorteile mit der Möglichkeit einer Wiederverwendung von Komponenten, objektorientiertem Design und besserer Wartbarkeit verbinden.

Obwohl der Einsatz objektorientierter Techniken bei eingebetteten Systemen und Applikationen mit Echtzeitbedingungen nicht unumstritten ist [120, 58], kann die Entwicklung und Wiederverwendung von getesteten Komponenten auch bei diesen Systemen durch Nutzung von Techniken objektorientierter Analyse und Designs profitieren. Auf Basis einer objektorientierten Rahmenarchitektur wird die erforderliche Zuverlässigkeit und Fehlerfreiheit dieser Systeme leichter garantiert.

Forschung und Analysen der letzten Jahre prognostizierten im Umfeld eingebetteter Systeme verschiedene Trends [71, 59]. Das anhaltende Wachstum in diesem Bereich und der zukünftig wichtiger werdende Umgang mit Verteilung und Heterogenität in diesen echtzeitfähigen und eingebetteten Systemen werden besonders hervorgehoben. Die Europäische Union verfolgt eine strategische Ausrichtung in diesem Themenbereich, um die Entwicklung von eingebetteten Systemen für den Alltag zu fördern [53].

Die Beteiligung an zwei europäischen Forschungsprojekten zu objektorientierten, eingebetteten Systemen mit Echtzeitbedingungen [52, 54] erlaubt es, die verwendete Softwarearchitektur in realistischen Testszenarien zu

prüfen und zu bewerten. Die in dieser Dissertation vorgestellte Entwicklungsmethodik nutzt die Erfahrungen aus der Softwareentwicklung in den Forschungsprojekten und vereinfacht die Entwicklung durchgängig von Entwurf bis Analyse einer verteilten Applikation.

1.1 Aufgabenstellung

Ziel der Arbeit ist die Definition einer Entwicklungsmethodik für verteilte Systeme mit Echtzeitbedingungen, die asynchrone Nachrichtenkommunikation in Rundrufnetzen ausnutzt. Die vorgeschlagene Rahmenarchitektur basiert dabei auf einer objektorientierten Laufzeitumgebung. Es werden Anforderungen an diese definiert und plattformunabhängige Konzepte für ihren Entwurf beschrieben. Durch Kombination von Softwareentwurfsmustern wird dabei eine asynchrone und direkte Publiziere-/ Abonniere-Nachrichtenkommunikation ermöglicht.

Für die Validierung des Ansatzes wurde eine Implementierung auf Basis von Java mit der Echtzeiterweiterung RTSJ verwendet. Durch minimalen Protokollaufwand sollen die Leistungscharakeristiken der zugrunde liegenden physikalischen Netze voll genutzt werden. Mit der als Prototyp entwickelten *EventChannelNetwork API* [32] wurde der Einsatz dieses Kommunikationsverfahrens in zwei europäischen Forschungsprojekten untersucht [52, 54].

Der hier vorgeschlagene Ansatz einer Nachrichtenkommunikation ist mit Hinblick auf existierende Netze und Feldbussysteme im Umfeld eingebetteter Systeme entworfen. Das implementierte Rahmenwerk erlaubt die Entwicklung verteilter Anwendungen und nutzt die in diesen Systemen üblichen Rundrufnetzinfrastrukturen.

Zentrale Komponente des Kommunikationsdienstes sind Nachrichtenkanäle, die Anforderungen an aktive Objekte (Kontrollfäden für die Verarbeitung der ausgetauschten Nachrichten) und Nachrichtentypen definieren. Ein Kanal gruppiert Nachrichten und stellt das Thema der Publiziere-/ Abonniere-Kommunikation dar.

Ausgehend von dieser Idee vertritt und belegt die Arbeit folgende Thesen.

> **These 1** Ohne Protokollaufwand für die Synchronisierung zwischen Kommunikationspartnern vereinfacht asynchrone Kommunikation mit Nachrichtenkanälen die Entwicklung eingebetteter verteilter Systeme auf Basis gängiger Rundrufnetz- und Feldbus-Infrastrukturen.

Die vorgestellte Methodik verwendet eine Rahmenarchitektur und abstrahiert von technischen Netzen. Bei der Implementierung werden die Eigenschaften verbreiteter Vernetzungstechniken von eingebetteten Systemen ausgenutzt. Mit Nutzung von Nachrichtenkommunikation nach dem Publiziere-/Abonniere-Muster zur Vernetzung verteilter, objektorientierter Komponenten können plattformunabhängig Systeme entworfen werden.

> **These 2** Mit Verwendung einer deskriptiven Programmiermethode wird die Verwendung asynchroner Nachrichtenkommunikation weiter unterstützt und die Wiederverwendung der entwickelten Komponenten einfach möglich.

Durch die Beschreibung von Komponenten und Kanälen in einer XML-Datei kann Standard-Programmcode bei der Kommunikation verwendet oder durch einen Generator erzeugt werden. Mit der Beschreibung von Kanälen können Echtzeitbedingungen bei der Verarbeitung von Nachrichten in Komponenten festgelegt werden.

> **These 3** In verteilten Systemen – mit statisch vorhersagbaren harten Echtzeitbedingungen – ermöglicht die deskriptive Programmiermethode die Erzeugung eines Modells, das den Kommunikationsaufwand in einem Kommunikationsknoten beschreibt und Echtzeitbedingungen für Verarbeitungsfristen festlegt.

Das Modell der Ablaufkoordinierung, das die generierten Kontrollfäden für die Kommunikation enthält, unterstützt den Entwickler bei der Analyse der verteilten Anwendung für jeden einzelnen Kommunikationsknoten. Er kann diese Teilmodelle entsprechend erweitern.

Obwohl die drei Thesen die Möglichkeiten und Einschränkungen eingebetteter Echtzeitsysteme beachten, bleibt der Architekturvorschlag und vorgestellte Prototyp allgemein einsetzbar. Aus der Vorstellung der verwendeten Echtzeiterweiterung für Java werden allgemeine Systemanforderungen abgeleitet. Mit der Beschreibung von notwendigen Parametern der Ausführungsumgebung, wie zum Beispiel der Ablaufkoordinierung nach festen Prioritäten, wird die vorgestellte Methodik auch auf andere Programmierplattformen, Sprachen und Bibliotheken übertragbar.

Der entwickelte Prototyp der Rahmenarchitektur basiert zwar auf Java-Technologie, aber es werden immer allgemeine Lösungen vorgestellt. Für

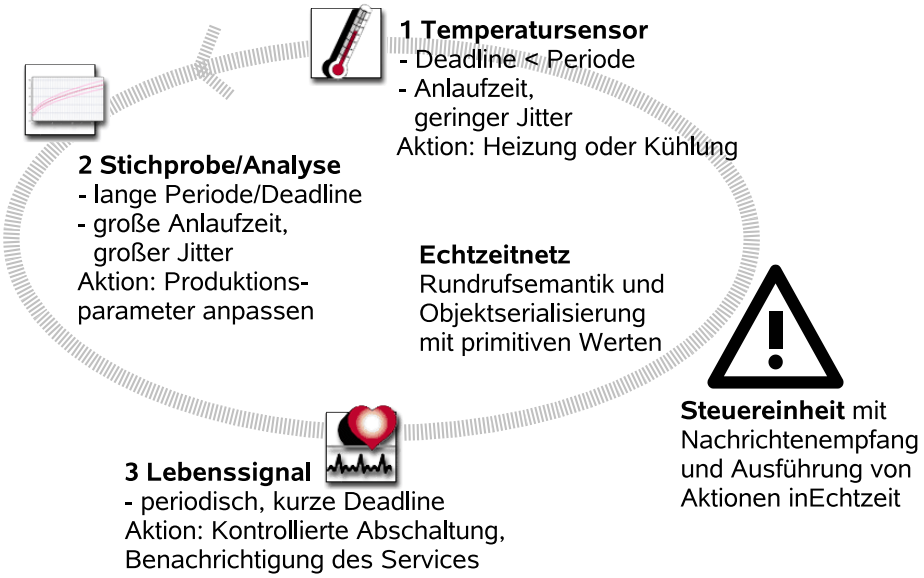

Abbildung 1.1: Beispiel einer Steuereinheit für ein Produktionssystem

die Analyse der Echtzeitbedingungen wird das Modellierungs- und Analysewerkzeug MAST verwendet. Das hierfür erzeugte Modell der Kontrollfäden kann analog auch mit anderen Werkzeugen verwendet werden, die ein lineares Modell für ereignisgesteuerter Systeme bei der Systemanalyse einsetzen.

1.2 Allgemeingültiges Beispielszenario

Zur besseren Erklärung der nachfolgenden Modelle und Analysen führt dieser Abschnitt eine einfache Steuerung für ein verteiltes Produktionssystem als allgemeingültiges Beispielszenario ein. Das konkrete Beispiel aus der Automation schränkt die Methodik und den Einsatz der Rahmenarchitektur nicht ein, erlaubt aber, besonders für harte Echtzeitbedingungen, eine Fokussierung auf die mit der Methodik verbundenen Unterstützung bei der statischen Analyse der entwickelten Applikation.

Verwendet wird ein Kommunikationsnetz mit Rundruf. Dieses physikalische Netz muss die Einhaltung harter Echtzeitbedingungen beim Versand von Nachrichten unterstützen. Das Produktionssystem besteht aus vier unterschiedlichen Kommunikationsknoten. Der Temperatursensor (1), eine Komponente für die Analyse von Stichproben (2) und ein periodi-

scher Signalgeber als Lebenssignal (3) werden vom 4. Knoten, der zentralen Steuereinheit, beobachtet. Abbildung 1.1 zeigt diese Kontrollschleife. Die Sensoren (1-3) erzeugen Nachrichten, die von der Steuereinheit mit Echtzeitbedingungen verarbeitet werden. Im Folgenden soll die Verarbeitung der unterschiedlichen Nachrichten in der Steuereinheit kurz erläutert werden.

Grundlage für die weiteren Überlegungen sind drei Nachrichtenkanäle, die mit den drei Sendern im Produktionssystem verknüpft sind. Die Steuereinheit muss Nachrichten dieser Kanäle empfangen und ihre Verarbeitung in entsprechenden Fristen garantieren. Der Einfachheit halber werden alle Kontrollinformation und Messwerte über ein physikalisches Netz gesendet und behandelt.

- Die periodische Messung mit einem Temperatursensor (1) ist die einfachste Komponente in der verteilten Anwendung. Die Steuereinheit reagiert auf Nachrichten des Sensors und regelt die notwendige Heizung bzw. Kühlung des Systems.
- Eine zufällige Stichprobe wird von Sensor 2 mit einer periodischen Wiederholung genommen. Mit ihr wird die Korrektheit der Fertigung überprüft. Falls die so ermittelten Werte von den Sollwerten zu weit abweichen, muss die Steuereinheit Korrekturen veranlassen. Da die Qualität der Produkte vor der endgültigen Verpackung und Auslieferung noch einmal überprüft wird, erfolgt diese Prüfung in einer größeren Periode und die tatsächliche Bereitstellung der Daten kann zeitlich stärker schwanken (großer Jitter[1]).
- Das periodisch erwartete Lebenssignal der Verarbeitungsstraße (Sensor 3) ist sicherheitskritischer. Bei Nichteintreffen muss mit sofortigem und geordnetem Systemstop reagiert und eine Nachricht an den Service gesendet werden.

Die Tabelle 1.1 listet die entsprechenden Systemparameter auf. Neben Perioden, Bearbeitungsfristen, ersten Eintreffzeitpunkten und Jitter-Werten, die die Sender der Kanäle definieren, werden auch die schlechtesten anzunehmenden Bearbeitungszeiten (“worst case execution time”, WCET) für Nachrichten der drei Kanäle in der Steuereinheit aufgeführt. Für die Ermittlung der WCET-Werte ist es notwendig, die Verarbeitungslogik des jeweiligen Kanals mit einem entsprechenden Werkzeug für die WCET-Analyse zu überprüfen.

[1] d.h. maximales Zittern bei der Bestimmung des genauen Eintreffzeitpunktes

Nachrichten-kanäle für ...	Temperatur (K1)	Stichprobe (K2)	Lebenssignal (K3)
Periode	600	1200	400
Bearbeitungsfrist[A]	300	1200	200
Erster Eintreff-zeitpunkt	100	350	100
Jitter[B]	50	150	50
WC Bearbei-tungszeit[C]	50	250	50

[A] Diese Frist berechnet sich ab dem tatsächlichen Eintreffzeitpunkt einer Nachricht.
[B] Aus Periode und erstem Eintreffzeitpunkt berechnete weitere Eintreffzeitpunkte können maximal um diesen Jitter (das maximale Zittern) verzögert sein.
[C] Bearbeitungszeit für Nachrichten dieses Kanals in der Steuereinheit im schlimmsten anzunehmenden Fall ("worst case", WC).

Tabelle 1.1: Zeitangaben für Nachrichtenkanäle und ihre Behandlerlogik in der Steuereinheit

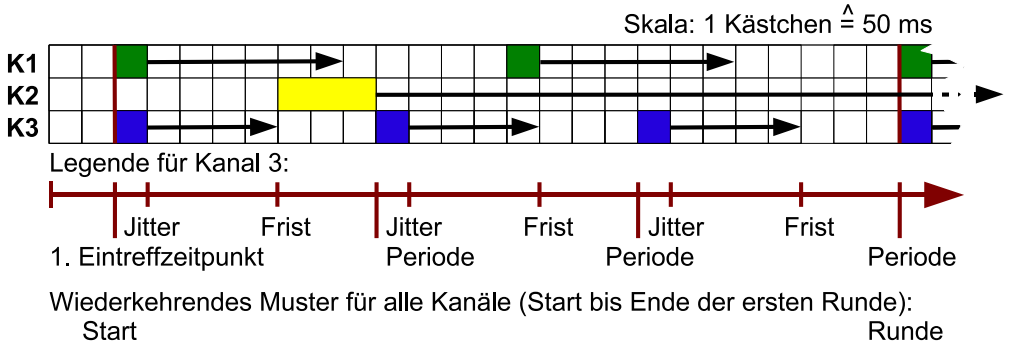

Abbildung 1.2: Nachrichtenkanäle mit allen Eintreffzeitpunkten, Ungenauigkeiten und Fristen in der ersten Runde

Die Abbildung 1.2 illustriert grafisch wie die drei Nachrichtenkanäle nebenläufigen Empfang und Verarbeitung erfordern. Die Grafik zeigt die Eintreffzeitpunkte, Ungenauigkeiten (Zittern oder Jitter) und Bearbeitungsfristen für Nachrichten innerhalb einer Runde. Die Beschriftung an der abgebildeten X-Achse bezieht sich auf die Lebenssignalnachrichten des 3. Kanals. Die dargestellte (erste) Runde beginnt mit dem frühesten ersten Eintreffzeitpunkt aller Nachrichtenkanäle und hat als Länge (Start bis Runde) das kleinste gemeinsame Vielfache (kgV) der Perioden aller beobachteten Nachrichtenkanäle. Nachfolgende Runden zeigen ein sich wiederholendes Empfangsmuster. Der Eintreffzeitpunkt innerhalb der Runde wird für jeden Kanal mit dem Beginn eines Pfeils markiert. Die Länge des Pfeils ist abhängig von der maximal erlaubten Bearbeitungsfrist[2]. Die Ungenauigkeit wird als Bereich des Zitterns (Jitter) beim Eintreffen einer Nachricht durch eine farbige Zeitspanne dargestellt. Der tatsächliche Beginn der Nachrichtenverarbeitung und der Ablauf der Frist beginnt in diesem Bereich.

1.3 Beiträge dieser Arbeit

In dieser Arbeit wird eine durchgängige Methodik zur Entwicklung verteilter Systeme mit Echtzeitbedingungen beschrieben. Die mit der Methodik verwendete Rahmenarchitektur basiert dafür auf asynchroner Nachrichtenkommunikation. Die Methodik unterstützt die Entwicklung bei Entwurf, Implementierung, sowie Analyse der verteilten Systeme. Echtzeitbedingungen werden lokal für jeden Knoten des verteilten Systems gefordert und der Einsatz einer Rahmenarchitektur stellt planbare Voraussetzungen für Empfang und Verarbeitung in jedem Knoten sicher.

Für die Garantie von Echtzeitbedingungen sind echtzeitfähige physikalische Netze erforderlich. Ihre Garantien werden ohne großen Verwaltungsaufwand an die verteilte Applikation weitergegeben. Die Arbeit selbst umfasst keine Kommunikationsprotokolle auf dieser Ebene. Die im Folgenden einzeln beschriebenen Beiträge betreffen nur die Anwendungsschicht und den Softwareentwurf des verteilten Systems.

[2]Für den Nachrichtenkanal K2 ist das Ende des Pfeils außerhalb des dargestellten Zeitfensters in der nächsten Runde.)

1.3.1 Asynchrone Nachrichtenkommunikation in Echtzeitbedingungen

Die asynchrone Form der Kommunikation widerspricht im ersten Moment der Anforderung von Echtzeitbedingungen. Das Wort "asynchron" ist das gegenteilige Wort von "synchron", welches sich aus den altgriechischen Wortstämmen "syn" (mit, gemeinsam) und "chronos" (Zeit) ableitet. Im ursprünglichen Sinn bedeutet "asynchron" also "nicht gleichzeitig" oder "zeitlich nicht übereinstimmend" und diese fehlende Abstimmung passt wenig zu Echtzeitbedingungen, die zeitliche Abstimmung garantieren. Bei asynchroner Kommunikation, wie sie in dieser Dissertation verwendet wird, muss die fehlende zeitliche Abstimmung durch eine entsprechende Kommunikationsinfrastruktur für den Nachrichtenaustausch bereitgestellt werden.

Echtzeitbedingungen in verteilten Applikationen werden bisher oft durch synchrone Interaktion und enge Kopplung zwischen Komponenten garantiert. Der Einsatz asynchroner Nachrichtenkommunikation erfordert eine Infrastruktur für Empfang und Versand. Dies bedeutet zusätzlichen Aufwand bei der Interaktion, aber es erlaubt Vereinfachungen beim Entwurf der Komponenten. Kommunikation mit Nachrichten ermöglicht die Entwicklung skalierbarer verteilter Systeme. Durch Replizierung und Austausch von Komponenten kann leicht das Verhalten des Gesamtsystems verändert und verbessert werden.

Das Empfangen von Nachrichten erfordert Arbeit des Prozessors. Für Echtzeitsysteme muss diese eingeplant werden. Um den Empfang garantieren zu können, werden Kontrollfäden mit höchster Priorität eingesetzt, die von anderen Aktivitäten des Prozessors nicht blockiert werden können. Da die Verarbeitung planbar sein muss, dürfen Nachrichten nicht beliebig eintreffen. Für die Verarbeitung in der vorgestellten Rahmenarchitektur wird deshalb mit periodischen und sporadischen[3] Kontrollfäden Rechenkapazität reserviert.

Da der Prozessor mit dem Empfang weiterer Nachrichten beschäftigt sein kann, wird in der Regel mit der Verarbeitung einer Nachricht nicht direkt begonnen. Jeder Knoten muss diese entsprechend puffern. Um dennoch die Verarbeitung innerhalb von Zeitfristen zu garantieren, erfordert der nebenläufige Empfang die Koordination zwischen Empfänger- und Verarbeitungskontrollfäden. Wenn kein geeigneter Ablaufplan existiert, der

[3] aperiodisch, aber mit Mindestzwischenzeiten

alle Echtzeitbedingungen erfüllt, muss in der Rahmenarchitektur darauf reagiert werden können.

Automatische Speicherbereinigung in einer objektorientierten Laufzeitumgebung kann unvorhersehbare Verzögerungen verursachen. Die Rahmenarchitektur verwendet für den Empfang von Nachrichten deshalb nur fest allozierte Objekte mit festgelegten Speicherkapazitäten. Dynamisches Verhalten mit der Ankündigung neuer Nachrichtenkanäle und einer nachträglichen Registrierung für den Nachrichtenaustausch ist nur mit Einschränkungen bei der Garantie von Echtzeitbedingungen möglich.

Die Rahmenarchitektur selbst ist Beitrag dieser Arbeit und mit einer prototypischen Implementierung wird die Evaluierung der eingeführten Software- und Architekturmuster belegt.

1.3.2 Deskriptive Entwurfsmethode

Der komponentenbasierte Entwurf verteilter Applikationen für diese Rahmenarchitektur erlaubt die statische Beschreibung der Komponenteninteraktion. Eine deskriptive Entwurfsmethode verteilter Systeme ist möglich.

Die Kommunikation in verteilten objektorientierten Systemen erfordert Entwicklungsaufwand bei Entwurf und Implementierung der einzelnen Komponenten. Der Einsatz einer Kommunikationsrahmenarchitektur bietet eine Standardbibliothek und geprüfte Schablonen für den Entwurf. Um den Entwicklungsaufwand weiter zu minimieren, wird die Kommunikation in XML-Dateien beschrieben und durch die Einführung von Nachrichtenkanälen abstrahiert.

Die Verwendung von Schablonen beim Softwareentwurf verführt leicht zu “copy and paste” (Kopieren und Einfügen) bei der Entwicklung neuer Komponenten. Da die Anpassung der alten Werte ein sorgfältiges Arbeiten erfordert und fehleranfällig ist, wird zusammen mit der Beschreibung der Komponenten und ihrer Kommunikation auch die Unterstützung durch die Entwicklungsumgebung angeraten.

1.3.3 Planbarkeit der Ablaufkoordinierung

Wenn für sicherheitskritische Systeme die statische Prüfung der Ablauffähigkeit notwendig ist, dann kann neben der Generierung von Programmcode für die Kommunikation aus der Beschreibung auch ein Modell für

dessen Analyse und eventuelle Zertifizierung ermittelt werden. Umfangreiche Simulationen und Tests werden eingespart.

Das erzeugte Modell der Kontrollfäden in der Kommunikationsinfrastruktur entspricht dem automatisch generierten Programmcode und der Implementierung für die Rahmenarchitektur. Der Entwickler kann die Teilmodelle um anwendungsspezifische Teile ergänzen. Mit einem Modell für jeden Knoten können die Echtzeitbedingungen des verteilten Systems für jeden Knoten verifiziert werden.

Für die Analyse selbst ist es notwendig, den generierten Programmcode, die verwendete Bibliothek, sowie die vom Entwickler geschriebenen Komponenten für die Nachrichtenverarbeitung mit einem Werkzeug zur Ermittlung möglicher Ausführungszeiten unter ungünstigsten Bedingungen ("worst case execution time", WCET) zu untersuchen. Diese Werte sind zusammen mit den beschriebenen Echtzeitverarbeitungsbedingungen Voraussetzung für die statische Ablaufanalyse der verteilten Applikation.

1.3.4 Durchgängige Methodik für Entwicklung

Neben einer Rahmenarchitektur, für die Unterstützung der Nachrichtenkommunikation auf einem für Viele-zu-Viele-Kommunikation ausgelegten Netzprotokoll, definiert die vorgestellte Methodik Anforderungen an das verwendete Laufzeitsystem. Ein Laufzeitsystem muss mehrere echtzeitfähige Kontrollfäden und eine Ablaufkoordinierung für feste Prioritäten unterstützen. Durch eine Systembeschreibung mit Angabe von Systemvoraussetzungen und Anwendungszielen definiert die Dissertation eine Entwurfsmethode, mit der die Entwicklung zuverlässiger, verteilter und sicherheits- oder geschäftskritischer Systeme vereinfacht werden soll.

Sicherheitskritische Systeme mit harten Echtzeitbedingungen erfordern oft eine statische Prüfung dieser Garantien. Die Garantie von Verarbeitungszeiten wird mit Trennung von Empfang und Verarbeitung ermöglicht. Die vorgestellte Rahmenarchitektur sieht die Beschreibung von Bedingungen und Garantien in einer XML-Datei vor, und daraus wird für (generierten) Programmcode bei der Kommunikation ein Modell für die Ablaufkoordinierung ("scheduling") in Analysewerkzeugen wie zum Beispiel MAST [72] generiert.

Die vorgestellte Rahmenarchitektur für eine Kommunikationsinfrastruktur mit Nachrichtenkanälen verbindet objektorientierte Entwicklung von Komponenten mit der Beschreibung von Nachrichtenkommunikation.

Die Rahmenarchitektur unterstützt Echtzeitbedingungen beim Systementwurf, soweit die eingesetzte Hardware und das verwendete Laufzeitsystem diese Garantien anbieten. Durch eine leichtgewichtige[4] Beispiel-Implementierung und die Nutzung von entsprechenden Entwurfsmustern wird der Entwurf mit Echtzeitbedingungen auf Anwendungsebene ermöglicht.

1.4 Gliederung der Arbeit

Diese Dissertation gliedert sich in 6 Kapitel. Ausgehend von diesem Einleitungskapitel mit Motivation, Beschreibung der Aufgabenstellung und Beiträgen der Arbeit, werden in Kapitel 2 grundlegende Begriffe, existierende Ansätze und notwendige Überlegungen für den Einsatz asynchroner Nachrichtenkommunikation eingeführt. Verwandte Arbeiten, auch aus dem Bereich der Systementwicklung und -beschreibung, werden verglichen. Eine Lösungsskizze der oben vorgestellten Thesen bietet das Kapitel 3. Kapitel 4 vertieft die notwendigen Entwurfsentscheidungen und erläutert die beispielhafte Umsetzung mit Echtzeit-Java und XML. Im Anschluss überprüft Kapitel 5 den vorgestellten Ansatz anhand konkreter allgemeingültiger Einsatzszenarien und belegt so die Evidenz der entwickelten Methodik. In Kapitel 6 wird die Dissertation mit einer Zusammenfassung der wichtigsten erreichten Ziele und einem Ausblick auf weitere mögliche wissenschaftliche Aktivitäten geschlossen.

Die folgenden Kapitel im Einzelnen:

In Kapitel 2 wird zunächst der Stand der Technik bei Systembeschreibungsverfahren zur Vereinfachung der Softwareentwicklung diskutiert. Für die Entwicklung verteilter Systeme führt dieses Kapitel Begriffe und Konzepte der Kommunikation und Synchronisation sowie entsprechender Kommunikationsdienste ein. Diese werden in Zusammenhang mit eingebetteten Systemen, Echtzeitbedingungen und möglichen Vernetzungssystemen eingeordnet. Die Vorteile asynchroner Nachrichtenkommunikation werden aus diesen Vorbetrachtungen ermittelt. Abschnitt 2.8 bietet dabei einen exemplarischen Überblick wesentlicher verwandter Kommunikationsinfrastrukturen und bewertet mögliche Alternativen.

Mit Kapitel 3 wird eine Lösungsskizze für eine notwendige Rahmenarchitektur bei der Nachrichtenkommunikation in Echtzeit näher beschrie-

[4] vollständig in Echtzeit-Java entwickelte

ben. Ausgehend von der allgemeinen Analyse von Anforderungen werden Entwurfsentscheidungen für eine objektorientierte Ablaufumgebung motiviert. Daraus ergeben sich konkrete Anforderungen an die Ablaufkoordinierung der Laufzeitumgebung und verwendete Netze. Ein plattformunabhängiges Entwurfsmuster wird dargestellt. In Verbindung mit wieder verwendbaren und austauschbaren Komponenten wird eine einheitliche Methodik für die Beschreibung der Kommunikation über Kanäle definiert, die eine statische Analyse der Ablaufkoordinierung ermöglicht.

Kapitel 4 erläutert die *EventChannelNetwork* Prototypimplementierung. Obwohl der Ansatz asynchroner Kommunikation plattformunabhängig ist, werden hier auch Anforderungen an Echtzeit-Java (RTSJ) formuliert, die für einen asynchronen Nachrichtendienst notwendig sind. Eine XML-Beschreibung wird für die deskriptive Programmierung von verteilten Systemen verwendet.

Für die Evaluierung der Rahmenarchitektur beschreibt Kapitel 5 Testszenarien für die Rahmenarchitektur und erklärt Möglichkeiten für die praktische Nutzung der vorgestellten Softwareentwicklungsmethodik verteilter Systeme. Die Rahmenarchitektur wird mit weichen und harten Echtzeitbedingungen untersucht. An allgemeingültigen Beispielen werden die Vorteile der Prototypimplementierung bei der Entwicklung verteilter Systeme deutlich.

Kapitel 6 gibt eine Zusammenfassung der Entwicklungsmethodik mit Einsatz asynchroner Kommunikation in verteilten, objektorientierten Systemen mit Echtzeitbedingungen. Einen Ausblick auf mögliche weiterführende Arbeiten über diese Dissertation hinaus wird gegeben. Ansatzpunkte ergeben sich sowohl aus der plattformunabhängigen Entwicklungsmethodik, wie aus der Nutzung der entwickelten Prototypimplementierung in zukünftigen Versionen des Echtzeit-Java Standards.

Zur besseren Übersicht in der Ausarbeitung bietet die Abbildung auf der folgenden Seite eine schematische Gliederung der einzelnen Kapitel und Hauptabschnitte. Die Abschnitte bauen inhaltlich aufeinander auf, aber für die selektive Lektüre sind wesentliche Abhängigkeiten durch Pfeilverbindungen hervorgehoben.

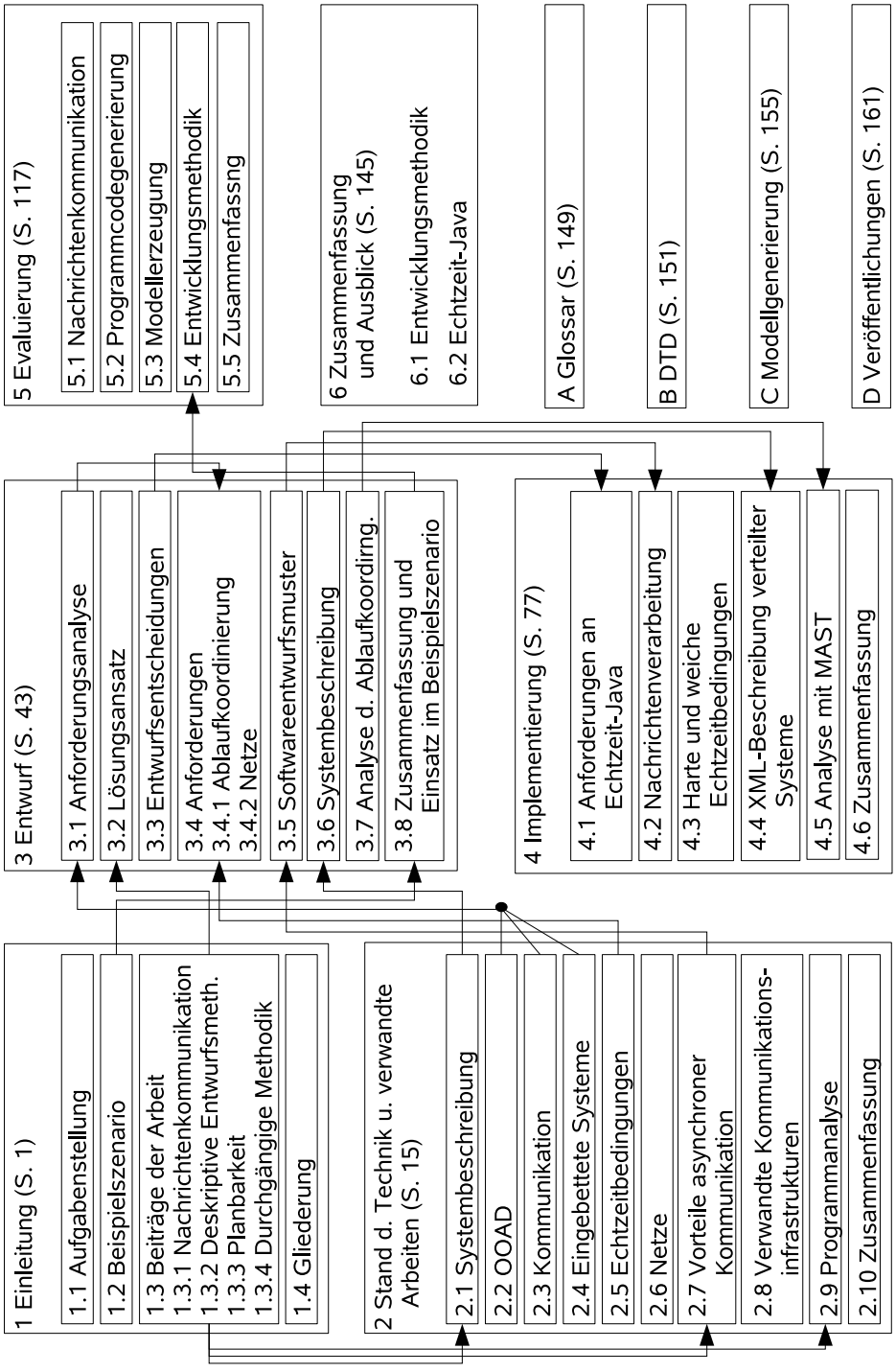

Abbildung 1.3: Gliederung und Abhängigkeiten der Ausarbeitung

Kapitel 2

Stand der Technik und verwandte Arbeiten

Der Einsatz einer Systembeschreibung bei der Entwicklung von Softwaresystemen ist grundsätzlich nicht neu. Um die Vereinfachung durch das mit der Methodik eingeführte Verfahren zu verdeutlichen wird in Abschnitt 2.1 der aktuelle Stand der Technik hierzu beleuchtet. Neben Sprachen wie der UML mit umfangreichen Darstellungsformen wird insbesondere die Verwandtschaft mit domänenspezifischen Sprachen (DSL) verdeutlicht. Von diesen Ansätzen kann besonders die objektorientierte Softwareentwicklung profitieren. Mit Techniken aus objektorientierter Analyse und Design, wie sie Abschnitt 2.2 anregt, wird die Entwicklung komponentenbasierter verteilter Systeme vereinfacht.

Um die Vorteile asynchroner Nachrichtenkommunikation mit Echtzeitbedingungen in verteilten Systemen klarer darzustellen, führt der Abschnitt 2.3 Begriffe und Konzepte aus den Bereichen Kommunikation und Synchronisation ein. Diese werden im Zusammenhang mit der Zielarchitektur eingebetteter Systeme (vgl. Abschnitt 2.4) unter Echtzeitbedingungen (vgl. Abschnitt 2.5) eingeordnet.

Basierend auf der Darstellung möglicher Kommunikationsnetze mit Unterstützung für Rundruflogik in Abschnitt 2.6 werden die Vorteile asynchroner Nachrichtenkommunikation eingeführt.

Im Abschnitt 2.8 werden die Unterschiede zu existierenden Kommunikationsinfrastrukturen beleuchtet und verwandte Arbeiten dargestellt.

Für den Einsatz mit harten Echtzeitbedingungen wird mit der Methodik die Generierung von Kommunikationsprogrammcode und einem äquivalenten Analysemodell unterstützt. Dies erfordert die Abschätzung der

Ausführungszeit im ungünstigsten Fall. In Abschnitt 2.9 wird die Nutzung solcher Werkzeuge eingeführt und ein Überblick der notwendigen Technik dargestellt.

Eine Zusammenfassung und erste Bewertung der eingeführten verwandten Arbeiten und Techniken schließt das Kapitel ab. Die Notwendigkeit der Vereinfachung des Entwicklungsprozesses verteilter Systeme wird hervorgehoben.

2.1 Vereinfachung durch Systembeschreibung

Beim Entwurf von verteilten objektorientierten Systemen bietet die "Unified Modeling Language" (UML) eine standardisierte Sprache für die Modellierung von Software. Die UML definiert eine Sprache mit Bezeichnern für die meisten Begriffe, die bei der Modellierung wichtig sind. Sie legt mögliche Beziehungen zwischen diesen Begriffen fest. Durch grafische Notationen vereinfacht die UML die Beschreibung von Systemen im Entwurf. Sie bietet eine möglichst allgemeine Modellierungsmethode für Softwaresysteme und durch die Definition von Profilen wird versucht, die UML auch für Teilbereiche zu spezialisieren. Ein Konzept für Metamodelle erlaubt diese Erweiterungen.

Für sicherheitskritische Systeme mit Echtzeitbedingungen ist hier besonders das "UML Profile for Schedulability, Performance and Time" [39] (SPT-Profil, auch bekannt als UML-RT) interessant. Mit diesem Profil wird der UML-Entwurfsansatz um Standard-Echtzeitnotationen erweitert, um Modellanalysen für Echtzeitbedingungen zu ermöglichen. Gu und andere [43] beschreiben beispielhaft die Abbildung eines Systems, modelliert mit UML-RT, auf den im Automobilbereich populären Standard für Echtzeitbetriebssysteme OSEK/VDX [2]. Da die Standardisierung von Automobilkomponenten schon länger angestrebt wird, existiert für die OSEK/VDX-Plattform auch eine eigene Beschreibungssprache zur Spezifikation von Softwarekomponenten, Hardwareanbindung und Kommunikation. Mit "OSEK Implementation Language" [85] können Softwarekomponenten der OSEK/VDX-Plattform beschrieben werden.

Aber auch UML-Standard-Diagramme, zum Beispiel für die Beschreibung von Anwendungsfällen [36] oder Komponenten [102, 103] werden bei der Entwicklung von komponentenbasierter Software für verteilte und eingebettete Echtzeitsysteme schon eingesetzt. Ansätze hier versuchen die Beschreibung von Komponenten ohne eine objektorientierte Umgebung und

Sprache zu ermöglichen und trotzdem von der mächtigen Beschreibungsform mit UML zu profitieren.

UML in der Version 2.0 unterstützt zwar durch unterschiedliche Diagramme verschiedene Schritte im Softwareentwicklungsprozess, aber die Komplexität dieser Beschreibungsformen verhindert gleichzeitig den einfachen und durchgängigen Einsatz eines Modells bei der Entwicklung. Durch die von UML unterstützen Vorgehensweisen der modellgetriebenen Architektur ("Model-Driven Architecture", MDA) und modellgetriebenen Entwicklung, die automatische Modelltransformation bis hin zur Generierung von Programmcode verspricht, soll die Softwareentwicklung vereinfacht werden. Trotz Entwicklung von Spezifikationen durch die Object Management Group (OMG) [81] ist die Unterstützung hierfür aber noch fragmentarisch [25].

Ein anderer Entwurfsansatz wird mit domänenspezifischer Sprache ("Domain-specific Language", DSL) oder domänenspezifischer Modellierung ("Domain-specific Modeling", DSM [113]) vorgegeben. Die Komplexität der UML fördert diese Entwicklung. Ausgehend vom Fachmodell eines bestimmten Problembereiches sollen Programme leichter in Software umgesetzt werden können. Lösungen für eine Domäne haben viel gemeinsam und Unterschiede lassen sich durch eine begrenzte Anzahl von Parametern und deren Abhängigkeiten definieren. Auf diese Weise entstehen einfache Programmbeschreibungen und die Umsetzung in ein ablauffähiges Computerprogramm kann automatisch erfolgen. Bei der Entwicklung dieser Beschreibungen lohnt sich der Einsatz von Software-Assistenten, die den Programmierer als Erweiterung in einer integrierten Entwicklungsumgebung (z.B. als Plug-In für Eclipse [31]) bei der Sammlung der notwendigen Parameter unterstützen.

2.2 Erfahrungen objektorientierter Analyse und Designs

Eine komponentenorientierten Beschreibung verteilter Systeme profitiert von Erfahrungen objektorientierter Analyse und Designs. Systembeschreibungen wie sie im letzten Abschnitt eingeführt werden, sind eng mit objektorientierter Softwareentwicklung [82] verbunden.

Der Einsatz einer Rahmenarchitekur ermöglicht bei der Entwicklung von gleichartigen Applikationen einer Domäne die Vorgabe eines Rahmens für ihre Implementierung. Die Entwicklung verteilter Systeme kann so zum

Beispiel durch die Trennung zwischen Kommunikation und sonstiger Verarbeitungslogik in der Applikation vereinfacht werden.

Durch Einsatz von Entwurfsmustern, wie zum Beispiel einer Fabrikmethode, kann die Erzeugung von Objekten gesteuert werden und die lose Kopplung der Systemkomponenten wird schon im Rahmen einer evtl. generierten Basis-Applikation festgelegt.

2.3 Einsatz von Kommunikation

Basis für verteilte Systeme ist die Verfügbarkeit einer Interprozesskommunikation [109]. Nebenläufige und miteinander vernetzte Dienste erlauben Systeme mit gesteigerter Performanz, mehr Zuverlässigkeit, besserer Skalierbarkeit und Kosteneffizienz [94]. Diese Vorteile im Bereich der Standard- und Unternehmenssoftware-Anwendungen sind auch für eingebettete Systeme nutzbar. In solchen Systemen werden Netze mit modernen Sensoren, Aktoren und Prozessoren eingesetzt, um sicherheits- und geschäftskritische Systeme zu verwirklichen. Für ihre Entwicklung sind unterschiedliche Formen der Vernetzung und Kommunikations- und Synchronisationsmechanismen denkbar. Dieser Abschnitt beschreibt mögliche Prinzipien und Paradigmen bei der Kommunikation mit Nachrichten und der Synchronisation zwischen unterschiedlichen vernetzten Komponenten. Die allgemein üblichen Fachbegriffe in diesem Kontext werden eingeführt.

Dieser Abschnitt basiert in leicht veränderter Form auf einem Artikel für die Ada Deutschland Tagung 2005. Die Anforderungsanalyse für asynchrone Nachrichtenkommunikation beim Entwurf von objektorientierten, verteilten Systemen unter Echtzeitbedingungen wird darin detaillierter beschrieben. Diese und andere eigene Veröffentlichungen im Rahmen der Dissertation werden in Anhang D zeitlich sortiert aufgelistet.

2.3.1 Kommunikationsformen

Ein verteiltes System basiert auf vernetzten Komponenten und Prozessoren, die untereinander Informationen per Nachrichten austauschen. Es werden Kopien von Daten im entfernten Adressraum verwendet [1].

[1] Systeme, die für den Austausch von Informationen gemeinsame Daten in einem globalen Speicher verwenden, werden in dieser Dissertation nicht weiter betrachtet.

	verbindungsorientiert	verbindungslos
synchron	synchroner entfernter Aufruf	Rendezvous
asynchron	asynchroner entfernter Aufruf	Nachrichtenversand

Tabelle 2.1: Kommunikation und Synchronisation

In Systemen mit harten Echtzeitbedingungen muss eine rechtzeitige Übermittlung garantiert werden. Protokolle, strukturiert in Schichten, erlauben die Kommunikation auf Anwendungsschicht. Der Anwendungsnutzen des Kommunikationsprotokolls unterscheidet zwei grundlegende Verfahren:

Leitungsvermittlung: Dieses Verfahren geht von dem Prinzip aus, dass für eine Kommunikation ein exklusiver Nachrichtenkanal vom Absender zum Empfänger geschaltet wird. Diese Ende-zu-Ende-Verbindung[2] kann temporär für nur eine Nachrichtenübertragung dienen. Wenn erst nach Aufbau einer Verbindung Nachrichten übertragen werden, spricht man deshalb auch von **verbindungsorientierter** Kommunikation.

Paketvermittlung: Bei diesem Verfahren wird die zu übertragende Information in einzelne Pakete geteilt. Jedes Paket enthält dabei neben dem zu übertragenden Informationsfragment die komplette Absender- und Empfängeradresse. Da für den Austausch kein exklusiver Nachrichtenkanal existieren muss, wird hier auch von **verbindungsloser** Kommunikation gesprochen. Wenn die Informationen einer Nachricht wegen ihrer Größe ohne Aufteilung in einem Paket vermittelt werden können, bietet Paketvermittlung eine effiziente Methode der Nachrichtenübertragung. Der Begriff **nachrichtenorientierte** Kommunikation wird oft äquivalent zur Bezeichnung dieser Kommunikationsform verwendet.

Verbindungsorientierte Kommunikationsdienste werden auch als **zuverlässige** Netzdienste bezeichnet. Eine **verlust-** und **verfälschungsfreie** Kommunikation kann mit entsprechendem Prüfaufwand aber auch mit Paketvermittlung erbracht werden. Verbindungslose Kommunikation kann so unterschiedliche Zuverlässigkeit mit entsprechenden Fehlersemantikklassen bieten [116]. Bei **"vielleicht"** ("maybe") findet keine Fehlerbehandlung statt, die Nachricht wird entweder gar nicht oder höchstens einmal

[2] Außer Punkt-zu-Punkt-Verbindungen umfasst dies auch Mehrpunktverbindungen für Rund- oder Gruppenrufe.

übertragen. **"Mindestens einmal"** ("at-least-once") stellt sicher, dass bei Nachrichtenverlust die Übertragung bis zum Erfolg wiederholt wird. Es erfolgt keine Filterung von Duplikaten. Auch bei **"höchstens einmal"** ("at-most-once") beruht die Fehlerbehandlung auf Wiederholung der Nachrichtenübertragung, alle Duplikate werden hier ausgefiltert. Ohne Absturz des Dienstgebers, was zum Verlust einer Nachricht führen kann, ist dies äquivalent zur Semantik von **"genau einmal"** ("exactly-once"), wobei Absturz und Wiederanlauf von Komponenten mit abdeckt und genau eine Übertragung garantiert wird.

In einer objektorientierten Anwendungsumgebung, wie zum Beispiel der virtuellen Maschine für Java, gibt es für beide Kommunikationsformen konkrete Programmbibliotheken: Der entfernte Methodenaufruf ("Remote Method Invocation", RMI, vgl. Abschnitt 2.8.1) [104] erlaubt die verbindungsorientierte Kommunikation und mit dem Java Nachrichtendienst ("Java Message Service", JMS, vgl. Abschnitt 2.8.3) [105] ist verbindungslose Kommunikation möglich. Diese Standard-Implementierungen erfüllen beide keinerlei Echtzeitbedingungen und sind nicht für eingebettete Systeme mit begrenzten Ressourcen geeignet.

2.3.2 Synchronisation

Synchronisation ist eine orthogonale Charaktereigenschaft für Kommunikation, die direkt mit der Kommunikationsform verbunden ist. Mit ihr wird beschrieben, wie entfernt ablaufende Prozesse sich bei Kommunikation untereinander verhalten. Synchrones Verhalten beschreibt den gleichzeitigen Ablauf von Aktivitäten. Für Kommunikation bedeutet dies, dass Senden und Empfangen auf Sender- und Empfängerseite gleichzeitig stattfinden.

Wenn Arbeit für synchrone Kommunikation von Prozessen, die auch sonstige Aufgaben erbringen, selbst ausgeführt wird, dann müssen diese Prozesse aufeinander "warten". Die Erbringung ihrer Aufgaben wird blockiert. Im Gegensatz dazu erlaubt asynchrone Kommunikation eine nebenläufige Ausführung der Prozesse, unabhängig von der Zeit und dem Zustand anderer Kommunikationspartner. Beim Austausch der Nachrichten erfordert dies Pufferkapazität und oft wird die eigentliche Kommunikationsarbeit dabei durch zusätzliche Hilfsprozesse erbracht. Diese Hilfsprozesse sind entweder Teil einer unterliegenden Schicht (z.B. des Betriebsystems) oder Dienste der asynchronen Kommunikationsinfrastruktur. Abbildung 2.1 zeigt beispielsweise einen Prozess A, der eine Nachricht sen-

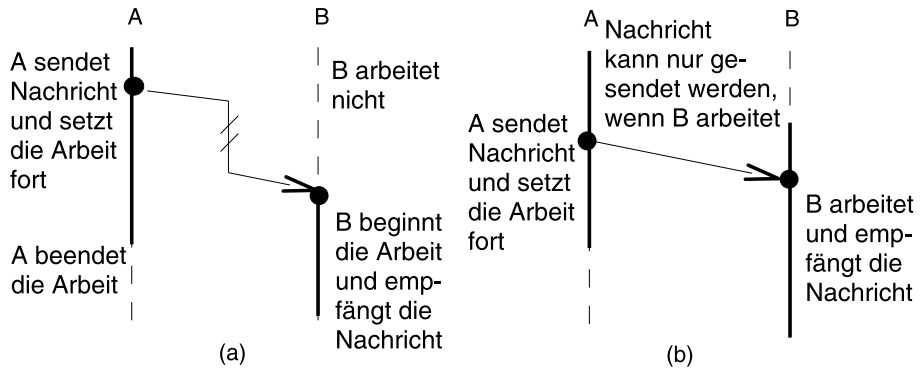

Abbildung 2.1: Persistente (a) und transiente (b) asynchrone Kommunikation

det und seine sonstige Arbeit direkt weiterführt[3]. Er wartet dabei nicht auf die erfolgreiche Übertragung der Nachricht an Prozess B und wartet insbesondere auf keine Antwort als Bestätigung.

Für die eingeführten Formen von Kommunikation ergeben sich so verschiedene Kombinationen für Kommunikations- und Synchronisationsformen, die in Tabelle 2.1 aufgeführt sind.

2.3.3 Persistenz vs. Transienz

Zusammenhängend mit asynchronem Nachrichtenversand der Komponente A zeigt Abbildung 2.1 außerdem mögliche Persistenzstrategie beim Umgang mit diesen Nachrichten.

Persistenz beim Übertragen von Nachrichten erfordert, dass gesendete Nachrichten durch eine Kommunikationsinfrastruktur bis zu ihrer Auslieferung an den Empfänger gespeichert werden (vgl. Fall (a) in Abbildung 2.1). So ist es möglich, dass der Empfänger gerade nicht aktiv empfängt, sondern andere Aufgaben bearbeitet oder nicht aktiv ist. Im Gegensatz dazu erfordert **Transienz** bei der Übertragung (vgl. Fall (b) der Abbildung2.1) keine Zwischenspeicherung von gesendeten Nachrichten.

Kriterium	Option	
Adressierung	direkt	indirekt
Blockierung	synchron	asynchron
Pufferung	ungepuffert	gepuffert, Mailbox

Tabelle 2.2: Kommunikationsmodelle

2.3.4 Kommunikationsmodelle

Bei der Kommunikation von verteilten Systemen auf Basis von Nachrichtenaustausch ohne gemeinsam benutzten globalen Speicher lassen sich nach Seck die in Tabelle 2.2 vorgestellten Kriterien und Optionen unterscheiden [96].

Bei direkter **Adressierung** spricht der Sender den Empfänger direkt und im Allgemeinen hart codiert an. Sender- und Empfängeradresse sind fest im Programm hinterlegt und die Kommunikation kann nur erfolgen, wenn Sender und Empfänger kommunikationsbereit sind. Im Unterschied dazu wird bei indirekter Adressierung über eine Empfangsstelle (Mailbox) kommuniziert.

Mit **Blockierung** lässt sich die in Abschnitt 2.3.2 beschriebene Synchronisation zwischen Kommunikationspartnern unterscheiden. Ein Sender, der synchron kommuniziert, blockiert, bis seine Nachricht über das Netz losgeschickt worden ist und eventuell eine Empfangsbestätigung vom Empfänger zurückgekommen ist. Der korrespondierende Empfänger blockiert, bis eine Nachricht eingetroffen ist. Bei asynchroner Kommunikation kann der Sender direkt nach Absenden der Nachricht weiterarbeiten, und ein Empfänger blockiert keine sonstigen Aktivitäten für deren Empfang.

Unabhängig von der Adressierungsart und dem Synchronisationsverhalten ist eine **Pufferung** auf Sender- und Empfängerseite möglich. Bei asynchroner Kommunikation weiß der Sender nicht, ob und wann die Puffer frei sind. Sind die Puffer voll, gehen Daten verloren, oder es muss blockiert werden, was einer asynchronen Kommunikation widerspricht.

Im Folgenden wird die asynchrone Option bei Blockierung einzelner Prozesse zusammen mit indirekter Adressierung und gepuffertem Nachrichtenaustausch betrachtet. Asynchrone Kommunikation erlaubt die (zeitliche) Entkopplung von Sender und Empfänger; Parallelarbeit wird unterstützt.

[3] Fall (a) und (b) unterschieden sich bei der Zwischenspeicherung der Nachricht, vgl. Abschnitt 2.3.3.

2.3.5 Publiziere-/Abonniere-Kommunikation

Eine Form der asynchronen und indirekten Kommunikation bieten Publiziere-/Abonniere-Nachrichtendienste. Kommunikation über solche Nachrichtendienste lässt sich in zwei Schritte unterteilen. Bevor der eigentliche Nachrichtenaustausch stattfindet, müssen sich die Komponenten im Netz untereinander oder bei einem Vermittlerdienst registrieren. Ein Publizist erzeugt Nachrichten, die für den oder die Abonnenten interessant sind. Oft unterstützt der Sender durch eine öffentliche Ankündigung die Registrierung der Abonnenten. Das Publiziere-/Abonniere-Kommunikationsmuster erlaubt so die Entkopplung von Anbietern und Konsumenten in **Raum** und **Zeit** [35].

- Die räumliche Entkopplung wird durch Einführung eines Dienstes für die Zustellung einer Nachricht ermöglicht. Der Publizist sendet seine Nachrichten über diesen Nachrichtendienst und der Abonnent erhält die Nachrichten indirekt zugestellt. Publizierende und abonnierende Komponenten müssen keine Referenzen zueinander besitzen.

- Die zeitliche Entkopplung erlaubt es den Kommunikationspartnern trotz unterschiedlicher Aktivitätszeiten an einer Kommunikation teilzunehmen. Mit einem persistenten Kommunikationsmechanismus ist es möglich, zu senden und zu empfangen, auch wenn der Partner nicht verbunden ist. Diese Entkopplung bezüglich Synchronisation und Gleichlauf der Kommunikationspartner erlaubt ohne Blockierung die Ausführung anderer Aktionen. Über die Verfügbarkeit einer Nachricht werden Komponenten zum Beispiel asynchron über eine Rückrufmethode informiert.

Es existieren zwei Basisverfahren für die Nachrichtenübertragung bei Publiziere-/Abonniere-Systemen: **“push”** und **“pull”**. Ohne Verlust der Allgemeingültigkeit sind beide Typen in Abbildung 2.2 ohne Entkopplung in Raum oder Zeit und mit jeweils nur einer Sender- und Empfängerinstanz dargestellt. Erst nach der Registrierung unterscheiden sich die beiden Verfahren. Beim Push-Modell werden interessanten Nachrichten vom Publizisten in die Empfangsschnittstelle des Abonnenten “übermittelt”. Das Pull-Modell agiert umgekehrt: der Abonnent “fordert” hier die Nachrichten aktiv vom Publizisten an.

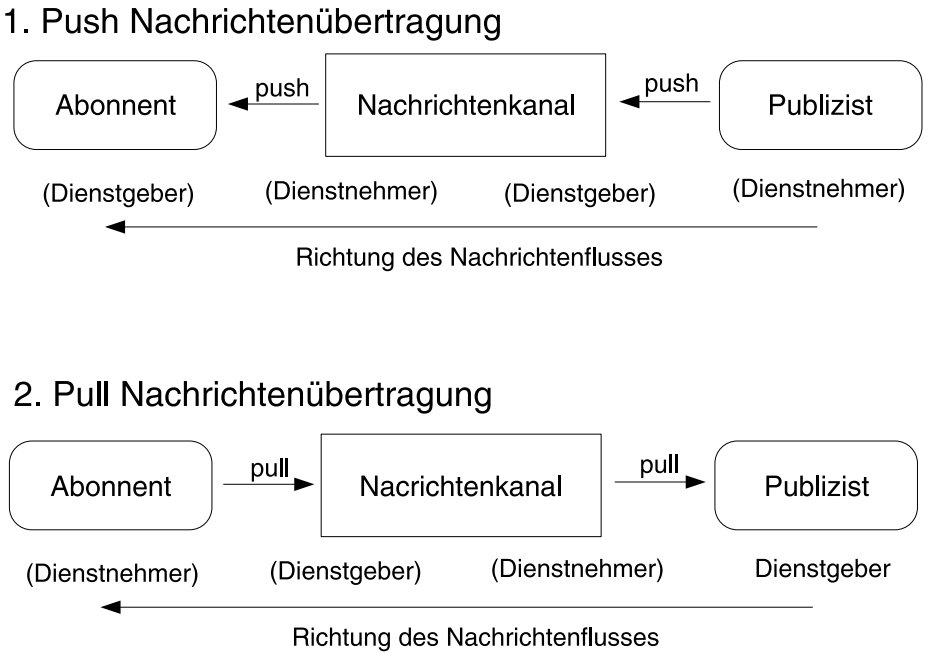

Abbildung 2.2: Nachrichtenübertragung mit Push- oder Pull-Verfahren

Eugster und andere [35] identifizieren noch eine weitere Unterscheidung und führen drei Konzepte für die Abonnierung ein: **themenbasierte**, **inhaltsbasierte** oder **typbasierte** Publiziere-/Abonniere-Systeme sind denkbar.

1. **Themenbasierte** Publiziere-/Abonniere-Kommunikation ist die früheste Form dieser Systeme und basiert auf der Notation eines Themas oder eines Titels. Themen sind analog zur Bildung von Gruppen. Mit der Registrierung des Empfängers für ein Thema erhält er alle Nachrichten dieser Gruppe ohne Filterung.

2. Publiziere/Abonniere-Kommunikation **basierend auf Inhalten** ist ein weiteres Konzept. Die zu empfangenden Nachrichten werden abhängig vom tatsächlichen Inhalt und nicht nur nach externen Eigenschaften, wie zum Beispiel der Zugehörigkeit zu einem Thema, selektiert.

3. Bei **typbasierter** Auswahl wird die Auswahl nicht vom Inhalt, sondern von der Struktur (dem Typ) einer Nachricht bestimmt. Dieses Schema ersetzt oft themenbasierte Auswahl mit einem Filter abhängig vom Typ der Nachrichten.

Für den Einsatz in sicherheits- und geschäftskritischen verteilten Systemen mit der vorgestellten Entwicklungsmethodik eignet sich die Verwendung eines themenbasierten, direkten und asynchronen Publiziere-/Abonniere-Nachrichtendienstes. Diese Systeme verlangen vorhersagbaren Kommunikationsaufwand bei Zustellung und Selektion von Nachrichten. Die asynchrone Kommunikation mit Publizisten und Abonnenten unterstützt die lose Kopplung der Komponenten. Ein direktes Adressierungsmodell vereinfacht die Zustellung der Nachrichten und die themenbasierte Abonnierung reduziert den Verwaltungsaufwand bei der Selektion im Abonnenten.

2.4 Eingebettete Systeme

Das Einsatzgebiet von sicherheits- und geschäftskritischen Systemen ist weit und umfasst sowohl sicherheitskritische Anwendungen in Automationssteuerung, Automobiltechnik und Avionik, als auch viele geschäftskritische Anwendungsfälle in Multimedia und Telekommunikation. Diese oft eingebetteten Systeme nutzen mittlerweile einen größeren Anteil der produzierten Mikroprozessoren als herkömmliche Arbeitsplatzrechner und Dienstgeber [21]. Die mit dieser Arbeit vorgestellte Methodik für die Entwicklung verteilter, objektorientierter Anwendungen mit Echtzeitbedingungen will Erkenntnisse der objektorientierten Softwareentwicklung für Arbeitsplatzsysteme auf die Softwareentwicklung bei verteilten, eingebetteten Systemen übertragen [12, 98].

Eingebettete Systeme bezeichnen in ein umgebendes technisches System eingebettete und mit diesem in Wechselwirkung stehende Computersysteme. Sie sind in einer Vielzahl von Anwendungsbereichen, sowohl in sicherheitskritischen als auch geschäftskritischen oder allgemeinen Systemen zu finden. Sie versehen ihren Dienst meist weitgehend unsichtbar im Flugzeug, Auto, Kühlschrank oder in alltäglichen Geräten der Unterhaltungselektronik. Aufgrund des Einsatzgebietes unterliegen sie häufig stark einschränkenden Randbedingungen: Minimale Kosten und damit geringer Platz-, Energie- und Speicherverbrauch sind notwendig.

Definitionen für eingebettete Systeme betonen die Integration von Hardware mit Software. Bei der Klassifikation von eingebetteten Systemen wird durch die Notwendigkeit der Reaktion auf das umgebende System die Untermenge der reaktiven Systeme identifiziert. Wenn diese Systeme auch Echtzeitbedingungen folgen, spricht man von Echtzeitsystemen (vgl. Abbildung 2.3 [95]). Ein sicherheitskritisches eingebettetes System liegt vor,

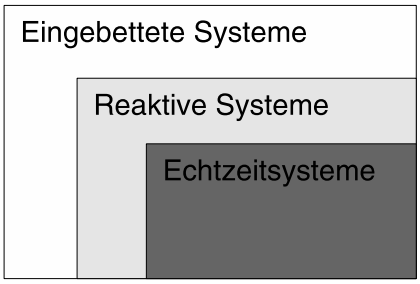

Abbildung 2.3: Echtzeitsysteme als Untermenge eingebetteter Systeme

wenn von seiner korrekten Funktionsweise Menschenleben oder die Unversehrtheit von Einrichtungen abhängen.

Kommunikation im Bereich der eingebetteten Systeme erfordert eine besonders auf die Leistungsbegrenzung und eingeschränkte Kommunikationskapazität ausgerichtete Kommunikationsinfrastruktur. Gegen zentralisierte Architekturen sprechen Faktoren wie Zuverlässigkeit und Ausfalltoleranz sowie das Problem von zentralen Fehlerpunkten, die die Funktionstüchtigkeit gesamter Anwendungen beeinträchtigen können.

Bei der Entwicklung eingebetteter Systeme muss ebenfalls auf diese Einschränkungen Rücksicht genommen werden. Liggesmeyer und Rombach [67] definieren Grundlagen, die bei der Bewertung von Softwareentwicklungsmethodiken für die im Folgenden vorgestellte Lösung beachtet werden müssen.

Interdisziplinäre Entwicklung Aus der Integration eines eingebetteten Systems in ein umgebendes technisches System resultiert auch die Integration des notwendigen Softwareentwicklungsprozesses mit unterschiedlichen anderen mechanischen, elektrischen oder Software-Entwicklungen. Die Verbindung mit Hardware bestimmt noch immer weitgehend die Entwicklung der Software [42]. In einem Softwareentwicklungsprozess, der die Aufteilung in Subsysteme unterstützt, können leichter funktionale und nichtfunktionale Anforderungen umgesetzt und integriert werden. Das Vorhandensein einer offenen Plattform würde die Integration unterschiedlicher Systembestandteile verbessern, ist aber bei eingebetteten Systemen (noch) unüblich.

Unternehmensübergreifende Entwicklung Bei Entwicklung eingebetteter Systeme sind unterschiedliche Fachdisziplinen beteiligt. Eine firmenübergreifende und verteilte Entwicklung ist notwendig. Eine Ver-

zahnung zwischen zum Beispiel Automobilhersteller und Zulieferern muss deshalb auch in der Softwarentwicklungsmethodik unterstützt werden.

Bedeutung nichtfunktionaler Eigenschaften Echtzeitverhalten, Zuverlässigkeit, Fehlertoleranz und Sicherheit sind nichtfunktionale Eigenschaften, die besonders bei eingebetteten Systemen wichtig sind. Kostenintensive und nachträglich oft unmögliche Softwarewartung in eingebetteten Systemen erfordert eine hohe Qualität eingebetteter Software. Diese kann ein Softwareentwicklungsprozess mit frühzeitiger Erkennung bzw. systematischer Vermeidung von Fehlern verbessern. Durch Analysen auf Modellebene kann dies früh im Prozess sichergestellt werden.

Wirtschaftliche Herausforderungen Entwicklungskosten und -zeit spielen bei der Entwicklung eingebetteter Systeme eine große Rolle. Die Entwicklungsmethodik muss kurze Entwicklungszeiten und außerdem kosteneffiziente Entwicklungsverfahren erlauben.

Lange Produktlebenszyklen Die im Vergleich zur technischen Weiterentwicklung elektronischer Bauteile langen Produktlebenszyklen für eingebettete Systeme[4] verursachen Kompatibilitätsprobleme bei der Entwicklung. Durch Unterstützung von Wiederverwendung, Spezialisierung und Varianten kann der Softwareentwicklungsprozess dies verhindern.

2.5 Echtzeitbedingungen

Echtzeitbedingungen werden im Rahmen von Rechen- bzw. Verteilungsmodellen für verteilte, eingebettete Systeme durch Einhaltung von Zeitgrenzen gestellt. Aus diesen Bedingungen an Rechtzeitigkeit und Erfüllung der Kommunikation in gegebenen Zeitgrenzen lässt sich nach [62] eine Klassifizierung von Echtzeit mit **harten**, **weichen** und **festen** Bedingungen ableiten. Diese Einteilung definiert sich über die schlechteste anzunehmende und die durchschnittliche Ausführungszeit.

Harte Echtzeit: Alle Antworten mit harten Zeitschranken müssen innerhalb eines spezifizierten Zeitintervalls ausgeführt sein. Wenn das Sys-

[4] 25 Jahre für Entwicklung, Produktion und anschließende Betriebs- und Servicephase z.B. im Automobilbereich [91]

tem eine Zeitschranke nicht einhält, ist das Gesamtsystem unvorschriftsmäßig.

Weiche Echtzeit: Weiche Zeitbedingungen werden durch die Einhaltung einer durchschnittlichen Antwortzeit charakterisiert. Verfehlt ein System die durchschnittliche Antwortzeit, äußert sich dies in Mängeln der allgemeinen Dienstequalität.

Feste Echtzeit: Die 3. Klasse der festen Zeitbedingungen erfordert ebenfalls eine durchschnittliche Antwortzeit, die aber innerhalb eines Zeitintervalls mit harten Zeitgrenzen zu erbringen ist.

Objektorientierte Softwareentwicklung mit Java wurde bereits 2000 durch die "Real-Time Specification for Java" (RTSJ) für den Einsatz in Echtzeitanwendungen erweitert [17]. Der in dieser Dissertation entwickelte Prototyp basiert auf RTSJ, sowie einiger im Rahmen des HIJA Forschungsprojektes [56] beschriebenen Einschränkungen dieser Spezifikation für sicherheits- und geschäftskritische Systeme. Bei der Beschreibung einer allgemeinen Rahmenarchitektur werden diese Einschränkungen als Grundlage der beschriebenen Methodik zur Nutzung eines echtzeitigen Nachrichtendienstes verwendet.

Kommunikation mit vorgegebenen Zeitgrenzen ermöglicht erst die Entwicklung verteilter sicherheits- und geschäftskritischer Systeme. Mit Einsatz einer objektorientierten Hochsprache kann man die Voraussetzungen für einen komponentenbasierten Systementwurf schaffen. Die Wiederverwendung getesteter Komponenten und der Einsatz redundanter Systemteile verbessert die Fehlertoleranz und Skalierbarkeit des Systems. Mit Einsatz eines Publiziere-/Abonniere-Nachrichtendienstes, der asynchrone Kommunikation in vorhersagbaren Zeitgrenzen bietet, unterstützt dieses Entwicklungsverfahren eingebettete Systeme und die dort existierenden Netze.

2.6 Netze

Ein Netz ist die Grundlage eines verteilten Systems. Es ermöglicht die Kommunikation der einzelnen Knoten und ist deshalb eine kritische Ressource, da der Verlust von Kommunikation den Verlust globaler Dienste des Systems bewirken kann [64]. Die Erfüllung von Echtzeitbedingungen

bei der Kommunikation erfordert den Einsatz von echtzeitfähigen Netzen. Neben einer physikalischen Verbindung besteht das Netz aus Netzzugangspunkten, über die einzelne Knoten mit einem geschichteten Protokoll zuverlässig, sicher, effizient und (bei Echtzeit-Netzen) mit Einhaltung von Zeitgrenzen kommunizieren können. Dieser Abschnitt stellt eine übliche Infrastruktur und Vernetzung bei eingebetteten Systemen kurz vor.

Ein Beispiel für eine Protokollschichtung ist das ISO/OSI Referenzmodell, das mit 7 Schichten von physikalischer bis Anwendungsschicht arbeitet, aber im Umfeld eingebetteter Systeme zu komplex ist und zu viel Aufwand durch Protokolldaten bewirkt. Deshalb wird für eingebettete Echtzeitsysteme oft ein reduzierter Protokollstapel mit nur drei Schichten (physikalisch, Datensicherung, Anwendung) eingesetzt. Die Anwendung greift direkt auf die Datensicherungsschicht zu und der Nachrichtenversand an alle Komponenten ("broadcast") erlaubt oft eine einfache Busverkabelung ohne Routing. Unterschieden werden kann bei diesen Protokollen zwischen **ereignisgesteuertem** und **zeitgesteuertem** Zugriff auf das Kommunikationsmedium (Netz).

Das Angebot an Netzen mit Echtzeitunterstützung ist relativ breit; es lässt sich jedoch durch den Anwendungsbereich etwas strukturieren. Im Folgenden werden Standardsysteme für Luftfahrt-, Automobil- und Automationstechnik mit Rundruflogik verglichen.

Avionik: ARINC 629

ARINC 629 [9] ist ein Datenbus-Standard (Netzprotokoll), der von Boeing für die Verkabelung und Steuerung in den Boeing 777 Flugzeugen entwickelt wurde. Die Spezifikation legt fest, wie Terminals und angeschlossene Systeme eines Bussystems Daten versenden und empfangen können. Dabei werden sowohl periodischer als auch nicht-periodischer Datentransfer durch ein Protokoll mit Zeitintervallen unterstützt.

Automobiltechnik: CAN, TT-CAN, TTP, FlexRay

Auf dem Automobilsektor konkurrieren neu entwickelte Protokolle, wie das zeitgesteuerte "Time-Triggered Protocol" (TTP [114, 33]) oder das gemischt ereignis- und zeitgesteuerte FlexRay Protokoll [16] mit etablierten Protokollen wie "Controller Area Network" (CAN [40]) oder "Time-Triggered CAN" (TT-CAN).

Das Controller Area Network (CAN) wurde von der Firma Bosch für die Kommunikation in Industrieanlagen und elektronischen Steuerungsanlagen entwickelt und findet heute verbreitet im Automobil Einsatz. Durch

sein nachrichtenbasiertes Kommunikationsmodell unterstützt CAN aber von sich aus nur weiche Echtzeitbedingungen. Erst durch Verbindung der Nachrichtenkommunikation mit Vergabe der Identifizierer der Nachrichten, die direkt auf die Priorität der Übertragung Auswirkungen haben, kann das Protokoll auch für harte Echtzeitbedingungen genutzt werden [61].

Durch ein zeitbasiertes Multiplexverfahren und entsprechende Ressourcenzuteilung bietet eine zeitgesteuerte Architektur ("Time-Triggered Architecture" [114]) zeitliche Synchronisation, die für die Bereitstellung von harten Echtzeitbedingungen genutzt wird.

Automation: ProfiBus, Ethernet/IP

In der Automation werden schon lange speziell entwickelte Feldbusse eingesetzt [111]. Neben einfacher Verkabelung gibt es in diesem Einsatzfeld aber auch einen Trend hin zur Standardisierung auf Basis von Internettechnologie wie TCP/IP.

Netze wie zum Beispiel das Avionics Full Duplex Switched Ethernet (AFDX/ARINC 664 [26, 27]) Protokoll für deterministisches Ethernet oder andere Punkt-zu-Punkt-Verbindungsnetze werden hier nicht weiter betrachtet. Anders als nur die Funktionalität einer Transportschicht[5], wie sie für die vorgestellte asynchrone Nachrichtenkommunikation notwendig ist, bieten diese Protokolle bereits im Protokollstapel Mechanismen, die synchrone Kommunikation über zum Beispiel entfernte Methodenaufrufe unterstützen. Asynchrone Nachrichtenkommunikation ist einfacher und benötigt diesen zusätzlichen Aufwand nicht. Die in den folgenden Kapiteln vorgestellte Rahmenarchitektur für den Entwurf verteilter Systeme mit Einsatz asynchroner Kommunikation mit Echtzeitbedingungen soll verteilte Anwendungen für sicherheitskritische Systeme mit geringeren Investitionen ermöglichen.

2.7 Vorteile asynchroner Publiziere-/Abonniere-Kommunikation im Echtzeiteinsatz

Obwohl eine asynchrone und zeitentkoppelte Kommunikation im ersten Moment den deterministischen und statisch verifizierbaren Anforderun-

[5] Open Systems Interconnections (ISO/OSI) Referenzmodell für Kommunikation

gen in sicherheitskritischen Echtzeitsystemen widerspricht, bietet eine Publiziere-/Abonniere-Kommunikation verschiedene Vorteile für solche verteilten Systeme. Die Anpassung an Echtzeitsysteme verspricht lohnenswerte Ziele im Gegensatz zu synchroner Kommunikation.

Anonyme Kommunikation: Sender benötigen keine Information über die Adressen und die Anzahl der Empfänger und genauso müssen die Empfänger nicht die Identität der Sender kennen.

Entkopplung von Publizist und Abonnent: Die Entkopplung in Adressierung, Zeit und Synchronisation verspricht größere Flexibilität und eine einfachere Wiederverwendung von Programmcode.

Viele-zu-Viele-Kommunikation: Die Ausnutzung von rundruforientierten Feldbussystemen im Umfeld von eingebetteten Systemen unterstützt Viele-zu-Viele-Kommunikation.

Skalierbarkeit: Einfache erweiterbare asynchrone Kommunikation ohne blockierende Anfrage/Antwort-Interaktionen skaliert gut von einfachen bis zu großen Systemen.

Programmierung von Steuerungsgeräten in der Automation: Das Publizieren von Informationen ist bei Steuergeräten ein häufig verwendetes Kommunikationsmodell. Es ist kein Wissen über interne Speicherabbildungen, Datenbankstrukturen oder Steuerungsgeräte notwendig. Die Nachrichten eines Kommunikationsknoten werden im Rundruf an alle interessierten Kommunikationspartner gesendet.

Effiziente Nutzung von Netzbandbreite: Kein Netzverkehr für Anfragen notwendig, Nutzung direkter ereignisgesteuerter Kommunikation. Rundruf-Kommunikation sieht keine Anfrage-Antwort-Interaktion vor. Nachrichten werden bei Verfügbarkeit verschickt und auf Empfängerseite gefiltert.

Portabilität über Plattformgrenzen: Die verwendete Kommunikationsinfrastruktur ist zwar als Implementierung auf Basis der Echtzeit-Java-Spezifikation (RTSJ) ausgelegt, das Kommunikationsprotokoll selbst ist aber von Systemarchitektur, Programmierplattform und Vernetzungstechnik unabhängig.

Ausrichtung auf Echtzeitbedingungen: Ereignisgesteuerter Nachrichtenversand eignet sich für den Einsatz mit Signalströmen und Statusaktualisierungen in Echtzeitsysteme.

Um diese Vorteile asynchroner Kommunikation für das Anwendungsgebiet deterministischer, echtzeitfähiger Systeme zu erschließen, müssen vorhersagbare Zeitgrenzen auf Sender- und Empfängerseite in der Kommunikation garantiert werden. Die in dieser Arbeit vorgestellte Rahmenarchitektur verspricht diese Garantien und ist Grundlage für eine durchgängige Methodik bei der Entwicklung verteilter Systeme mit Echtzeitbedingungen.

2.8 Verwandte Kommunikationsinfrastrukturen

Im Umfeld verteilter Systeme sind unterschiedliche Kommunikationsinfrastrukturen entwickelt worden. Die folgenden Unterabschnitte stellen eine Auswahl vor. Ihre Eignung bezüglich Echtzeitbedingungen und des Einsatzes in eingebetteten Systemen wird hierbei besonders bewertet. Kriterium bei der Auswahl ist außerdem die Nutzung mit einer objektorientierten Laufzeitumgebung, für die die mit dieser Arbeit entwickelte Methodik und Rahmenarchitektur ausgelegt ist. Die meisten der vorgestellten Arbeiten basieren deshalb auf Java Technologie, die neben einer objektorientierten Laufzeitumgebung auch über eine echtzeitfähige Systemplattform verfügt.

Die Entwicklung für sicherheits- und geschäftskritische Systeme ist hier noch am Anfang. Für eine bessere Beschreibung der Anforderungen an die vorgestellte Rahmenarchitektur werden deshalb auch etablierte Standards aus dem Automobilbereich wie OSEK/VDX und AUTOSAR aufgegriffen und verglichen.

2.8.1 Unterschiede zu synchroner Anfrage/Antwort Architektur

In Java wird bei synchroner Kommunikation mit entfernten Methodenaufrufen seit der Version 1.1 der Einsatz von RMI (“Remote Method Invocation”) [104] angeboten. RT-RMI (“Real-Time” RMI) für echtzeitfähige Systementwicklung ist derzeit Teil der Forschung [119, 110]. RMI konkurriert dabei direkt mit dem plattformübergreifenden Kommunikationsmodell von CORBA (“Common Object Request Broker Architecture”), auf dessen Basis die Kommunikation zwischen Anwendungen verschiedener Sprachen und Plattformen ermöglicht wird. Abschnitt 2.8.7 vergleicht deshalb die Echtzeitvariante von CORBA.

Asynchrone Kommunikation, wie sie diese Arbeit für die Entwicklung von verteilten Systemen verwendet, unterscheidet sich insbesondere durch die Anforderung an den Protokollstapel verwendeter Netze. Synchrone Punkt-zu-Punkt-Kommunikation erfordert für die Nutzung in Netzen mit Rundruf zusätzlichen Kommunikationsaufwand zur Emulation der Leitungsvermittlung. Für Nachrichten, die außerdem mehrere Empfänger besitzen, wird dieser Aufwand evtl. für mehrere einzelne Punkt-zu-Punkt-Verbindungen notwendig. Um dies zu vermeiden lohnt sich der direkte Zugriff auf Rundrufnetze wie ihn diese Arbeit beschreibt.

2.8.2 Verwandtschaft mit diensteorientierter Architektur und Webdiensten

Webdienste über HTTP und die diensteorientierte Architektur ("Service Oriented Architecture", SOA) propagieren seit geraumer Zeit ein zustandsloses Kommunikationsmodell für die Entwicklung von Unternehmenssoftware-Applikationen durch lose Kopplung. Im Gegensatz zu verteilten Objekten, die über entfernte Methodenaufrufe miteinander verbunden sind und kommunizieren, fordert diese Architektur eine Entkopplung und einfache Kommunikation über Nachrichten.

Die Trennung in Komponenten mit Geschäftslogik der Anwendung, Prozess-(Ablauf-)logik und Datentransformationscode soll Applikationen änderungsfreundlicher gestalten. Der Einsatz eines Geschäftsbusses unterstützt die lose Kopplung dieser Komponenten und die Orchestrierung der Dienste wird über eine zentrale Geschäftsprozesssteuerungskomponente geleistet. Der so mögliche Applikationsentwurf ist vergleichbar zur in dieser Arbeit vorgestellten Kommunikationsrahmenarchitektur und der möglichen Applikationsbeschreibung. Durch Zeitbedingungen für die Nachrichtenverarbeitung wird dieser Ansatz erweitert.

2.8.3 Anleihen von Java Messaging Service (JMS)

Echte asynchrone Kommunikation im Java Umfeld wurde 2002 mit der Definition von JMS ("Java Messaging Service") [105] eingeführt. Dieser skalierbare, asynchrone Nachrichtendienst bietet sowohl eine themenbasierte Publiziere-/Abonniere-Kommunikation als auch die inhaltsbezogene Filterung der übertragenen Nachrichten. Die Architektur selbst ist zentralisiert.

Durch die Beschreibung von Nachrichtenkanälen wird in dieser Arbeit die Applikationsarchitektur dezentralisiert und die Überprüfung von Echtzeitbedingungen in die Laufzeitumgebungen beteiligter Kommunikationspartner verschoben.

2.8.4 Gegensatz zu Jini, JavaSpaces, OSGi und JXTA

Für die Implementierung von dezentralen Ad-hoc-Netzen zwischen unterschiedlichsten Internet-fähigen Endgeräten hat Sun Microsystems ein auf RMI basierendes Protokoll für Java-Applikationen entwickelt: Jini [107]. Durch Einsatz von Zwischenspeichern (JavaSpaces [106]) analog zu einer schwarzen Tafel können asynchrone Kommunikationsstrukturen implementiert werden. Anwendungen, die eine Gleiche-zu-Gleichen-Kommunikationsform ("Peer-to-Peer", P2P) verwenden, lassen sich so auf Basis von Eins-zu-eins-Kommunikation mit entfernten Methodenaufrufen über RMI entwickeln.

Ausgehend von dienstorientierter Applikationsentwicklung hat ein breites Konsortium von Software- und Hardwareanbietern auf Basis von Java die Programmschnittstellenbibliothek "Open Software Gateway Initiative" (OSGi) definiert. Dieser ebenfalls auf Ad-hoc-Netze ausgerichtete Ansatz zur Diensteinstallation basiert zwar auf einem Vernetzungsansatz für die Verteilung und das Auffinden von Diensten, arbeitet aber ansonsten in einem lokalen Service-Container, der für den lokalen Zugriff auf Software- und Hardwaredienste gedacht ist.

Eine leichtgewichtigere und nicht nur auf Java ausgerichtete Variante[6] für die Entwicklung von P2P Anwendungen ist seit 2001 die JXTA Plattform [22]. Basierend auf sechs zugrundeliegenden Protokollen (z.B. Ankündigung oder Suche) ermöglicht dieses System die Entwicklung von verteilten Gleiche-zu-Gleichen-Anwendungen. Die Kommunikation selbst ist immer synchron und asynchrones Verhalten wird allein durch lokale Pufferung von Anfragen und Anforderungen ermöglicht.

Ähnlich wie im Abschnitt 2.8.1 mit RMI beschrieben, ist für den Einsatz dieser Bibliotheken mit Netzen, die Viele-zu-Viele-Kommunikation unterstützen, zusätzlicher Kommunikationsaufwand für die Leitungsvermittlung notwendig. Bei der Evaluierung der entworfenen Rahmenarchitektur für weiche Echtzeitbedingungen (vgl. Abschnitt 5.2) wird dieser Aufwand am Beispiel einer P2P-Bibliothek und Anwendung untersucht. Der mit der

[6] Es gibt auch Implementierungen auf Basis von C oder Perl.

EventChannelNetwork API verfolgte Ansatz spart diesen Aufwand und erlaubt die Beschreibung verteilter Systeme mit Echtzeitbedingungen.

2.8.5 Ähnlichkeit mit Java InfoBus

Zur losen Kopplung von Java Komponenten (JavaBeans) innerhalb einer virtuellen Maschine hat Sun Microsystems ein eng mit diesem Komponentenkonzept verwobenes Protokoll zur Interaktion definiert. Der Java InfoBus [108] definiert einen themenbezogenen Informationsbus, mit dem die direkte Kommunikation über Methodenaufrufe ergänzt werden kann. Dieser Ansatz wird derzeit nicht in Hinblick auf Verteilung über mehrere virtuelle Maschinen und Adressbereiche ausgedehnt und es sind keinerlei Aussagen über Einhaltung von Zeitbedingungen bei der Interaktion vorgesehen.

Eine auf Basis von Java Komponenten eingeführte Entwicklungsmethode über grafische Werkzeuge, wie zum Beispiel das Bean Development Kit (BDK), unterstützt dabei auch den Informationsbus zur Kommunikation zwischen Komponenten einer Applikation [44]. Der Entwickler kann sich auf die Funktionalität der Komponenten konzentrieren und beschreibt grafisch deren Verbindung. Das Laufzeitsystem regelt die Kommunikation.

2.8.6 JavaGroups als Vergleich

JavaGroups [13, 14] definieren ebenfalls ein gruppenbasiertes und anonymes Kommunikationsprotokoll. Dieses Open-Source-Projekt wird zum Beispiel bei der netzweiten Kopplung von Enterprise-Java-Containern des JBoss-Projektes [5] zur Bildung von Rechnerbündeln (“cluster”) eingesetzt.

Die in JGroups [3] umbenannte Programmbibliothek beschreibt in erster Linie einen frei konfigurierbaren Protokollstapel, über den sich das zugrundeliegende Verbindungsprotokoll (z.B. UDP, TCP) verbergen lässt. Dadurch lassen sich beliebige Frage-/Antwortmuster, mit und ohne Persistenz etc. abbilden. Bei dieser Kommunikationsinfrastruktur steht besonders die Flexibilität und nicht die Performanz der Verarbeitung im Vordergrund.

2.8.7 Vergleich mit RT-CORBA

Die Object Management Group (OMG) hat ein Objekte/Dienste-Informationsmodell für heterogene Applikationen in verschiedenen Sprachen und auf unterschiedlichen Plattformen entwickelt. Um die "Common Object Request Broker Architecture" (CORBA) für Echtzeitbedingungen anzupassen, ist deshalb die Echtzeiterweiterung (RT-CORBA [78, 80]) definiert worden. RT-CORBA beschreibt auf Basis der synchronen CORBA-Kommunikationsplattform auch eine asynchrone Nachrichteninteraktion. Neben dem relativ hohen Protokollaufwand bei dieser Kommunikation ist besonders die TCP/IP-Orientierung der CORBA-Plattform ein Hindernis für die Nutzung im Umfeld sicherheits- oder geschäftskritischer Systeme, die Netze mit Viele-zu-Viele-Kommunikation verwenden. Ein weiterer Ansatz der OMG, um beschränkte Ressourcen und typische Probleme eingebetteter Systeme zu behandeln, stellt die MinimumCORBA Spezifikation [79] dar. Beide Spezifikationen sind orthogonale Weiterentwicklungen der CORBA Rahmenarchitektur, aber die Integration der unterschiedlichen Anforderungen für eine CORBA-Implementierung auf eingebetteten Systemen mit beschränkten Ressourcen und Echtzeitbedingungen bleibt unklar.

Mit der Einführung von Echtzeitbedingungen im CORBA Softwareentwurf (RT-CORBA) hat die OMG (Object Management Group) zwei asynchrone Kommunikationsprotokolle auf Basis der etablierten synchronen Infrastruktur definiert. Mit synchroner Kommunikation kann unter RT-CORBA beziehungsweise in unterschiedlichen ORB-Implementierungen (z.B. TAO [93]) auf einen Nachrichten- und Benachrichtungsdienst [76, 77] zugegriffen werden. Dieser zentralisierte Ansatz und die TCP/IP-Orientierung von CORBA müssen beim Einsatz mit eingebetteten Systemen und dort üblichen Rundrufnetzen aber kritisch betrachtet werden.

2.8.8 Unterschiede gegenüber OSA+

Konzepte, die von Hardware-Seite die Erfordernisse von verteilten Echtzeitsystemen angehen, befinden sich ebenfalls in der aktuellen Entwicklung. Mit der "Open System Architecture Platform for Universal Services" (OSA+) [20] wird auf einer Dienste-orientierten Mikrokern-Architektur eine Auftragsbearbeitung eingeführt. Die zu komplexe Infrastruktur einer CORBA Middleware wird durch eine reduzierte Mikrokern-API ersetzt. Auf diesem Kern mit nur sechs Funktionen müsste eine objektorientier-

te Schnittstelle zur Nutzung in hochsprachlich entwickelten, sicherheits- oder geschäftskritschen Systemen aber erst noch implementiert werden.

2.8.9 Ähnlichkeit zu Kommunikation nach AUTOSAR oder OSEK/VDX

Die steigende Komplexität von Steuerungsgeräten und -netzen im Automobilbau hat zu entsprechenden Standardisierungsbemühungen und einer herstellerübergreifenden Entwicklung von Spezifikationen geführt. Zwei dieser Entwicklungen sind OSEK/VDX ("Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen" und "Vehicle Distributed eXecutive" [2]) und AUTOSAR ("AUTomotive Open System ARchitectures" [1]). Die OSEK-Initiative wurde bereits 1993[7] von deutschen Automobilherstellern und -softwarezulieferern gegründet. Ergebnis war die Spezifikation von Schnittstellen eines Echtzeitbetriebssystems für eingebettete Komponenten, das im Automobilbereich eingesetzt wird. Mit den Spezifikationen OSEK-COM [83] und OSEK-NM [84] legte das Konsortium Schnittstellen für die Kommunikation zwischen Programmteilen und das Management von Steuergeräten innerhalb eines Autos fest. Die spezifizierte Kommunikationsschicht definiert das interne und externe Verhalten von Komponenten bei der Kommunikation zwischen Anwendungen und Prozessen. Diese Kommunikation basiert auf Nachrichten, deren Empfang über Warteschlangen fester Länge realisiert wird und einen nichtblockierenden Nachrichtenaustausch auf Applikationsschicht unterstützt. Die Kommunikation in OSEK/VDX erfordert die Funktionalität einer Transportschicht und die Softwarekomponenten der Kommunikationspartner bauen auf der Betriebssystemschnittstelle von OSEK/VDX auf.

Auch das im Jahr 2003 gestartete AUTOSAR-Projekt, das die Arbeiten der OSEK-Gremien fortsetzt [50], fordert noch keine objektorientierte Laufzeitumgebung für die Entwicklung verteilter Systeme im Automobil. Mit modellbasierten Konzepten versucht man eine sogenannte Basissoftware zu entwickeln, die technische Details der Kommunikation kapselt. Mit Integration von Komponenten über einen Container und nachrichtenorientierter Kommunikation wird versucht, der steigenden Heterogenität im Umfeld von Automotiveapplikationen [75] zu begegnen.

[7] 1994 Zusammenschluss mit VDX

2.8.10 Zusammenfassung

Die Tabelle 2.3 auf der folgenden Seite zeigt einen Überblick der in diesem Abschnitt vorgestellten Kommunikationsinfrastrukturen. Zum Vergleich werden Entwurfskriterien der Kommunikationsrahmenarchitektur dieser Arbeit aufgelistet.

Die Eignung asynchroner Nachrichtenkommunikation für objektorientierte verteilte Systeme mit Echtzeitbedingungen wird derzeit noch wenig ausgenutzt. Der Einsatz objektorientierter Laufzeitumgebungen in verteilten Systemen ist meist mit dem Konzept für transparente entfernte Methodenaufrufe verknüpft. Für die Garantie von Echtzeitbedingungen wird dabei die Idee verteilter Objekte und Methodenaufrufe über Verbindungen mit fest vorgegebenen Kapazitäten genutzt; Beispiele sind Java RMI und (RT-)CORBA.

Der Einsatz flexibler asynchroner Kommunikationsinfrastrukturen für objektorientierte Systeme ist im Gegensatz dazu oft ohne Zusicherungen von Zeitgrenzen oder Beschreibungen für die Dienstgüte versehen. Beispiele sind neben dem Java Nachrichtendienst JMS insbesondere Ergänzungen für asynchrone Kommunikationsformen mit Jini, dem asynchronen Einsatz von Webdiensten oder Echtzeiterweiterungen für CORBA. Diese Infrastrukturen basieren oft auf Standard-IP-Netzen und sind nicht auf beschränkte Ressourcen und Rundruflogik eingebetteter Systeme ausgerichtet.

Java als objektorientierte Laufzeitumgebung unterstützt zwar Echtzeitbedingungen, verfügbare Kommunikationsarchitekturen machen davon jedoch selten Gebrauch. Echtzeit mit Objektorientierung ist zwar nicht unmöglich, aber Entwicklungen aus dem Automobilbereich zeigen, dass diese Verfahren derzeit selten gefordert werden. Auch Anforderung nach einer Beschreibung von Systemdiensten und -komponenten werden von Spezifikationen wie AUTOSAR zwar beschrieben, aber der Systementwurf mittels einer Beschreibung (z.B. durch Webdienste oder mit OSGi-Modulen) wird nicht auf die Unterstützung bei der Ablaufanalyse notwendiger Kontrollfäden erweitert.

Kriterien zur Bewertung der unterschiedlichen Kommunikations-infrastrukturen	*Java RMI / CORBA*	Webdienste / SOA	JMS	Jini, JavaSpaces	OSGi	JXTA	Java InfoBus	JGroups	RT-CORBA	OSA+	OSEK/VDX	AUTOSAR	*EventChannelNetwork*
Asynchrone Kommunikationsformen		(✓)	✓	✓		✓		✓	(✓)		✓	✓	✓
Einsatz direkt mit Rundrufnetzen						(✓)		(✓)			✓	✓	✓
Objektorientierte Laufzeitumgebung	✓	✓	✓	✓	✓	(✓)	✓	✓	(✓)			(✓)	✓
Wissen über Echtzeitbedingungen	(✓)								✓	✓		✓	✓
Deskriptive Systementwicklung		✓		(✓)	(✓)	(✓)					✓	✓	✓
Unterstützung einer Ablaufplananalyse													✓

Tabelle 2.3: Infrastrukturen im Überblick

2.9 Ausführungsanalyse und Zeitabschätzung im ungünstigsten Fall

Für die Entwicklung (verteilter) Echtzeitsysteme sind verschiedene Analysewerkzeuge verfügbar [74]. Um die Verarbeitungsfristen nebenläufiger Kontrollfäden zu prüfen, erleichtern sowohl Werkzeuge zur Analyse der Ablaufkoordinierung ("scheduling"), als auch Unterstützung bei der möglichst genauen Abschätzung der Ausführungszeit von Programmcode im ungünstigsten Fall (WCET) die Entwicklung. Für die Analyse von Ablaufplänen ist die WCET notwendig. Analysewerkzeuge ermöglichen die frühzeitige Vermeidung von Fehlern aus der Nichteinhaltung von Zeitfristen und es können sicherheitskritische Systeme mit hohen Sicherheitsanforderungen implementiert werden.

Die im Umfeld von sicherheitskritischen Systemen oft notwendige Zertifizierung und ausführliche Prüfung von Applikationen kann durch den Einsatz von Analysewerkzeugen vereinfacht werden. Als Standard in der Softwareentwicklung für den Flugverkehr existiert zum Beispiel DO-178B "Software Considerations in Airbone Systems and Equipment Certification" der RTCA ("Radio Technical Commission for Aeronautics") [90]. Ziel zukünftiger Softwareentwicklung muss es sein, die Ergebnisse einer statischen Analyse von Ablaufplänen in den Anforderungskatalog von Zertifizierungsverfahren zu integrieren. Eine Methodik, die durchgängig von Entwurf bis Analyse den Entwickler unterstützt ist hierfür notwendig.

Werkzeuge und Simulationen für die Analyse der Ausführung von Kontrollfäden arbeiten auf einer Modellebene und erfordern eine Systembeschreibung mit Zeitfristen und Informationen zur Ablaufkoordinierung des auszuführenden Programms als Eingabe. Für die gute Abschätzung von Programmcodeausführungszeiten, die als Grundlage für die Ausführungsanalyse benötigt wird, ist außerdem konkretes Wissen über die Ausführungsplattform notwendig. Informationen der verwendeten Hardware über Cache-Aufbau, Pipelining-Verhalten und zum Beispiel Sprungvorhersagen ermöglichen diese Zeitabschätzung. Der Leitfaden für die Methodik ("methodology handbook") [57] des HIJA-Forschungsprojektes gibt eine detaillierte Aufstellung entsprechender Werkzeuge und Techniken.

2.10 Zusammenfassung

Die Verwendung einer deskriptiven Programmiermethode zur Generierung von Programmcode und automatischer Erzeugung von Analysemodellen für die Verteilung und asynchrone Kommunikation soll die Entwicklung vereinfachen. Durch Nutzung einer objektorientierten Rahmenarchitektur können Muster für die Softwareentwicklung eines verteilten Systems vorgegeben werden.

Um der Komplexität bei der Entwicklung nicht durch eine komplexe und noch schwierigere Beschreibungsmethode zu begegnen, muss eine einfache Systembeschreibung mit Konzepten aus der Anwendungsdomäne verteilter Systeme gefunden werden, die Entwurf, Programmcodegenerierung sowie die Analyse als Teil eines durchgängigen Softwareentwicklungsprozesses unterstützt.

Um diese durchgängige Methodik für die Entwicklung verteilter Systeme mit Echtzeitbedingungen zu beschreiben, baut diese Dissertation auf bekannten Verfahren des objektorientierten Softwareentwurfs auf. Eine Beschreibungsmethode für die Kommunikation in verteilten Echtzeitsystemen ergänzt zusammen mit einer unterstützenden Rahmenarchitektur diese Ansätze bei der Entwicklung. Die Entwicklung eines asynchronen Nachrichtendienstes, der Echtzeitbedingungen unterstützt, ist besonders für den Einsatz mit verteilten eingebetteten Systemen lohnenswert.

Verwandte Arbeiten bei der Kommunikation in verteilten Systemen werden dargestellt und die Eignung asynchroner Nachrichtenkommunikation für eingebettete Systeme und Echtzeitsysteme mit der Nutzung einer objektorientierten Laufzeitumgebung wird verdeutlicht. Die Eigenschaften, der im Umfeld eingebetteter sicherheits- und geschäftskritischer Echtzeitsysteme üblichen Netze und Feldbussysteme mit Rundruflogik, sollen dabei ausgenutzt werden.

Kapitel 3

Entwurf einer Methodik mit notwendiger Rahmenarchitektur

Die in dieser Dissertation vorgestellte durchgängige Entwicklungsmethodik für verteilte Systeme ist das Ergebnis der im letzten Kapitel beschriebenen Möglichkeiten für asynchrone Nachrichtenkommunikation in Echtzeitbedingungen, objektorientierter Rahmenarchitekturen und Analysen für verteilte Systeme auf Modellebene.

Allgemeine Anforderungen an die Methodik und die verbundene Rahmenarchitektur werden aus den im Abschnitt 1.1 auf Seite 2 aufgestellten Thesen abgeleitet. Die durchgängige Unterstützung der Entwicklung durch diese Methodik wird als abstrakter Lösungsansatz entwickelt. Aus dem aktuellen Stand der Technik und dieser Lösungsidee werden konkrete Entwurfsentscheidungen gefolgert. Mit der Festlegung auf eine objektorientierte Laufzeitumgebung werden weitere Anforderungen an die notwendige Kommunikationsinfrastruktur und verwendete Kommunikationsnetze beschrieben.

Ein direkter asynchroner Nachrichtendienst für die Publiziere-/Abonniere-Kommunikation wird entworfen und erläutert, wie Echtzeitbedingungen erfüllt werden können. Der Entwurf in Abschnitt 3.5 verwendet dazu bekannte Software-Entwurfsmuster und beschreibt eine plattformunabhängige Rahmenarchitektur. Überlegungen zur Implementierung mit Echtzeit-Java finden sich im nachfolgenden Kapitel 4.

Zur einfachen Verwendung der Rahmenarchitektur wird in Abschnitt 3.6 eine ergänzende Systembeschreibung für verteilte Systeme mit Nachrichtenkanälen entworfen. Diese Beschreibung ist Grundlage für die Konfiguration der einzelnen Kommunikationsknoten und die Generierung von

Standard-Programmcode. Sie wird auch für die statische Analyse der Ablaufkoordinierung notwendiger Kontrollfäden in den Kommunikationsknoten genutzt.

3.1 Allgemeine Anforderungsanalyse

Ergebnis des im letzten Kapitel vorgestellten Stands der Technik ist die Abgrenzung einer Methodik für die Entwicklung verteilter Systeme mit Nachrichtenkommunikation in Echtzeitbedingungen in verwandten Forschungsbereichen. Dieses Kapitel nutzt diese Abgrenzung, um aus allgemeinen Anforderungen einen Entwurf für die Methodik, die verbundene Rahmenarchitektur und geeignete Techniken bei der Umsetzung zu ermitteln.

Forschungsbereiche, die zur Bestimmung erster Anforderungen an die Methodik herangezogen werden sind dabei

- eine objektorientierte Softwareentwicklung,
- die Systembeschreibung zur deskriptiven Entwicklung von Softwaresystemen,
- geeignete Kommunikationsmodelle für eine lose Kopplung von Systemkomponenten,
- echtzeitfähige Netzinfrastrukturen mit garantierten Übertragungszeitfristen,
- mögliche Ausführungsumgebungen in eingebetteten Systemen und
- Mechanismen, die Echtzeitbedingungen in nebenläufigen Systemen sicherstellen können.

Ausgehend hiervon ergibt sich eine erste Liste von Anforderungen.

1. Vereinfachung von Entwurf, Implementierung und Analyse der verteilten Applikation durch Nutzen einer Systembeschreibung
2. Einsatz asynchrone Nachrichtenkommunikation über Rundrufnetze mit Echtzeitbedingungen bei der Verarbeitung

Mit der Definition von Nachrichtenkanälen kann der Entwickler bei der Beschreibung von Komponenteninteraktionen von der technischen Implementierung abstrahieren. Kanäle beschreiben die Kommunikation zwischen einzelnen Knoten und bieten die Grundlage für die Erzeugung von Standard-Programmcode und einer äquivalenten Modellbeschreibung dieser Kommunikation.

Durch Ausnutzung von Rundrufnetzen und Bereitstellung einer Nachrichtenkommunikation mit Publiziere-/Abonniere-Logik ermöglicht die Methodik die Entwicklung verteilter Systeme. Bei der Notwendigkeit harter Echtzeitbedingungen wird das dynamische Verhalten der Publiziere-/Abonniere-Kommunikation durch eine statische Zuordnung von Publizisten und Abonnenten ersetzt. Die asynchrone Nachrichtenkommunikation wird durch Netze mit Viele-zu-Viele-Kommunikation unterstützt und vereinfacht die Entwicklung verteilter Systeme.

Übertragung in Echtzeit wird von den verwendeten Kommunikationsnetzen erwartet. Die Entwicklungsmethodik bietet dies nicht. Sie ermöglicht dem Entwickler nur die Beschreibung von Echtzeitbedingungen bei der Verarbeitung von Nachrichten. Der Empfang und die Verarbeitung von Nachrichten müssen dies lokal sicherstellen. Dazu bietet die Methodik Unterstützung.

3. Einsatz objektorientierter Entwicklungstechniken bei Analyse und Design

Eine für die Methodik notwendige Rahmenarchitektur, die es erlaubt die Funktionalität asynchroner Kommunikation von der Implementierung der weiteren Programmlogik zu trennen, kann durch objektorientierte Muster und Rahmen unterstützt werden.

4. Unterstützung für Wiederverwendung, Spezialisierung, Varianten

5. Lose Kopplung der Komponenten des verteilten Systems

Die Trennung zwischen Kommunikation und weiterer Programm- und Verarbeitungslogik vereinfacht damit die Entwicklung der verteilten Applikation. Lose gekoppelter Komponenten vereinfachen ihre Wiederverwendung und die Entwicklung von Spezialisierungen und Varianten, die zur Unterstützung einer breiten Systemplattform hilfreich sind. Die Entwicklungsmethodik muss dabei die Ergänzung neuer und redundanter Komponenten unterstützen, um die Skalierbarkeit des verteilten Systems sicher zu stellen.

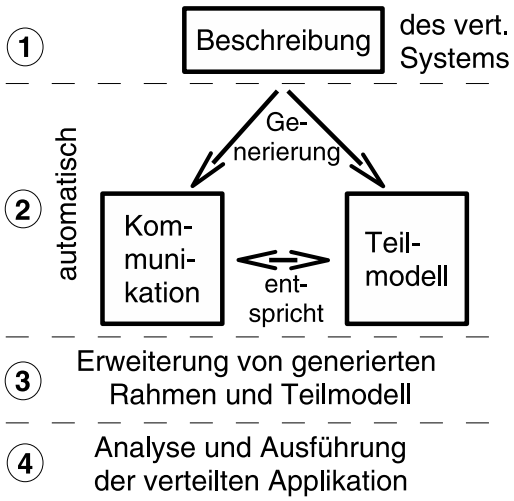

Abbildung 3.1: Lösungsansatz der Entwicklungsmethodik

3.2 Lösungsansatz der notwendigen Methodik

Eine Methodik für die durchgängige Unterstützung des Softwareentwicklungsprozesses, wie sie von dieser Dissertation beschrieben wird, lässt sich in vier Schritte unterteilen. Abbildung 3.1 illustriert diesen Lösungsansatz grafisch. Die Aufgaben des Entwicklers sind dann in jedem Schritt wie folgt:

1. Ausgangspunkt für die Entwicklung eines verteilten Systems ist die Beschreibung des Systems mit Knoten und Nachrichtenkanälen. Der Entwickler muss im 1. Schritt für jeden Knoten alle Verbindungen zu notwendigen Nachrichtenkanälen festlegen und die Rolle des Knotens als "Sender" oder "Empfänger" für diese Kanäle definieren.
   Durch Angabe von Kommunikationseigenschaften, wie zum Beispiel Periode und Verarbeitungsfristen, werden hier die weiteren Schritte unterstützt.

2. Der 2. Schritt ist automatisiert und für das beschriebene System wird Standard-Programmcode für die Kommunikation und eine äquivalente Modellbeschreibung generiert. Die generierte Infrastruktur erfüllt dazu die Anforderungen der Beschreibung von Schritt 1.

3. Im Folgeschritt 3 kann der Entwickler ausgehend von ebenfalls generierten Programmrahmen und konfigurierten Programmteilen die

Geschäftslogik der Applikation vervollständigen. Außer Programmcode muss hier auch das Modell für das Analysewerkzeug um eine Beschreibung des neuen Programmcodes erweitert werden.

4. In Schritt 4 muss der Entwickler generierten und selbstgeschriebenen Programmcode auf seine Ausführungszeit im ungünstigsten Fall hin abschätzen[1]. Diese Zeiten ergänzen die Modellbeschreibung und für Systeme mit harten Echtzeitbedingungen kann durch Einsatz eines Werkzeugs zur Analyse der Ablaufkoordinierung in jedem Knoten die Einhaltung gegebener Echtzeitfristen vor der Ausführung untersucht werden.

3.3 Entwurfsentscheidung und resultierende Anforderungen

Der vom letzten Abschnitt vorgegebene Lösungsansatz für die Entwicklungsmethodik verteilter Systeme mit Echtzeitbedingungen ist plattform- und sprachunabhängig. Um das Verfahren für den Einsatz mit objektorientierten Laufzeitumgebungen zu ermöglichen, untersucht dieser Abschnitt die Möglichkeiten und Einschränkungen durch die Verwendung einer virtuellen Maschine bei der Ausführung einer verteilten komponentenbasierten Applikation.

Die Methodik orientiert sich an einer Zielplattform sicherheits- und geschäftskritischer eingebetteter Systeme. Objektorientierte Laufzeitumgebungen unterstützen die Entwicklung entsprechender Applikationen, die virtuelle Maschine übernimmt zum Beispiel die Anforderung und Freigabe von Hauptspeicher (automatische Speicherbereinigung, “Garbage Collection”, GC) und anderen Ressourcen. GC verringert die Gefahr von Speicherlecken in der Applikation. Sie hat aber auch Auswirkungen auf die Ausführungszeit des Programms, was für Echtzeitsysteme Schwierigkeiten mit sich bringt. GC kann unter Umständen die Ausführungsdauer von Programmläufen verringern, da sie eine Zusammenfassung der Freigabeoperationen bewirkt. Der im Allgemeinen erhöhte Verwaltungsaufwand zeigt aber oft die gegenteilige Wirkung. Außerdem ist der genaue Zeitpunkt ihrer Durchführung kaum vorherzusehen und deshalb für sicherheitskritische Echtzeitsysteme nicht geeignet.

[1] Hier empfiehlt sich die Unterstützung des Entwicklers mit einem WCET-Analysewerkzeugs.

Neben GC beschreibt [45] noch weitere nützliche Aufgaben einer Laufzeitumgebung für den Einsatz mit einer Verteilungsplattform ("middleware") in verteilten Systemen. Für die beabsichtigte asynchrone Kommunikation zwischen Komponenten in einem verteilten System werden daraus folgende Anforderungen ermittelt.

Ressourcenverwaltung bezeichnet nicht nur die Verwaltung der Ressource Hauptspeicher, sondern ist auch für Betriebsmittel wie Prozesse, Kontrollfäden oder Verbindungen hilfreich. Ziel der Ressourcenverwaltung ist die Verbesserung von Performanz, Skalierbarkeit und Verfügbarkeit einer Anwendung in der Laufzeitumgebung.
Da handelsübliche ("Commercial Off-The-Shelf", COTS) Hardware und Betriebssysteme nicht immer den Anforderungen von sicherheitskritischen Echtzeitsystemen genügen (z.B. keine Unterstützung für asynchrone Kommunikationsverfahren, vgl. Abschnitt 2.3 auf Seite 18ff.), muss eine Rahmenarchitektur für verteilte Systeme die asynchrone Kommunikationsarbeit mittels zusätzlicher Hilfsprozesse im Hintergrund erbringen. Die zusätzlichen Prozesse, Kontrollfäden und Speicherkapazität für Zwischenspeicher müssen von der Laufzeitumgebung bereitgestellt werden.

Nebenläufigkeit erlaubt parallele Arbeit in einer oder mehreren Applikationen. Die Basisfunktionalität dafür wird durch das Betriebssystem ermöglicht. Prozesse und Kontrollfäden sind Objekte des Betriebssystems. Die Laufzeitumgebung nutzt diese Basisfunktionalität, um eine eigenständige Prozess- und Kontrollfadenverwaltung mit Unterstützung der Nebenläufigkeit aufzubauen.
Die vorgestellte Rahmenarchitektur erfordert Nebenläufigkeit bei Erbringung der asynchronen Kommunikation. Um die Arbeit für Nachrichtenempfang und Nachrichtenverarbeitung parallel und miteinander verzahnt auf den Prozessoren der verteilten Anwendung ausführen zu können, muss die Laufzeitumgebung diese verwalten. Abschnitt 3.4.1 beschreibt die dafür notwendigen Verfahren genauer.

Verbindungsverwaltung ist notwendig für verteilte Anwendungen. Verbindungen sind Endpunkte von Kommunikationskanälen innerhalb einer Anwendung. Sie existieren an den Grenzen einer Schicht oder Komponente und sind im aktiven Zustand immer mit einem Prozess oder Kontrollfaden assoziiert. Verbindungen benötigen, wie andere Ressourcen, Speicher und Prozessorzeit. Bei steigender Anzahl sind sie ein limitierender Faktor. Eine feste Zuordnung von Verbindungen

zu Prozessen oder Kontrollfäden ist somit nicht sinnvoll und kann schnell zu Engpässen in der Performanz einer Anwendung führen. Eine Verwaltung mit Reservoirspeicherung ("Pooling") vergibt freie Verbindungen auf Anfrage an Kontrollfäden und nach der Verwendung wird die Verbindung wieder in den Speicher zurückgestellt.
Da die Verwaltung der asynchronen Kommunikation einen Teil der Rahmenarchitektur darstellt, sind die Anforderungen an die verwendete Laufzeitumgebung hinsichtlich des Zugriffs auf einen Netzzugangspunkt zum Netz oder Feldbussystem mit Rundruflogik auf die Bereitstellung der Funktionalität einer ISO/OSI Transportschicht reduziert. Mittels eines relativ einfachen Kommunikationssockels (vgl. Abschnitt 2.3) soll der Aufwand für die Anpassung an neue Netze in der Rahmenarchitektur gering gehalten werden.

Verfügbarkeit ist eine Anforderung, die an die Anwendung selbst gestellt wird, jedoch vor allem von ihrer Umgebung umzusetzen ist. Laufzeitumgebung und Hardware sind hier gefragt. Die Laufzeitumgebung muss eine stabile und deterministische Ausführung der Programmlogik garantieren. Beim Zugriff auf Netzressourcen müssen Fehler an die Anwendungsschicht weitergegeben werden.
Die Rahmenarchitektur unterstützt Publiziere-/Abonniere-Kommunikation in verteilten Systemen. Diese Form asynchroner Kommunikation ermöglicht die Entkopplung von Sendern und Empfängern und vereinfacht so den Entwurf von robusten verteilten Anwendungen. Mit einem Fehlerbenachrichtungsmodell über Ausnahmen informiert die Rahmenarchitektur die Anwendung über mögliche Fehlerfälle und Probleme bei der Verfügbarkeit von Netzdiensten. Die Anwendung kann hierauf entsprechend reagieren. Kommunikationsfehler, die das verwendete Netz nicht an die Anwendungsschicht weiterreicht, müssen durch entsprechende Sicherungsmaßnahmen im Anwendungsprotokoll behandelt werden.

Sicherheit ist für verteilte Systeme essentiell. Die Verteilung über ein Netz und mehrere Kommunikationsknoten bietet mehr Angriffspunkte für Angreifer. Im einfachsten Fall unterstützt die Laufzeitumgebung Mechanismen zur Zugriffskontrolle (Authentifizierung) und zur Vergabe von Zugriffsrechten (Autorisierung).
Die Rahmenarchitektur sieht in der aktuellen Version keine dieser Sicherheitsmechanismen vor. Schutz der eigentlichen Kommunikation ist ebenfalls nicht in der Rahmenarchitektur implementiert. Vertraulichkeit (Schutz vor Abhören) und Integrität (Schutz vor Manipula-

tion) bei der Kommunikation kann die Anwendung aber durch Verwendung von Verfahren der asymmetrischen Verschlüsselung implementieren.

Mit Einsatz einer objektorientierten Laufzeitumgebung für die Entwicklung sicherheits- und geschäftskritischer verteilter Systeme unterscheidet sich die vorgestellte Rahmenarchitektur noch stark von existierenden Lösungen beim Entwurf verteilter Systeme mit Echtzeitbedingungen. Derzeit muss der Entwickler objektorientierter verteilter Systeme noch über ein umfassendes Verständnis von komplexen Techniken, wie zum Beispiel Ablaufkoordinierung und Speichermanagement, verfügen.

Eine Rahmenarchitektur verschiebt dieses Wissen in die Implementierung der Laufzeitumgebung und der Entwickler kann sich mit Hilfe der vorgestellten Entwicklungsmethodik auf die eigentliche Funktionalität seiner Anwendung konzentrieren. Der folgende Abschnitt 3.4 untersucht die Notwendigkeit der Ablaufkoordinierung und die Anforderungen an verwendete Kommunikationsnetze genauer und beschreibt diese im Zusammenhang mit dem bereits vorgestellten Stand der Technik.

Eine auch ohne Einsatz und Erweiterung einer Laufzeitumgebung denkbare Entwicklungsmethodik erfordert die hier vorgestellten Anforderungen ähnlich. In abgewandelter Version sind die vorgestellten Verfahren auch für den Einsatz mit einer anderen Softwarearchitektur und den damit verbundenen Möglichkeiten für Softwareentwurf, Programmcodeentwicklung und Korrektheitsanalysen hilfreich. Die vorgestellte objektorientierte Laufzeitumgebung und der Einsatz eines komponentenorientierten Entwurfs mit objektorientierten und deskriptiven Sprachmitteln erfüllt die beschriebenen Anforderungen aber als einheitliche Umgebung für die durchgängige Entwicklung.

3.4 Anforderungen an Laufzeitumgebung und Infrastruktur

Der Einsatz einer objektorientierten Hochsprache mit Laufzeitumgebung erfordert bei der Entwicklung von sicherheits- und geschäftskritischen verteilten Systemen neben Prozessoren zur Ausführung der Applikationslogik, dem direkten Zugriff auf Sensoren und Aktoren und der Verbindung über ein echtzeitfähiges Netz (vgl. Abschnitt 2.6 auf Seite 28ff.) besonders

die Unterstützung von Echtzeitbedingungen durch spezielle Echtzeitkontrollfäden. Ein geeignetes Verfahren für die Ablaufkoordinierung ("scheduling") dieser Kontrollfäden mit Einhaltung von Echtzeitbedingungen wird in Abschnitt 3.4.1 diskutiert.

Die beschriebenen Anforderungen an eine Infrastruktur für die vorgestellte Methodik zur Entwicklung verteilter Systeme sind plattformunabhängig. Jede Plattform, die diesen Voraussetzungen genügt, eignet sich, um verteilte Systeme mit Echtzeitbedingungen bei Ausnutzung asynchroner Kommunikation zu ermöglichen. Bei Verwendung plattformunabhängiger Datentransformations- und -codierungsverfahren ist die Rahmenarchitektur auch für Kommunikation in einem plattformübergreifenden verteilten System geeignet.

Die folgenden Unterabschnitte fassen die Anforderungen der Methodik und Rahmenarchitektur speziell an eine echtzeitfähige und statisch analysierbare Ablaufkoordinierung so wie die Garantien der Netze zusammen. Sie geben einen Überblick für den aktuellen Stand der Technik und begründen Entscheidungen bei der Wahl der Laufzeitumgebung von Echtzeit-Java zur Implementierung eines Prototyps im nächsten Kapitel.

3.4.1 Ablaufkoordinierung

Nebenläufige Kontrollfäden, die auf einem Prozessor "parallel" Arbeit ausführen, müssen bei ihrer Ausführung aufeinander abgestimmt sein. Diese Ablaufkoordinierung wird von der Laufzeitumgebung (evtl. in Kooperation mit dem Betriebssystem) übernommen. Im System ist es notwendig, die Ausführung zu steuern. Deshalb werden für Kontrollfäden Prioritäten zur Abstimmung verwendet: Kontrollfäden mit hoher Priorität sind wichtiger und ihre Ausführung wird bevorzugt.

Um zu verhindern, dass Kontrollfäden mit einer höheren Priorität durch niederprioritäre, die einen Monitor besitzen, in ihrer Ausführung aufgehalten werden ("priority inversion", Prioritätsumkehr), muss die Laufzeitumgebung Mechanismen implementieren, die dies verhindern [97]. Mit Prioritätsvererbung ("priority inheritance") wird bei der Blockade eines höherprioritären Kontrollfadens durch einen Kontrollfaden mit niederer Priorität, der Faden mit geringerer Priorität auf die höhere Ebene gehoben, damit er den Monitor ohne Unterbrechung möglichst bald abgeben kann. Zur Vermeidung von möglichen Verklemmungen ("deadlocks") wird in [23] ein Protokoll mit einem Prioritätshöchstmaß ("priority ceiling") für

jeden Monitor definiert[2].

Prioritäten ermöglichen Kontrollfäden unterschiedlich zu gewichten. Das Laufzeitsystem regelt die Prozessorzuteilung, abhängig von diesen Prioritäten.

Der Einsatz von Prioritäten ist auch bei der Klassifizierung von Nachrichten innerhalb der Kommunikation hilfreich. Durch Definition einer Gewichtung beim Nachrichtenaustausch über einen Kanal ist es so möglich, die Priorität auf Sender- und Empfängerseite für die Verarbeitung zu bestimmen.

Ablaufkoordinierung ("scheduling") regelt die Abwicklung nebenläufiger Kontrollfäden. Es wird eine sequentielle Ablaufsteuerung für alle auszuführenden Kontrollfäden bestimmt und umgesetzt.

In Echtzeitsystemen muss die Ausführung aller Kontrollfäden pünktlich und mit vorhersagbaren Zeitgrenzen erfolgen. Harte Echtzeit in einem verteilten System verlangt, dass für die Kommunikation immer ein Kontrollfaden Zeit hat, der eingehende Nachrichten innerhalb einer harten Zeitgrenze aus dem Empfangspuffer ausliest und verarbeitet. Nebenläufige Kontrollfäden erlauben, diese Aufgaben (Empfang und Verarbeitung) aufzuteilen. Das Verfahren der Ablaufkoordinierung muss die notwendigen Zeitgrenzen für diese Kontrollfäden garantieren. Grundsätzlich besteht ein solches Verfahren aus drei Komponenten [118]:

- Algorithmus zur Planung des Zugriffs auf Ressourcen ("scheduling policy")
- Algorithmus für die Zuteilung der Ressourcen ("scheduling mechanism")
- Analyse des Verhaltens unter Annahme der schlechtesten Voraussetzungen bei gegebener Anordnung des Zugriffs und Zuteilung der Ressourcen ("scheduling/feasibility analysis")

[2] Eine Beschreibung der unterschiedlichen Verfahren zur Vermeidung von Prioritätsumkehr findet sich in Abschnitt 4.1 auf Seite 78 ff.

In Systemen mit harten Echtzeitbedingungen muss das Verfahren zur Ablaufkoordinierung statisch vorher berechenbar sein und eine Analyse erlauben. Sobald dieses Verhalten unter schlechtesten Voraussetzungen berechnet ist, kann es mit den Zeitbedingungen für das System verglichen werden, um sicherzustellen, dass alle Zeitschranken erfüllt werden.

Anforderungen an ein geeignetes Verfahren zur Ablaufkoordinierung werden im Folgenden am Beispiel der zwei Verfahren **Earliest Deadline First** (EDF) und **Fixed Priority Scheduling** (FPS) im Detail vorgestellt und für den Einsatz mit Echtzeitsystemen analysiert. Beide Verfahren arbeiten über die Vergabe von Prioritäten. Der ablaufbereite Kontrollfaden mit der höchsten Priorität wird von der Laufzeitumgebung ausgeführt.

- EDF ist einer der gebräuchlichsten Ablaufkoordinierungsalgorithmen für Echtzeitsysteme. Als zeitbasierter Algorithmus garantiert er die Einhaltung aller Zeitgrenzen. Bei EDF wird einem Kontrollfaden, abhängig von der nächsten zu erreichenden Zeitgrenze, eine Priorität zugeteilt. Der Kontrollfaden mit der kürzesten Frist erhält die höchste Priorität und damit den Prozessor. Wenn solch ein Ablaufplan existiert, der alle Zeitgrenzen erfüllt, ist dieses Verfahren erfolgreich. Um aber einen Ablaufplan zu definieren, der insgesamt den geringsten Schaden mit Nichterfüllung von Grenzen hat, ist EDF völlig ungeeignet. EDF ist sehr flexibel, aber durch die sich ständig ergebenden Prioritätsänderungen komplex in der Analyse und Handhabung.

- FPS beschreibt ein Verfahren der Ablaufkoordinierung über feste Prioritäten. Wenn mehrere Kontrollfäden ausführbereit sind, erhält wieder der mit der höchsten Priorität den Prozessor. Die Priorität der Kontrollfäden wird während der Programmausführung nicht verändert. Dies ermöglicht schon statisch vor der Programmausführung eine Planung auf Einhaltung von harten Zeitgrenzen. Zwei Algorithmen sind hier zum Beispiel "Deadline-Monotonic (D-M) Scheduling" [11] oder "Rate-Monotonic (R-M) Scheduling" [69]. Bei R-M-Ablaufkoordinierung wird die Priorität für Kontrollfäden abhängig von ihrer Periodenlänge berechnet[3]. Dieser Algorithmus ist "optimal", da falls ein möglicher Ablaufplan mit statischen Prioritäten existiert, dann wird durch den R-M-Algorithmus auch ein solcher Plan ermittelt. D-M-Ablaufkoordinierung bestimmt die Prioritäten,

[3] Je kürzer die Periode, desto höher die Priorität.

abhängig von den erlaubten Zeitgrenzen. Für Zeitgrenze gleich Periodenlänge entspricht dieser Ansatz der Lösung mit R-M. Bei der tatsächlichen Auslastung (U) eines Prozessors durch mehrere (m) Kontrollfäden erreichen diese Verfahren zwar eher schlechte Werte, aber es wird garantiert, dass die Leistung immer zur Verfügung steht und alle Zeitbedingungen eingehalten werden. Die folgende Ungleichung zeigt diese Anforderung und für den mathematischen Beweis sei auf die Literatur zu Ablaufkoordinierung seit 1970 (z.B. [69]) verwiesen. C_i ist die schlechteste (längste) Ausführungszeit für den Kontrollfaden i und T_i ist die Periode des entsprechenden Kontrollfadens.

$$U = \sum_{i=1}^{m} (C_i / T_i) \leq m(2^{1/m} - 1) \leq 1$$

Weil R-M-Ablaufkoordinierung nur für periodische Kontrollfäden definiert ist, wurde versucht, diesen Ansatz auf aperiodische Kontrollfäden zu erweitern [101, 10]. Dies ist aber für den Einsatz mit harten Echtzeitbedingungen nur für sporadisch aktive aperiodische Kontrollfäden mathematisch beweisbar.

Da die vorgestellte Methodik neben Entwurf und Implementierung auch die statische Analyse verteilter Systeme vor ihrer Ausführung unterstützen soll, muss bei der Ablaufkoordinierung für nebenläufige Kontrollfaden ein Verfahren mit fest vergebenen und vorhersagbaren Prioritäten verwendet werden.

Vom Einsatz des Verfahrens für die Ablaufkoordinierung mit festen Prioritäten ("Fixed Priority Scheduling") lässt sich außerdem die Unterstützung für periodische und sporadische Kontrollfäden ableiten. Oft werden diese Arten von Kontrollfäden schon von echtzeitfähigen Betriebssystemen unterstützt und die notwendige objektorientierte Laufzeitumgebung kann die Kontrollfäden und die Kontrollfadenverwaltung deshalb einfach auf die Betriebssystemressourcen und -dienste abbilden.

3.4.2 Netzzugriff und Kontrollfäden

Grundlage für ein sicherheits- und geschäftskritisches verteiltes System mit Echtzeitbedingungen ist die Verfügbarkeit eines echtzeitfähigen Netzes (vgl. Abschnitt 2.6 auf Seite 28ff.) in jedem Kommunikationsknoten. Um eine möglichst generische Integration von Echtzeitnetzen zu erreichen,

wird nur die Funktionalität einer ISO/OSI Transportschicht vorausgesetzt. Dies standardisiert den Zugriff auf zugrundeliegende Netze und für die meisten Standard-Hardwarekomponenten existieren entsprechende Zugriffsbibliotheken.

Für die effiziente Netzanbindung muss der Empfang einer Nachricht von ihrer Verarbeitung getrennt werden. Mit einem Kontrollfaden pro verfügbares Kommunikationsnetz, der mit höchster Priorität auf Daten am Netzzugangspunkt wartet, wird der Empfang von Nachrichten geregelt. Die Laufzeitumgebung erlaubt während der Wartezeit anderen Kontrollfäden mit niedrigerer Priorität die Verarbeitung von schon empfangenen Nachrichten.

Um unvorhersehbare Verzögerungen durch die automatische Speicherbereinigung in einer objektorientierten Laufzeitumgebungen zu verhindern, verwendet die vorgestellte Rahmenarchitektur beim Empfang von Nachrichten mit harten Echtzeitbedingungen nur fest allozierte Objekte. Mit Laufzeitumgebungen, die automatische Speicherbereinigung unter harten Echtzeitbedingungen erlauben [99, 100], sind auch dynamisch erzeugte Objekte denkbar.

3.5 Softwareentwurfsmuster für die Kommunikationsrahmenarchitektur

Die Rahmenarchitektur für asynchrone Nachrichtenkommunikation mit Echtzeitbedingungen orientiert sich am Softwareentwurfsmuster Ereigniskanal (“event channel” [15, 112]). Die Ereignisse innerhalb des Entwurfsmusters werden als vollständige Nachrichten verstanden. Und obwohl der Prototyp in der englischen Bezeichnung “EventChannelNetwork” [32] heißt, ist als deutsche Übersetzung eher “Nachrichtenkanal-Netz” geeignet.

Das Softwareentwurfsmuster Ereigniskanal entkoppelt die Teilnehmer an einem Gesamtsystem vollständig voneinander, so dass sie völlig eigenständig arbeiten können und über die Existenz oder Anzahl anderer Teilnehmer nichts wissen. Interaktion erfolgt über Ereignisse. Teilnehmer registrieren sich am Ereigniskanal, in dem sie angeben, bei welchen Ereignissen sie benachrichtigt werden sollen. Wenn ein Teilnehmer ein Ereignis (evtl. mit Daten) an den Ereigniskanal sendet, leitet dieser das Ereignis an die dafür registrierten Teilnehmer weiter [112].

Dieses Muster ist die Grundlage für die Kommunikation innerhalb der vorgestellten Rahmenarchitektur. Die Ausnutzung von Netzen und Feldbussystemen mit Rundruflogik und Anforderungen für die Erfüllung von Echtzeitbedingungen mit einer objektorientierten Laufzeitumgebung ergeben die nachfolgend beschriebenen Anpassungen.

Die Rahmenarchitektur des Nachrichtenkanal-Netzes unterstützt verteilte Systeme durch einen asynchronen Nachrichtendienst mit direkter Publiziere-/Abonniere-Kommunikation. Die Echtzeitbedingungen zugrundeliegender Rundrufnetze und der verwendeten Laufzeitumgebung werden in der Anwendungsschicht bereitgestellt. Mit einer separaten XML-Datei können Struktur und Verbindung der Netzknoten, die Implementierung der Nachrichtenbehandlung und Zeitbedingungen der Nachrichtenkommunikation beschrieben werden. Diese Informationen sind Basis für einen Programmcodegenerator sowie die Erzeugung eines Modells für statische Korrektheitsanalysen der Ablaufkoordinierung, der bei der Kommunikation verwendeten Kontrollfäden.

Durch Nutzung der Rahmenarchitektur ist es möglich, verteilte Systeme mit asynchroner Nachrichtenkommunikation zu entwickeln. Anders als in herkömmlichen Kommunikationsinfrastrukturen mit Publiziere-/Abonniere-Logik üblich, erfolgt die Kommunikation direkt ohne eine Vermittlungskomponente. Der Abonnent eines themenbasierten Nachrichtenkanals kennt Zeitgrenzen und Übertragungsparameter des Publizisten. Er verwendet die Programmbibliothek der Rahmenarchitektur, um den Empfang der Nachrichten des asynchron sendenden Publizisten sicherzustellen. Die Verarbeitung der empfangenen Daten wird vom Empfang getrennt und asynchron nach Zwischenspeicherung durchgeführt. Für die Einhaltung von Echtzeitbedingungen bei dieser Verarbeitung sind die von der Laufzeitumgebung bereitgestellte Ablaufkoordinierung und die geeignete Festlegung von Prioritäten der notwendigen Kontrollfäden verantwortlich.

In den folgenden Unterabschnitten werden für dieses Verfahren notwendige Komponenten, Kontrollfäden und ihr Zusammenspiel beschrieben.

3.5.1 Nachrichtenkanal

Der Nachrichtenkanal (“EventChannel” in Abbildung 3.2, analog zum Ereigniskanal-Muster) ist die zentrale Komponente im vorgestellten Nachrichtendienst. Ein Nachrichtenkanal repräsentiert das Thema der Publiziere-/Abonniere-Kommunikation. Ein passendes Objekt dieser Klasse

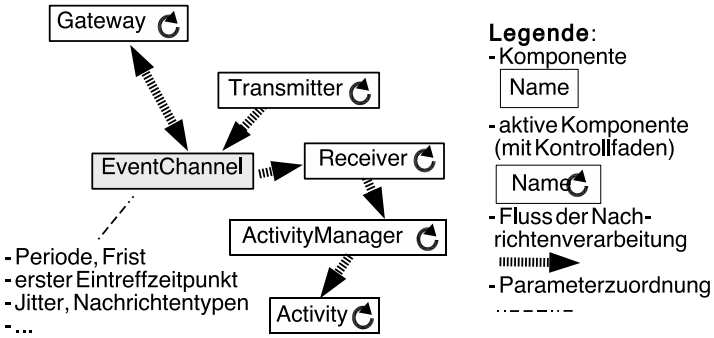

Abbildung 3.2: Nachrichtenkanal und zugeordnete verarbeitende Einheiten (mit Legende)

muss bei der Kommunikation auf Sender- und Empfängerseite bekannt sein. Nachrichten werden damit als "zum Kanal" bzw. "zum Thema" gehörend beschrieben und verwaltet. Nachrichten eines Kanals werden außerdem über einen Nachrichtentypen als Bezeichner unterschieden. Für einen Kanal verfügbare Nachrichtentypbezeichner werden im Nachrichtenkanal-Objekt definiert. Ansonsten wird dieses Objekt in jedem Kommunikationspartner zur Konfiguration beteiligter aktiven Komponenten[4], die Nachrichten senden, empfangen und verarbeiten, genutzt. Ihre Aufgaben werden im Folgenden beschrieben.

3.5.2 Empfänger-Kollektiv

Das Empfänger-Kollektiv ist als Einzelstück für einen Kommunikationsknoten implementiert und umfasst die aktiven Komponenten, die für das Empfangen von Nachrichten zuständig sind. Abbildung 3.3 zeigt das Kollektiv ("ReceiverCollective") mit den enthaltenen Empfängerkontrollfäden ("Receiver") in einem Klassenstrukturdiagramm zusammen mit anderen Komponenten der Rahmenarchitektur, die am Empfang von Nachrichten beteiligt sind. Nach dem Empfang einer Nachricht wird diese zur weiteren Verarbeitung in eine nach Prioritäten sortierte Warteschlange ("PriorityQueues") eingereiht. Empfängerkontrollfäden werden genau einem Kommunikationssockel ("Socket") zugeordnet. Sie werden mit höchster Priorität ausgeführt und blockieren, wenn am Sockel keine Nachricht verfügbar ist[5]. Mehrere Empfängerkontrollfäden sind notwendig, um den parallelen

[4]Komponenten mit einem eigenen Kontrollfaden

[5]Der Kommunikationssockel bietet eine sychrone Form des Nachrichtenaustauschs und der blockierende Empfangskontrollfaden erlaubt der eigentlichen Applikation den

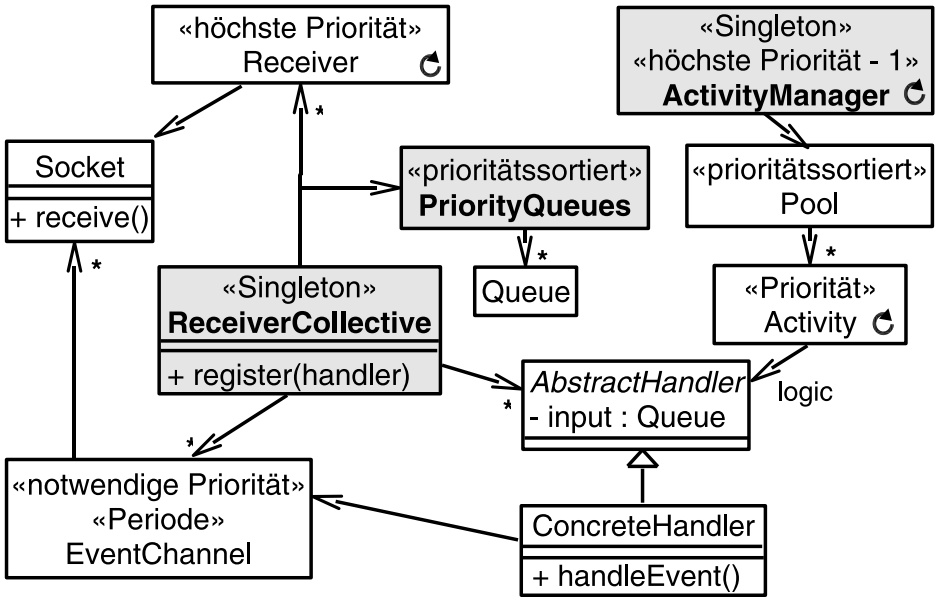

Abbildung 3.3: Klassenstrukturdiagramm: Rahmenarchitektur des Nachrichtenkanal-Netzes für Empfang mit Echtzeitbedingungen

Zugriff auf unterschiedliche physikalische Kommunikationsnetze zu ermöglichen. Beim Empfang unterschiedlicher Nachrichten mehrerer Nachrichtenkanäle über ein Netz reicht ein Kontrollfaden. Die Analyse der Ablaufkoordinierung, die in Abschnitt 3.7 beschrieben wird, stellt sicher, dass nebenläufig eintreffende Nachrichten mit den verfügbaren Kontrollfäden für die gegebene Laufzeitumgebung wirklich zu empfangen sind.

3.5.3 Kommunikationssockel

Der Zugriff auf das dem Nachrichtenkanal-Netz zugrunde liegende physikalische Netz wird über einen einfachen Byte-orientierten Kommunikationssockel ermöglicht. Diese Sockel bieten den Lese/Schreibe-Zugriff auf das Kommunikationsnetz, das im wesentlichen Funktionalität, vergleichbar zu Schicht 4 des ISO/OSI Referenzmodells für Kommunikation (Transportschicht), anbietet. Es wird kein zuverlässiges Protokoll als Basis vorausgesetzt.

Die Applikation ist für die Fehlerbehandlung verantwortlich und zwei grundsätzliche Verfahren dafür sind denkbar. Bei periodisch erwarteten

asynchronen Nachrichtenaustausch.

Nachrichten (z.B. ein Lebenssignal des Senders im Produktionssystem in Abschnitt 1.2) muss der Empfänger das Nicht-Eintreffen einer Nachricht erkennen und eine Fehlerbehandlung starten. Wenn die Nachricht aperiodisch und selten gesendet wird, muss der Sender sicherstellen (z.B. durch eine erforderliche Rückantwort des Empfängers), dass die Nachricht nicht verloren gegangen ist.

Bei Verwendung von Echtzeitnetzen ermöglicht der Kommunikationssockel, ohne weiteren Protokollaufwand, die Ausnutzung der Echtzeitgarantien. Für einen Vergleich möglicher Kommunikationsnetze und -protokolle siehe Abschnitt 2.6. Es ist denkbar, dass der Kommunikationssockel neben der direkten Interaktion mit dem zugrundeliegenden Netz weitere Funktionalität für die Aufteilung und Verschmelzung von Nachrichtenpaketen in Netztransporteinheiten anbietet. Voraussetzung für den Einsatz mit Echtzeitbedingungen ist, dass dieser zusätzliche Protokollaufwand begrenzt und nach oben abschätzbar bleibt.

3.5.4 Aktivitätsmanager für Verwaltung von Aktivitäten

Der Aktivitätsmanagerkontrollfaden (“ActivityManager”) ist für die Abarbeitung der Nachrichten in den Warteschlangen (“PriorityQueues”) des Kommunikationsknotens verantwortlich. Er ist ebenfalls als Einzelstück implementiert und garantiert die Zuordnung wartender, bereits empfangener Nachrichten an passende Aktivitätskontrollfäden (“Activity”) aus seinem Reservoirspeicher (“Pool”). Kontrollfäden mit passender Priorität werden aktiviert, Fristen abhängig vom Eintreffzeitpunkt der Nachricht angepasst und die Abarbeitung mit der registrierten Bearbeitungslogik (“ConcreteHandler”) für die Nachrichten gestartet.

3.5.5 FIFO-Warteschlangen

Die Interaktion aller aktiven Komponenten wird durch eine Kopplung mit FIFO-Warteschlangen (“Queue”) unterstützt. Sowohl zwischen Kontrollfäden des Empfänger-Kollektivs und dem Aktivitätsmanagerkontrollfaden einerseits, als auch bei der Zuteilung von Nachrichten an Aktivitätskontrollfäden andererseits wird dieses Verfahren eingesetzt.

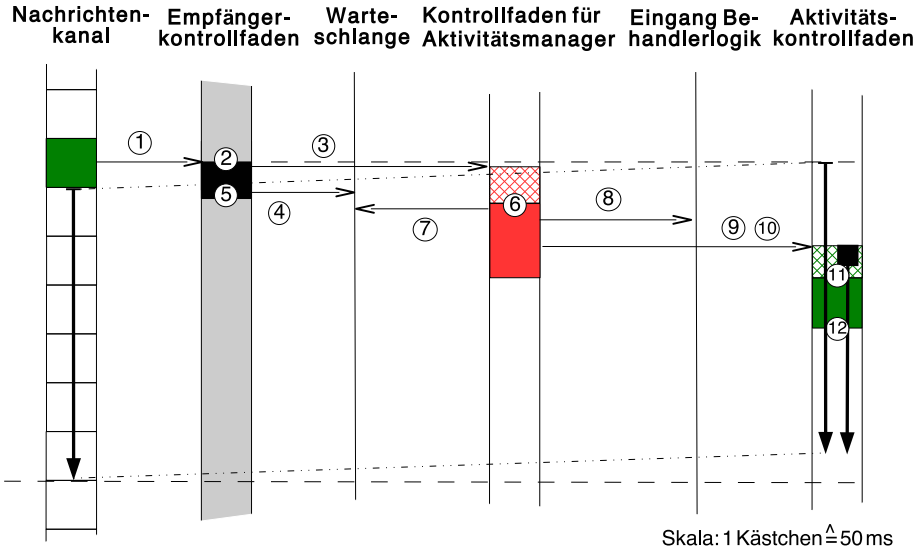

Abbildung 3.4: Nachrichtenempfang mit Fristneuberechnung des Aktivitätskontrollfadens

3.5.6 Zusammenspiel beim Nachrichtenempfang

Da sich die von der Rahmenarchitektur sichergestellten Echtzeitbedingungen auf eine fristgerechte Verarbeitung eingetroffener Nachrichten beziehen, wird der Eintreffzeitpunkt einer Nachricht mit einem Zeitstempel festgehalten. In der Abbildung 3.4 wird wieder ein farbig markierter Bereich für die Größe der Ungenauigkeit, des Zitterns beim Empfang verwendet. Der Aktivitätskontrollfaden, der die Nachricht zur Verarbeitung in die Eingangswarteschlange seiner Behandlerlogik geschrieben bekommt wird erst später gestartet und die Rahmenarchitektur nutzt den Zeitstempel um seine Verarbeitungsfrist durch die Laufzeitumgebung überwachen zu lassen. Die Aktivitätsmanagerkomponente ist neben der Zuteilung von Nachrichten an Aktivitäten auch für die Neuberechnung dieser Echtzeitkontrollfadenfristen[6] zuständig. Die Arbeit des Aktivitätsmanagers ergibt sich aus Kopierarbeiten und dieser Neuberechnung. Abbildung 3.4 zeigt am Beispiel des Temperatursensornachrichtenkanals die Interaktion der beteiligten Kontrollfäden und die Berechnung der angepassten Verarbeitungsfrist. Die im Nachfolgenden aufgelisteten Aktionen werden den in

[6]Die Abbildung zeigt die Verkürzung der Frist für den Aktivitätskontrollfaden mit zwei Pfeilen. Das Quadrat markiert den neuen Aktivierungsbeginn des Kontrollfadens durch die Laufzeitumgebung.

der Abbildung dargestellten Kontrollfäden (Empfängerkontrollfaden (E), Kontrollfaden für den Aktivitätsmanager (M) und Aktivitätskontrollfaden (A)) beziehungsweise der Rahmenarchitektur und Laufzeitumgebung (R) zugeordnet. In der Abbildung markieren einfarbige Abschnitte die Ausführung eines Kontrollfadens. Schraffierte Abschnitte zeigen Kontrollfäden, die in der Ausführung durch einen anderen Kontrollfaden blockiert sind. Der Empfängerkontrollfaden - mit der höchsten Priorität - wird durch das eigene Warten auf neue Nachrichten - im hellgrauen Abschnitt blockiert.

1. Eintreffen einer Nachricht im Eingangspuffer der Netzzugangsschnittstelle (R)
2. Aktivierung des Empfängerkontrollfadens (R)
3. Benachrichtigung des Aktivitätsmanagers (E)
4. Lesen der Nachricht: Kopieren der Attributwerte in das Warteschlangensystem (E)
5. Warten[7] des Empfängerkontrollfadens auf eine neue Nachricht (E)
6. Aktivierung des Aktivitätsmanagerkontrollfadens (R)
7. Lesen der empfangenen Nachrichten (M)
8. Kopieren der Nachricht in den Eingang der Behandlerlogik (M)
9. Neuberechnung der Frist für die erfolgreiche Verarbeitung ausgehend vom Zeitstempel der Nachricht (M)
10. Neuberechnung[8] und Aktivierung des Ablaufplans (R)
11. Aktivierung des Aktivitätskontrollfadens (R) und Verarbeitung der Nachricht (A)
12. Ende der erfolgreichen Verarbeitung innerhalb der vorgegeben Frist (R)

[7] blockierendes Lesen

[8] Da die Verarbeitungsfrist eines Aktivitätskontrollfadens tatsächlich nur verkürzt wird und die ursprünglich geplante Frist bestehen bleibt, kann der Ablaufplan nicht ungültig werden. Statische Korrektheitsanalysen der Ablaufkoordinierung bleiben gültig.

3.5.7 Weitere aktive Komponenten für die Kommunikation

Neben Empfängerkontrollfäden wird auf den Kommunikationssockel mit einer Komponente für die Versendung ("Transmitter") und eventuell für die Netzverbindung ("Gateway") unterschiedlicher zugrunde liegender Kommunikationsnetze zugegriffen (vgl. Abbildung 3.2 auf Seite 57). Analog zu den Empfängerkontrollfäden werden die Parameter dieser aktiven Komponenten durch den Nachrichtenkanal vorgegeben. Wenn ein Ablaufplan von Kontrollfäden für einen Kommunikationsknoten analysiert werden soll, dann müssen auch diese Kontrollfäden berücksichtigt werden.

3.5.8 Verwendete Entwurfsmuster bei Gestaltung der Rahmenarchitektur

Die soweit eingeführten Komponenten können in einem neuen Architekturentwurfsmuster zusammengefasst werden. Durch Trennung von Empfang und Verarbeitung, mit Kontrollfäden in einem Empfänger-Kollektiv, einem Aktivitätsmanagerkontrollfaden und der Nutzung von Warteschlangen wird asynchrone Nachrichtenkommunikation um die Einhaltung von Echtzeitbedingungen erweitert. Das so eingeführte Entwurfsmuster erfordert keine spezielle asynchrone Netzzugriffs- oder Betriebssystemunterstützung. Der Entwurf ist ähnlich des Reaktormusters [92] und basiert ebenfalls auf einem synchronen Nachrichten-Demultiplexer. Durch die Erweiterung mit der Aktivitätsmanagerkomponente und prioritätssortierten Warteschlangen können mehrere Nachrichtenkanäle mit überlappend eintreffenden und zu verarbeitenden Nachrichten berücksichtigt werden.

Das im letzten Absatz beschriebene Entwurfsmuster basiert im Wesentlichen auf der Integration schon bekannter Muster.

Reaktor [94, 121]: Das Reaktor-Muster ist mit der hier vorgestellten Architektur verwandt und erlaubt ebenfalls asynchrone Nachrichtenkommunikation auf Basis der Umkehr des Kontrollflusses. Mit diesem als Hollywood Prinzip: "Don't call us, we'll call you!" bekannten Muster ist es möglich, dass der Reaktor synchron auf Nachrichten mehrerer Kommunikationspartner wartet und beim Eintreffen einer Nachricht, diese mit registrierten Behandlungsmethoden verarbeitet. Der hier vorgestellte Ansatz erweitert das Muster durch den Einsatz von FIFO-Warteschlangen, um unterschiedliche Prioritäten in der Nachrichtenkommunikation zu unterstützen.

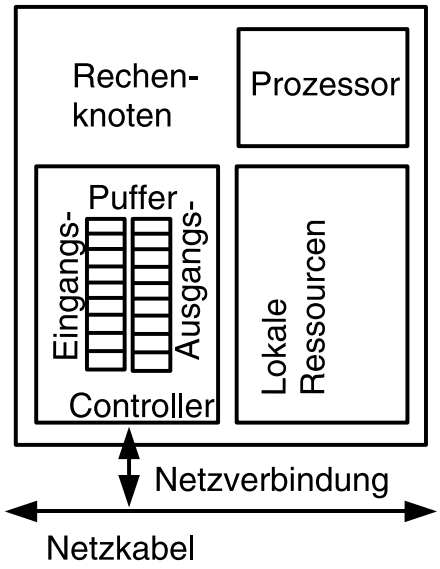

Abbildung 3.5: Netzzugangsschnittstelle

Beobachter [37]: Der Beobachter basiert ebenfalls auf dem Rückrufprinzip, aber das Muster ist weniger auf Nachrichtenkommunikation als auf direkte Methodenaufrufe ausgerichtet.

Aktivator [73]: Das zum Beispiel in der Java 2 Enterprise Edition oft verwendete Muster eines Aktivators[9] beschreibt den in dieser Architektur vorgesehenen Aktivitätsmanager. Sein Kontrollfaden steuert die Verarbeitung der asynchron eingetroffenen Nachrichten durch einen Aktivitätskontrollfaden aus einem Reservoirspeicher.

Der vorgestellte Nachrichtendienst ist die Grundlage für den Entwurf verteilter Systeme, die eine asynchrone, themenbasierte Publiziere/ Abonniere-Kommunikation verwenden. Die Rahmenarchitektur unterstützt ein direktes "Push"-Nachrichtenübertragungsmodell ohne zentrale Diensteinheiten.

Die mit einem Nachrichtenkanal verbundenen Nachrichten dürfen nur in periodischer oder sporadischer[10] Häufigkeit eintreffen. Mit dieser Einschränkung wird eine Abbildung auf die in der Laufzeitumgebung verfügbaren periodischen und sporadischen Kontrollfäden möglich. Die zugeordneten Empfängerkontrollfäden sind für den direkten Empfang bzw.

[9]Dort wird die Aktivierung von Geschäftsobjekten im Applikationsdienstgeber bei Bedarf geregelt.

[10]mit garantierten Empfangszwischenzeiten

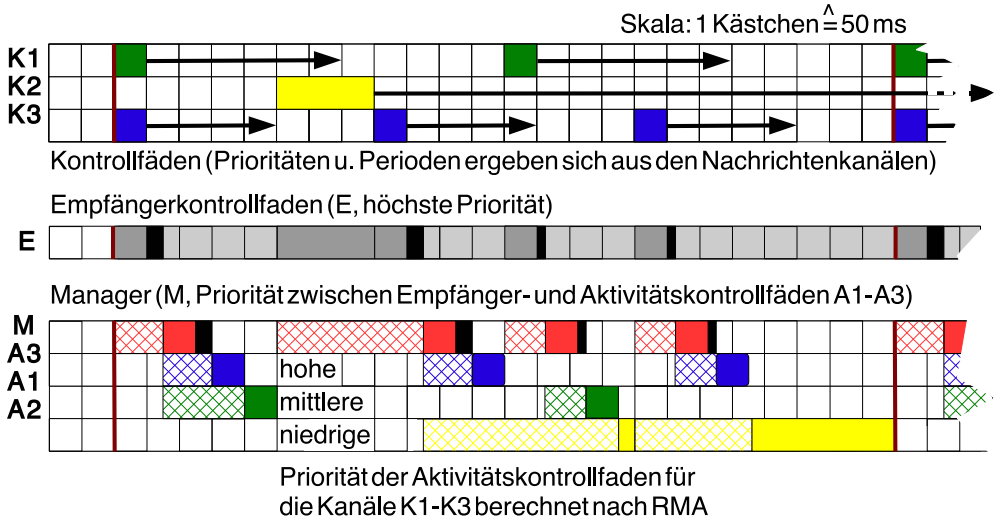

Abbildung 3.6: Kontrollfäden für Empfang von Nachrichten aus Beispiel in Abbildung 1.1 auf Seite 4

die zeitnahe Leerung des Eingangspuffers der Netzzugangsschnittstelle ("communication network interface", vgl. Abbildung 3.5) und die Einreihung der Nachrichten in ein nach Prioritäten sortiertes Warteschlangensystem verantwortlich.

Durch Nutzung eines Verfahrens der Ablaufkoordinierung mit festen Prioritäten (vgl. Abschnitt 3.4.1) kann für Systeme mit harten Echtzeitbedingungen statisch die Erfüllung der Bedingungen für Zeitgrenzen garantiert werden. Es ist möglich, periodisch und sporadisch eintreffende Nachrichten mittels Aktivitätskontrollfäden zu behandeln, die nach RMA ("rate monotonic analysis" [62]) vergebene Prioritäten besitzen.

Das in Abschnitt 1.2 vorgestellte Beispielszenario eines Produktionssystems mit drei Senderknoten erfordert in der Steuerungseinheit[11] den Nachrichtenempfang von drei Kanälen und die Verarbeitung mit Echtzeitbedingungen. Abbildung 3.6 zeigt noch mal das Empfangsmuster der verwendeten Nachrichtenkanäle und illustriert die Prioritäten der für Empfang und Verarbeitung notwendigen Kontrollfäden: Empfängerkontrollfaden (E), Kontrollfaden für den Aktivitätsmanager (M) und drei Aktivitätskontrollfäden ($A1$-$A3$).

- Der Empfängerkontrollfaden arbeitet mit höchster Priorität. Er be-

[11] 4. Knoten, ein Empfänger

obachtet seinen[12] Netzzugangspunkt des Echtzeitnetzes, über den Nachrichten empfangen werden. Ein helles Grau während der gesamten Ablaufzeit zeigt, dass der Empfängerkontrollfaden blockiert und jederzeit von der Netzzugangsschnittstelle durch eine eintreffende Nachricht aktiviert werden kann. Dank der höchsten Priorität wird er umgehend aktiv und kann die Nachricht entgegennehmen. Dunkelgraue Abschnitte zeigen geplante Empfangsfenster für einen oder mehrere Nachrichtenkanäle[13]. Überlappende Bereiche der Ungenauigkeit (Jitter) für periodische Eintreffzeitpunkte von Nachrichten werden so zusammengefasst. Schwarze Zeitfenster zeigen im Anschluss den für die Kopieraktivität beim Einreihen in das Warteschlangensystem notwendigen Aufwand. Alle Angaben ergeben sich aus den schlechtesten zu erwartenden Aktivitätszeiten, die für die Implementierung der Rahmenarchitektur für eine gegebene Laufzeitumgebung und Hardware berechnet oder geschätzt werden müssen.
Tatsächlich ist der Empfang von Nachrichten eine möglichst kurze Aktivität und die höchste Priorität des Empfängerkontrollfadens garantiert die sofortige Verarbeitung; auch den Empfang von Nachrichten außerhalb ihres geplanten Eintreffzeitpunktes.

- Der Kontrollfaden des Aktivitätsmanagers[14] wird nach einem Nachrichtenempfang durch den Empfängerkontrollfaden aktiviert. Seine schlechteste (d.h. größte) anzunehmende Zeitspanne, in der er aktiv sein muss, beginnt zeitgleich zum Empfangszeitfenster für Nachrichten. Da der Kontrollfaden mit einer Priorität kleiner der maximalen Priorität arbeitet, kann er seine Arbeit, die eingegangenen Nachrichten an Aktivitätskontrollfäden zuteilen, aber erst durchführen, wenn der Empfängerkontrollfaden wieder blockiert. In rot schraffierten Zeitspannen wartet der Kontrollfaden darauf, dass der Prozessor ihn ausführen kann. Rote Zeitfenster zeigen, wann er Arbeit leisten kann und die schwarzen Zeitanteile beschreiben diesmal den Aufwand beim Kopieren der Nachricht aus dem Warteschlangensystem in die Eingangswarteschlange einer Bearbeitungslogik[15]. Auch

[12] Das Empfänger-Kollektiv besteht für komplexere Vernetzungen aus mehreren Empfängerkontrollfäden, einem pro Kommunikationsnetz.

[13] Diese lange Zeitspanne ergibt sich aus den Angaben für die Jitter der Nachrichtenkanäle. Für den tatsächlichen Empfang ist wesentlich weniger Prozessorarbeit erforderlich.

[14] Vergleiche dazu die rote Zeile für den Kontrollfadenzustand.

[15] Dieser Wert berechnet sich analog zu den Angaben beim Empfängerkontrollfaden

für den Kontrollfaden des Aktivitätsmanagers gilt, dass diese Arbeit so früh wie möglich (sobald der Prozessor verfügbar ist) ausgeführt wird. Die Angaben im Diagramm ergeben sich aus den schlechtesten möglichen Aktivitätszeiten.
Da die Arbeit als Reaktion auf mehrere Nachrichtenkanäle mit unterschiedlicher Periode notwendig ist, kann der Kontrollfaden des Aktivitätsmanagers nicht mit einer festen Periode arbeiten. Um die Arbeit des Kontrollfadens bei der statischen Prüfung der Ablaufkoordinierung dennoch zu planen wird der größten gemeinsame Teiler (ggT) der Perioden aller Nachrichtenkanäle betrachtet. Die so berechnete Zeitspanne kann als Mindestzwischenzeit zwischen zwei Aktivierungszeitpunkten für einen sporadisch aktiven Kontrollfaden aufgefasst werden. Für das Beispiel in Tabelle 1.1 auf Seite 6 sieht man, dass $ggT(600, 1200, 400) = 200$ so eine einfach zu berechnende untere Grenze der notwendigen Aktivierungszeitpunkte darstellt.

- $A3$, $A1$ und $A2$ beschreibt drei Aktivitätskontrollfäden, die für die Nachrichtenverarbeitung der drei Nachrichtenkanäle reserviert sind. Für die Zuordnung der drei Nachrichtenkanäle zu diesen Kontrollfäden für die Verarbeitung werden hier die Farben aus den ersten drei Zeilen wieder aufgegriffen. Die Prioritäten der Kontrollfäden sind nach RMA verteilt: kürzere Periode bewirkt höhere Priorität[16]. Die Kontrollfäden werden parallel zu den frühesten Eintreffzeitpunkten der zu empfangenden Nachrichten periodisch aktiviert. Die schraffierten Zeitfenster zeigen jetzt die Phase vor Empfang und Zuteilung, in der der jeweilige Kontrollfaden von Kontrollfäden mit höherer Priorität an der Verarbeitung gehindert wird. Erst die farbigen Zeitfenster beschreiben die Nachrichtenverarbeitung. Für die realistische Analyse der Ablaufkoordinierung (vgl. Abschnitt 3.7) ist es notwendig, dass hier die maximale Zeitdauer im schlechtesten Fall der Verarbeitung verwendet wird. Das Beispiel des Aktivitätskontrollfadens $A2$ (für den Nachrichtenkanal $K2$ mit der längsten Periode) verdeutlicht, dass die Verarbeitung durch wichtigere Nachrichten unterbrochen sein kann. RMA garantiert, dass alle Kontrollfäden ihre Fristen einhalten können.

Ausgehend von dem in Abbildung 3.6 dargestellten Modell für die Kontrollfäden, die für den Empfang von Nachrichten in der Steuereinheit des

über die schlechteste (d.h. längste) Kopierarbeit der größten möglichen Nachricht.

[16]Die Prioritäten sind dabei geringer als die für den Kontrollfaden des Aktivitätmanagers.

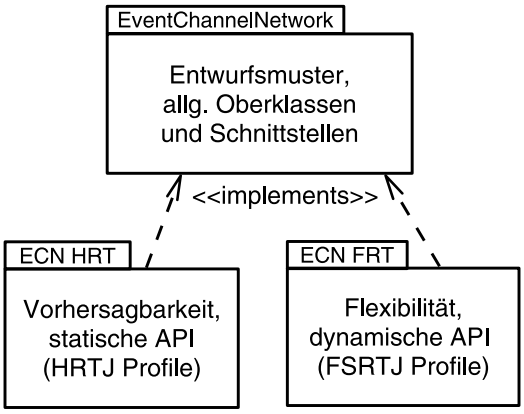

Abbildung 3.7: Versionen für harte und flexible Echtzeitbedingungen

Beispiels notwendig sind, wird dieses Modell im Abschnitt 3.7 für die statische Analyse des Gesamtsystems vor der Ausführung erweitert. Da die Rahmenarchitektur lediglich periodische und sporadische Kontrollfäden verwendet, erlaubt eine Ablaufumgebung mit einem Verfahren für die Ablaufkoordinierung von Kontrollfäden mit festen Prioritäten diese statische Analyse von harten Echtzeitbedingungen.

Da der Nutzen asynchroner Kommunikation beim Entwurf verteilter Systeme nicht nur für verteilte Systeme mit harten Echtzeitbedingungen erkennbar ist, wird für das in diesem Kapitel vorgeschlagene Software-Entwurfsmuster auch eine Implementierung für verteilte Systeme mit weichen Echtzeitbedingungen geplant. Wesentlicher Vorteil der vorgestellten Publiziere-/Abonniere-Kommunikation ist die Flexibilität beim Entwurf verteilter Systeme. Durch Nutzung auch von Nicht-Echtzeitnetzen mit Rund- oder Gruppenruflogik, wie zum Beispiel UDP/IP, können einfach verteilte Systeme mit Nachrichtenkanälen beschrieben und implementiert werden. Abbildung 3.7 illustriert zwei Versionen des Nachrichtenkanal-Netzes (“EventChannelNetwork”, ECN) für harte und flexible[17] Echtzeitbedingungen. Die Implementierungen beider Versionen der Rahmenarchitektur werden mit den Unterschieden der erforderlichen Echtzeit-Java Profile (HRTJ, FSRTJ) im nächsten Kapitel 4 beschrieben.

[17] weiche und harte

3.5.9 Grenzen und mögliche Erweiterungen

Die vorgestellte Rahmenarchitektur wurde in Hinblick auf den Lösungsansatz der geplante Methodik aus Abschnitt 3.2 und unter Berücksichtigung des aktuellen Standes der Technik entworfen. Der Entwurf ist dabei speziell auf Netze mit Rundruflogik und die Verwendung in einer objektorientierten Laufzeitumgebung ausgerichtet. Die Ausnutzung asynchroner Nachrichtenkommunikation und die Applikationsentwicklung mit generierten Rahmen vereinfacht zwar die Entwicklung verteilter Applikationen mit Echtzeitbedingungen bei der lokalen Verarbeitung, aber andere Freiheiten im Systementwurf werden eingeschränkt.

Obwohl asynchrone Nachrichtenkommunikation mit Publiziere-/Abonniere-Logik eine flexible Applikationsstruktur unterstützt, werden durch Forderungen nach harten Echtzeitbedingungen diese wieder beschränkt. Die Notwendigkeit einer statischen Analyse der Ablaufkoordinierung verlangt die statische Festlegung und Beschreibung möglicher Kommunikationsknoten und Interaktionen. Die Abstraktion von Nachrichtenkanälen erfordert und ermöglicht dies.

Die Netzanbindung über Kommunikationssockel mit nur einem Empfängerkontrollfaden pro Kommunikationsknoten vereinfacht den Empfang (Einreihung von Nachrichten in ein prioritätssortiertes Warteschlangensystem), aber die Gleichbehandlung aller Nachrichten unabhängig ihrer Priorität hat für Systeme mit einem “unschönen” Nachrichtenaufkommen auch Nachteile. “Unschöne” Systeme verwenden Nachrichten (und Nachrichtenkanäle) sehr unterschiedlicher Priorität. So kann der Empfang vieler niederprioritärer Nachrichten (eines sogenannten “babbling idiot”) den Empfang von Nachrichten hoher Priorität behindern. Solche Kommunikationsmuster mehrerer Nachrichtenkanäle sind eventuell nicht für nur ein physikalisches Kommunikationsnetz ausgelegt. Ein Verfahren um dieses Problem zu verhindern wäre der Einsatz von Prioritätsbändern [80]. Die Implementierung verkompliziert aber den bisher noch einfachen Entwurf. Es werden mehrere Empfängerkontrollfäden pro Kommunikationssockel mit unterschiedlichen Prioritäten eingeführt. Nur die wichtigen Nachrichten hoher Priorität werden durch den Empfängerkontrollfaden mit höchster Priorität behandelt. Der Verlust niederpriorer Nachrichten beim Empfang mit einem Empfängerkontrollfaden geringerer Priorität wird in Kauf genommen. Dieses Verfahren stellt auch bei einem geeigneten (d.h. ”schönen”) Nachrichtenaufkommen den Empfang aller Nachrichten sicher. Für einen Kommunikationssockel wird so eine Partitionierung der Rechenkapazität, abhängig von Prioritäten der verfügbaren Nachrichtenkanäle, er-

möglicht. Auf eine solche Erweiterung der Rahmenarchitektur ist für die prototypische Implementierung im nächsten Kapitel aufgrund des erheblichen Mehraufwands verzichtet worden.

Sicherheit ist ein weiterer Aspekt, der im aktuellen Entwurf nicht vorgesehen ist. Sie ist für verteilte Systeme zwar wesentlich, aber die Rahmenarchitektur bietet noch keine spezielle Unterstützung und fordert dies auch nicht von der Laufzeitumgebung. Sicherheit wird als Aufgabe der Applikationslogik verstanden. Mit Einsatz asymmetrischer Kryptosysteme kann durch Verwendung öffentlicher und privater Schlüssel die notwendige Vertraulichkeit und Integrität bei der Nachrichtenkommunikation und Identifikation zwischen Kommunikationspartnern implementiert werden.

3.6 Methode zur Systembeschreibung

Für die Entwicklung verteilter Applikationen, die die vorgestellte Rahmenarchitektur mit einer asynchronen Nachrichtenkommunikation nutzen, wird hier ein neues Verfahren vorgestellt, das die Beschreibung der Kommunikation (Nachrichtenkanäle und Behandler) einzelner Kommunikationsknoten eines Systems für die Programmierung vorsieht. Die damit verbundene Methodik unterstützt die Entwicklung durchgängig und dieser Abschnitt führt die notwendige Methode zur Systembeschreibung ein.

Die Beschreibung erfolgt durch Konfigurationsdateien, die zur Laufzeit ausgewertet werden oder deren Systembeschreibung mit einem Generator in Programmcode zur Übersetzung für die Ausführung mit einer objektorientierten Laufzeitumgebung umgewandelt werden kann. Abschnitt 4.4 zeigt die konkrete Implementierung des Prototyps mit Java-Klassen und XML-Beschreibungsdateien.

Wird der Systementwurf durch einen Programmier- und Entwicklungsassistenten (PEA, z.B. ein Eclipse-Plug-In, vgl. dazu die Arbeit von Januar [60]) geleitet, ermittelt dieser zuerst die notwendige Anzahl von Kommunikationsknoten. Abbildung 3.8 zeigt den notwendigen Benutzerdialog der Systemkonfiguration als Beispiel. Abhängig von der Unterscheidung, ob ein Kommunikationsknoten als Publizist oder Abonnent agiert, sind weitere Parameter obligatorisch. Für Publizisten müssen Informationen zum bereitgestellten Nachrichtenkanal und für Abonnenten Referenzen auf die Programmklassen zur Verarbeitung von Nachrichten eingege-

ben werden. In jedem Kommunikationsknoten wird die Verbindung zum physikalischen Kommunikationsnetz mittels einer Referenz auf die Treiberklasse des Zugriffsockels dargestellt. Eventuell physikalische Parameter, wie Portnummer oder Datenzwischenspeichergröße werden hier beschrieben. Die Beschreibung der für Versand oder Empfang verwendeten Nachrichtenkanäle umfassen folgende Parameter:

- **Bezeichner des Kanals**. Für jede Bezeichnerzeichenkette wird auch ein eindeutiger Zahlenwert verlangt, der bei der Kommunikation die Kanalbezeichnung kodiert.
- **Erster Eintreffzeitpunkt**, **Periode**, **Jitter** und **Bearbeitungsfrist** beschrieben das Sendeverhalten über den Nachrichtenkanal sowie die erforderliche Frist für die Verarbeitung nach Empfang. Kanäle für sporadisches Senden von Nachrichten verlangen statt der Periode die Mindestzwischenzeit von zwei Nachrichtensendungen.
- **Priorität** beschreibt die Wichtigkeit der gesendeten Nachrichten.
- **Nachrichtentypen**, die in diesem Kanal unterschieden werden können. Hier muss auch die Größe der unterschiedlichen Typen angegeben werden, um für Sende- und Empfangskopierarbeiten eine Verarbeitungszeit im schlechtesten Fall abschätzen zu können.

Für die Beschreibung und Anbindung der Verarbeitungslogik auf Abonnentenseite sind die folgenden Parameter notwendig:

- **Behandlerbezeichner** und **Bezeichner des Kanals**, dessen Nachrichten verarbeitet werden können. Für die Kodierung ist wieder die Abbildung auf eindeutige Zahlenwerte erforderlich.
- **Kapazität** der notwendigen Eingangswarteschlange für die Verarbeitung von mehreren aufgelaufenen Nachrichten.
- **Maximale Verarbeitungszeit** für Nachrichten im schlechtesten anzunehmenden Fall.
- **Anzahl** notwendiger paralleler Verarbeitungskontrollfäden.

Die hier aufgelisteten Parameter erlauben die generische Beschreibung von Nachrichtenkanälen und Behandlerlogiken. Notwendige Konfigurationsdateiformate können als eine domänenspezifische Sprache (vgl. Abschnitt 2.1 Für dieauf Seite 16ff.) beim Entwurf von verteilten Systemen

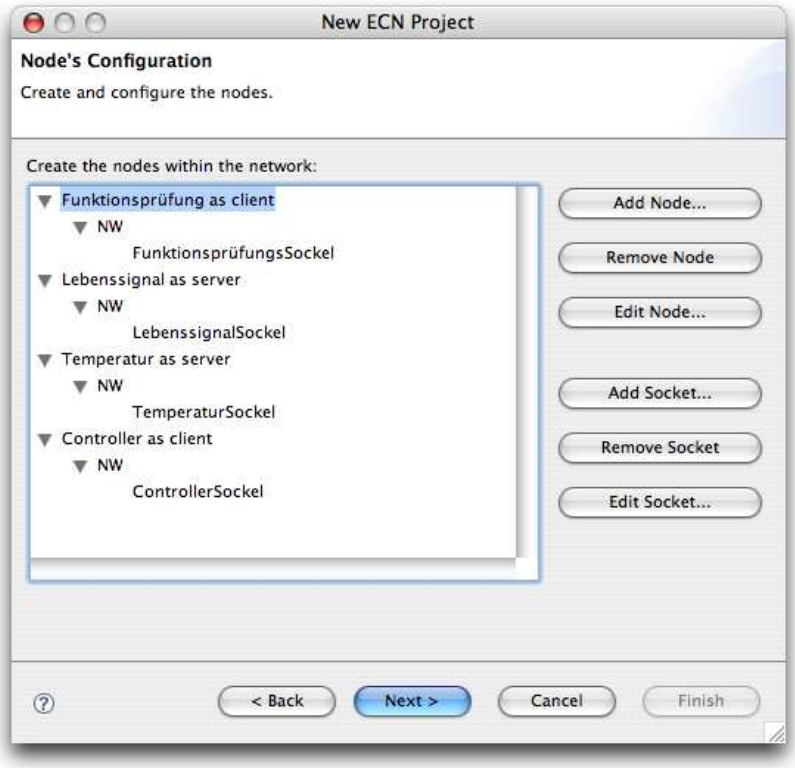

Abbildung 3.8: Systemkonfiguration über Eclipse-Plug-In

mit asynchroner Kommunikation verstanden werden. Das Implementierungskapitel 4 beschreibt in Abschnitt 4.4 die DTD eines entsprechenden XML-Formats. Da Informationen (z.B. über die Kanalbeschreibung) auf Publizisten- und Abonnentenseite identisch vorhanden sein müssen, lohnt sich der Einsatz eines PEA, der die Entwicklung unterstützt und konsistent halten kann.

3.7 Analyse der Ablaufkoordinierung

Als Erweiterung der schon im Abschnitt 3.5.8 auf Seite 62ff. zusammen mit den verwendeten Entwurfsmustern eingeführten Beschreibung der Ablaufkoordinierung von Kontrollfäden für den Empfang wird in diesem Abschnitt die statische Analyse mit einem Modell beschrieben. Das Modell für die Analyse kann mit Wissen über die Kommunikationsrahmenarchitektur aus den Informationen der Systembeschreibung generiert werden.

Der in Abschnitt 3.2 vorgestellte Lösungsansatz der Methodik beschreibt den Nutzen für die Integration einer Analyse. Durch die Vorgabe einer Rahmenarchitektur für komponentenbasierte verteilte Systeme ist die konsistente Generierung des Programmcodes für die Kommunikation möglich. Hierfür kann ein äquivalentes Teilmodell für die Analyse generiert werden. Programmcode und Modell enthalten die für asynchrone Nachrichtenkommunikation notwendigen Echtzeitkontrollfäden. Weitere Echtzeitkontrollfäden des Gesamtsystems, zum Beispiel in der Geschäftslogik, muss der Programmierer selbst hinzufügen.

Durch die Analyse des Modells der Echtzeitkontrollfäden in jedem Kommunikationsknoten kann die Einhaltung der Verarbeitungsfristen für die verbundenen Nachrichtenkanäle des verteilten Systems statisch überprüft werden. Außer dem in der Systembeschreibung gegebenen Kommunikationsverhalten für diese Nachrichtenkanäle sind für diese Analyse auch Abschätzungen für im schlechtesten Fall anzunehmende Aktivitäts- oder Ausführungszeit (“worst case execution time”, WCET) aller beteiligten Programmteile erforderlich.

Abschnitt 2.9 beschreibt die Notwendigkeit solcher Ausführungsanalysen und Zeitabschätzungen für den Einsatz mit sicherheits- und geschäftskritischen Systemen. Da die Analyse aber nur als Stand der Technik in der vorgestellten Methodik eingesetzt wird, wird für notwendige Arbeiten bei der Anpassung von Werkzeugen für Ausführungszeitabschätzungen im

ungünstigsten Fall (WCETA) auf die Erfahrungen mit dem Analysewerkzeug Gromit [51] in den Java- und Echtzeit-Projekten HIDOORS und HIJA verwiesen [58, 57].

Die hier vorgestellte Methodik nutzt dieses Verfahren. Für die eingesetzte Ablaufumgebung aus Hardware und Laufzeitumgebung können so notwendige Abschätzungen verwendet werden. Notwendige Zeitabschätzungen beziehen sich dabei auf:

- Arbeit der Kontrollfäden in der Programmbibliothek der Rahmenarchitektur, sowie
- Arbeit der Bearbeitungslogik, die im vorgegebenen Programmrahmen leicht identifiziert werden kann.

Die Zeitabschätzungen sind Teil des Analysemodells für die Ablaufkoordinierung der Echtzeitkontrollfäden in den einzelnen Kommunikationsknoten. Bei der Analyse nutzt die Dissertation ebenfalls Erfahrungen des EU-Forschungsprojektes HIJA. Wie in der Ergebnisbeschreibung zur Analyse der Ablaufkoordinierung ("Schedulability Analysis" [55]) beschrieben, wird hier eine Analyse für ein Einprozessorsystem mit unterbrechender ("preemtive") Verwaltung von Echtzeitkontrollfäden auf Basis von festen Prioritäten eingesetzt.

Für diese Analyse eignet sich die MAST-Suite [72] von Werkzeugen für die Analyse der Ablaufkoordinierung in Echtzeitsystemen. Die Nutzung selbst und das Format der notwendigen Steuerdatei für dieses Werkzeug ist Teil der Implementierung und wird erst in Abschnitt 4.5 beschrieben. Im Folgenden wird nur die Eignung des mit MAST verwendeten linearen Analysemodells für den Anwendungsfall mit Kommunikationsknoten für den Empfang mehrerer Nachrichtenkanäle exemplarisch am bereits eingeführten Beispielszenario untersucht.

Grundsätzlich soll mit einem Modell der Kontrollfäden in den einzelnen Laufzeitumgebungen eines Kommunikationsknoten geprüft werden, ob alle Echtzeitkontrollfäden ihre Fristen einhalten[18] können. Für Systeme mit harten Echtzeitbedingungen ist dies von der Methodik vor der Ausführung vorgesehen.

Abbildung 3.9 zeigt wieder den Steuerknoten des Produktionssystems aus Abschnitt 1.2. Das Modell basiert auf den Kontrollfäden der Abbildung 3.6

[18] Für Aktivitätskontrollfäden werden hier konkrete Verarbeitungsfristen der empfangenen Nachrichtenkanäle verwendet.

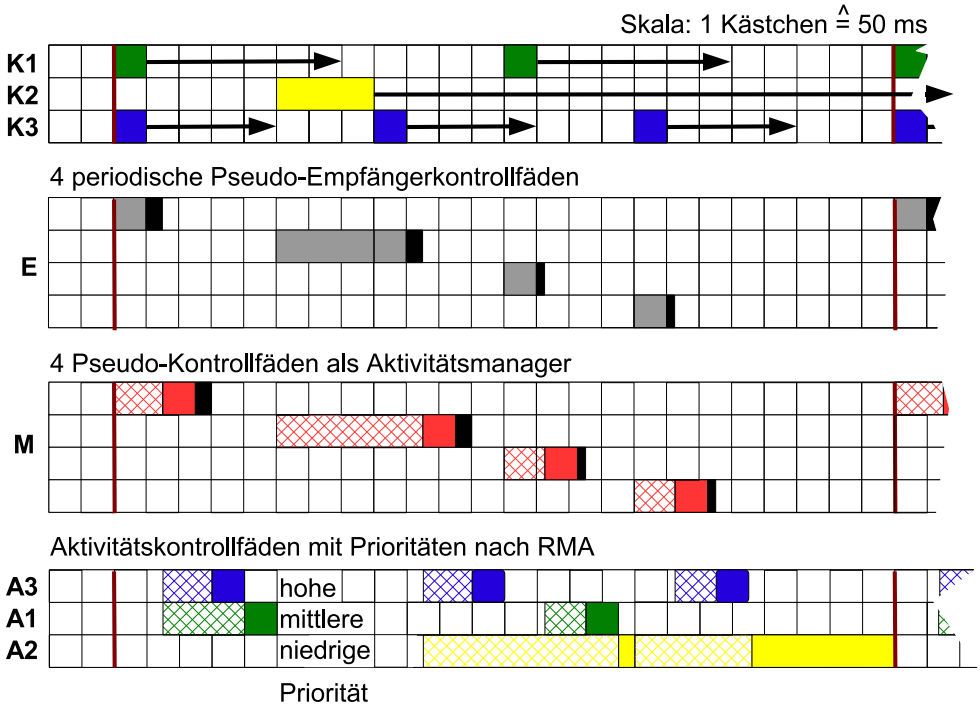

Abbildung 3.9: Pseudo-Kontrollfäden für Empfang von Nachrichten aus Beispiel in Abbildung 1.1 auf Seite 4

auf Seite 64. Im Unterschied zu der Abbildung dort werden hier mehrere Pseudo-Empfänger- und Pseudo-Aktivitätsmanagerkontrollfäden beschrieben. Je vier Kontrollfäden für Empfang und Verwaltung ergeben sich aus den für die Eintreffzeitpunkte der Nachrichtenkanäle berechneten Empfangsintervallen einer Runde. Es sind periodische Echtzeitkontrollfäden ohne Überlappung ihrer Ausführungszeit. Die Länge der Periode entspricht der berechneten Runde[19]. Die statische Analyse von Verfahren zur Ablaufkoordinierung mit festen Prioritäten werden in [48] und [46] beschrieben. Aus den Beschreibungsparametern (Periode, erster Eintreffzeitpunkt, Frist und Jitter) eines Nachrichtenkanals werden Eingabeparameter für die Analyse errechnet.

Da die Analyse nebenläufiger Nachrichten für das MAST-Werkzeug noch nicht implementiert ist und [38] nur die Abbildung für ein Modell mit nebenläufigen Nachrichten auf ein lineares Modell von MAST beschreibt, wird das tatsächliche Modell der Echtzeitkontrollfäden in der Rahmenarchitektur transformiert. Es werden Ausführungszeiten mit der schlechtes-

[19] kleinstes gemeinsame Vielfache (kgV) der Perioden aller Nachrichtenkanäle

ten (d.h. längsten) anzunehmenden Dauer verwendet. Die erlaubte Ungenauigkeit beim Eintreffen einer Nachricht (Jitter) bestimmt die notwendige Aktivitätszeit des Empfängerkontrollfadens. Überlappende Jitter-Zeitspannen werden im Modell zur Aktivitätszeit eines periodischen Echtzeitkontrollfadens zusammengefasst. Um den Zusammenhang zwischen externen Nachrichten (im Modell entspricht das allen möglichen Startzeitpunkten der Pseudo-Empfängerkontrollfäden) und der weiteren linearen Verarbeitung zu ermöglichen, wird die Aktivität des tatsächlichen Empfängerkontrollfadens und des Kontrollfadens für den Aktivitätsmanager für diese Startzeitpunkte vervielfacht. Da diese Kontrollfäden nicht überlappen, wird bei der Berechnung des Analysewerkzeugs nur die tatsächlich notwendige Arbeit verwendet. Es wird keine Ungewissheit über mögliche unterschiedliche Kontrollfadenserialisierungen in der schlechtesten Form der Ausführung aufsummiert. Abschnitt 4.5 in der Implementierung der Rahmenarchitektur zeigt die Beispielberechnung der verwendeten Produktionssteuerung und verdeutlicht so den Einsatz des Verfahrens.

3.8 Zusammenfassung und durchgängiger Einsatz im Beispielszenario

Für den Entwurf von verteilten Systemen mit Echtzeitbedingungen beschreibt dieses Kapitel einen Lösungsansatz für die durchgängige Softwareentwicklung. Die beschriebene Rahmenarchitektur zur Nutzung asynchroner Kommunikation unterstützt diese Methodik. Es wird eine plattformunabhängige Architekturbeschreibung mit bekannten Softwareentwurfsmustern entworfen. Die Eignung von objektorientierten Laufzeitumgebungen wird durch eine Anforderungsanalyse näher untersucht.

Durch Verwendung des in Abschnitt 1.2 auf Seite 1.2ff. vorgestellten Beispielszenarios kann die durchgängige Unterstützung durch die Methodik gezeigt werden. Das System kann mit Verwendung der in Tabelle 1.1 auf Seite 6 aufgelisteten Kommunikationsparametern beschrieben werden. Die Kanalbeschreibungen der Knoten 1-3 werden in den Empfängerknoten kopiert und konfigurieren die Kommunikationsinfrastruktur für den nebenläufigen Empfang der Nachrichten.

Der Einsatz einer echtzeitfähigen Laufzeitumgebung mit einem Verfahren für die Ablaufkoordinierung von nebenläufigen Echtzeitkontrollfäden mit festen Prioritäten ist erforderlich und die vorgestellte Rahmenarchitektur unterstützt dann die statische Analyse auf Einhaltung aller Verarbeitungs-

fristen (gegeben in den Kanalbeschreibungen) durch die Generierung von Programmcode für die Kommunikation und ein Teilmodell dieser Kommunikation für ein Analysewerkzeuge.

Die Art der zu verarbeitenden Nachrichten (d.h. das Verhalten der Kommunikationskanäle) zeigt dabei die Grenzen der Methodik und Rahmenarchitektur. Ergebnis der Analyse für die Ablaufkoordinierung der Kontrollfäden verwendet Perioden und Verarbeitungszeiten wie sie im Systementwurf beschrieben wurden. Dass diese Zeiten einen funktionierenden Ablaufplan der Kontrollfäden beschreiben kann die Methodik nicht garantieren. Durch Einsatz von leistungsfähigerer Hardware und Ablaufumgebung kann dies verbessert werden, eine Bewertung und Verbesserung der Konfiguration abhängig der Analyseergebnisse ist möglich wird, in der ersten Version der Entwicklungsmethodik aber nicht vorgesehen.

Die hierfür genutzte Systembeschreibung mittels Konfigurationsdateien erlaubt den Entwurf verteilter Systeme mit Angaben von Echtzeitbedingungen und Beschreibung des Kommunikationsverhaltens.

Ausgehend von den vorgestellten Lösungen und Entwürfen wird im nächsten Kapitel die Implementierung eines konkreten Prototyps umgesetzt. Als Laufzeitumgebung wird Echtzeit-Java untersucht und eine Systembeschreibung durch XML-Dateien definiert. Die Analyse von Echtzeitbedingungen wird mit Nutzung der MAST-Suite mit Werkzeugen für Echtzeitsysteme praktisch beschrieben.

Kapitel 4

Implementierung einer Rahmenarchitektur

Die Implementierung der Rahmenarchitektur hängt zusammen mit der durch die EU geförderten Arbeit für zwei Forschungsprojekte. Für "High Integrity Distributed Object-Oriented Realtime Systems" (HIDOORS, IST 2001-32329) und "High Integrity Java" (HIJA, IST 2003-511718) wurde dabei der Einsatz von Echtzeit-Java bei eingebetteten sicherheits- und geschäftskritischen verteilten Systemen um asynchrone Kommunikation erweitert. Details zu den beiden Forschungsprojekten finden sich im Unterabschnitt 5.1 bei der Evaluierung der Prototypen.

Ausgehend von der plattformunabhängigen Beschreibung und einer Methodik für die durchgängige Unterstützung der Entwicklung verteilter Systeme wird in diesem Kapitel die Implementierung des Prototyps auf Basis von Echtzeit-Java (vgl. Abschnitt 4.1) und XML (vgl. Abschnitt 4.4) erläutert. Da das Anwendungsfeld für sicherheits- und geschäftskritische verteilte Systeme groß und variantenreich ist, existieren für den Prototyp mit Java-Technologie mehrere Versionen. Alle Versionen des Nachrichtenkanal-Netzes folgen den eingeführten Software-Entwurfsmustern. Abschnitt 4.3 beschreibt, analog zu den im Forschungsprojekt HIJA ermittelten Profilen für Echtzeit-Java, eine Version für harte und eine für weiche Echtzeitbedingungen (vgl. Abbildung 3.7 auf Seite 67).

Besonders mit der Implementierung für harte Echtzeitbedingungen vereinfacht die vorgestellte Methodik den Softwareentwicklungsprozess. Applikationen für sicherheits- und geschäftskritische verteilte Systeme mit harten Echtzeitbedingungen verlangen Garantien, die durch ausführliche Tests nur schwierig vollständig zu erbringen sind (vgl. Stand der Technik

in Abschnitt 2.9). Abschnitt 4.5 beschreibt die praktische Verwendung statischer Analyse in jedem Kommunikationsknoten, um notwendige Zertifizierungsverfahren für verteilte Applikationen zu vereinfachen und die Korrektheit der Applikation vor der Ausführung prüfen zu können.

4.1 Anforderungen an Echtzeit-Java

Die nebenläufige Ausführung von Kontrollfäden ist fest in die Java Sprache und Laufzeitumgebung integriert [41]. Für den Einsatz von Java in sicherheits- und geschäftskritischen Applikationen muss diese Eigenschaft um Echtzeitbedingungen erweitert werden. Die hierfür entwickelte Spezifikation der Echtzeiterweiterung RTSJ [117] definiert deshalb spezielle Echtzeitkontrollfäden und koordiniert ihre Ausführung mit einem Ablaufplan. Bei nebenläufiger Ausführung von Kontrollfäden mit unterschiedlichen Prioritäten erlaubt die Java-Laufzeitumgebung die Unterbrechung ("preemption") von Kontrollfäden, um die Ausführung solcher Kontrollfäden mit höherer Priorität zu beschleunigen. Der konkurrierende Zugriff auf gemeinsam benutzte Ressourcen ist in Java (und RTSJ) mit gegenseitigem Ausschluss über Monitore möglich.

Diese Eigenschaften von Echtzeit-Java als objektorientierter Laufzeitumgebung eignen sich für den Einsatz mit der in Kapitel 3 vorgestellten Softwarearchitektur eines asynchronen Nachrichtendienstes bei harten und weichen Echtzeitbedingungen. Aus der genaueren Betrachtung der Spezifikation [18] ergeben sich für den Einsatz mit harten Echtzeitbedingungen in sicherheits- und geschäftskritischen verteilten Systemen dennoch einige Änderungswünsche. Auch die bestehende Praxis im Umfeld der Softwareentwicklung für solche Systeme hat Auswirkungen auf die hier vorgestellte Implementierung. Die notwendigen Änderungen werden im Folgenden kurz aufgelistet und danach detaillierter beschrieben.

- Ablaufkoordinierung mit festen Prioritäten, um die statische Analyse von Echtzeitbedingungen zu unterstützen und die Einhaltung von Zeitfristen vor der Ausführung überprüfen zu können.
- Verfahren gegen Prioritätsumkehr mittels Prioritätshöchstmaßen für Monitore, um Verklemmungen bei der Abarbeitung nebenläufiger Kontrollfäden zu verhindern.
- Echtzeitkontrollfäden mit periodischer und sporadischer Wiederholung, um die nebenläufige Abarbeitung mit einem Verfahren für die

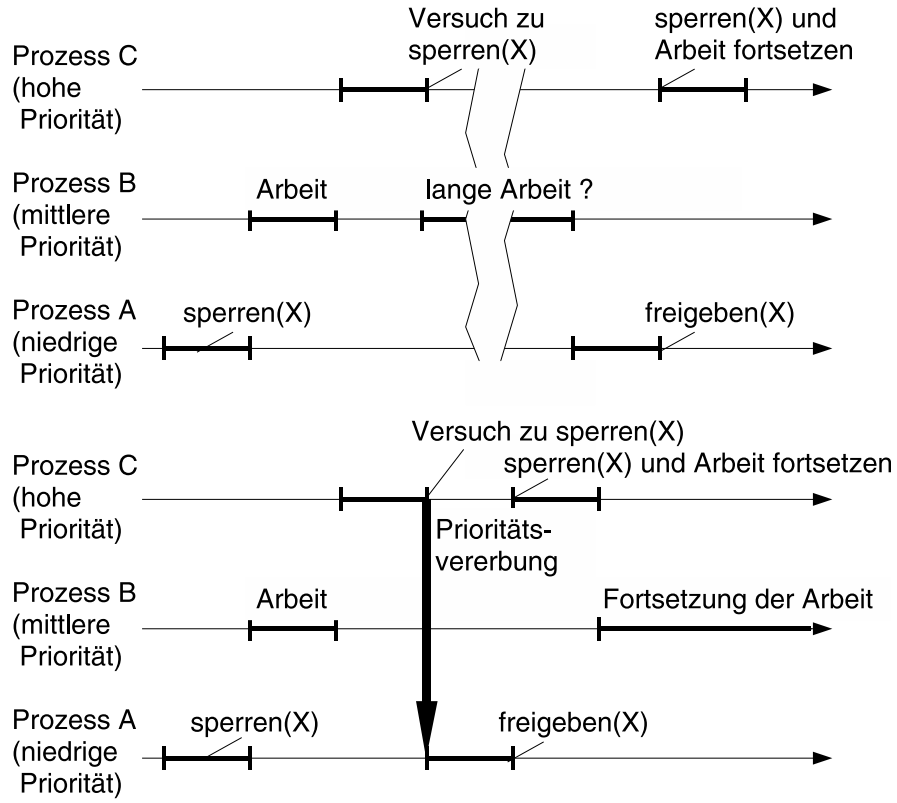

Abbildung 4.1: Problem der Prioritätsumkehr und Behebung mit Prioritätsvererbung

Ablaufkoordinierung mit Prioritäten planbar zu machen. Behandler für Fristverletzungen können in der Implementierung des Rahmenwerks ausgenutzt werden.

- Vermeidung von automatischer Speicherbereinigung, um unvorhersagbare Speicherfreigabezeiten zu verhindern.
- Serialisierung in harten Echtzeitfristen, um die Ausführungszeit bei der Kommunikation begrenzen zu können.
- Trennung von Initialisierungs- und Ausführungsphase, um notwendige Objekterzeugungszeiten in eine weniger zeitkritische Initialisierungsphase verlagern zu können.

Der Einsatz einer Ablaufkoordinierung mit festen Prioritäten und die Unterstützung der Unterbrechung von Echtzeitkontrollfäden durch die Lauf-

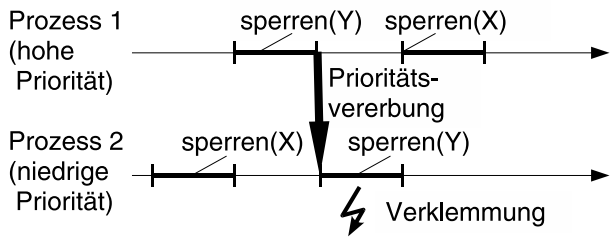

Abbildung 4.2: Verklemmung zweier Prozesse mit Prioritätsvererbung

zeitumgebung sind gegeben. Wenn ein Prozess mit höherer Priorität ausführbar ist, wird der gerade ausgeführte Prozess unterbrochen. Im Zusammenhang mit Monitoren für gegenseitigen Ausschluss bei Kontrollfäden wird aber in der Spezifikation als Verfahren zur Vermeidung von Prioritätsumkehr nur Prioritätsvererbung verlangt. Dieses Verfahren wurde schon in Abschnitt 3.4.1 vorgestellt und Abbildung 4.1 verdeutlicht nochmals das Problem und die Lösung:
Prozess[1] *A* und Prozess *C* benötigen dort den gleichen Monitor *X*. Da der niederpriore Prozess *A* den Monitor erst wieder freigeben kann, nachdem ein anderer Prozess *B* seine Arbeit erledigt hat, wird die Priorität von Prozess *C* tatsächlich auf das niedrige Niveau von Prozess *A* reduziert. Durch diese Prioritätsumkehr wird Prozess *C* erst nach dem weniger wichtigen Prozess *B* mit mittlerer Priorität und Prozess *A* abgeschlossen. Prioritätsvererbung verhindert dies. Die niedrige Priorität von Prozess *A* wird – bis zur Freigabe des Monitors – auf das höhere Prioritätsniveau von Prozess *C* gehoben und die Freigabe des Monitors beschleunigt.

Abbildung 4.2 illustriert aber auch einen Nachteil dieses Verfahrens. Beim Zugriff auf zwei Monitore (in unterschiedlicher Reihenfolge) wird zwischen nieder- und höherprioritären Prozessen eine Verklemmung nicht verhindert. In der Abbildung hält Prozess 2 den Monitor *X* und seine Priorität wird für eine schnelle Freigabe auf das Prioritätsniveau von Prozess 1 gehoben. Da Prozess 2 aber für seine Abarbeitung auch den Monitor *Y* benötigt und dieser von Prozess 1 gehalten wird, führt die Veränderung der Priorität nun in die Verklemmung.

Für den Einsatz mit statisch vergebenen festen Prioritäten ist deshalb eine Lösung mittels Prioritätshöchstmaß notwendig.

Abbildung 4.3 zeigt die Verwendung eines solchen Höchstmaßes für die Priorität beim Zugriff auf einen Monitor. So werden Verklemmungen ver-

[1] Prozess und Kontrollfaden sind in dieser Betrachtung äquivalent zu sehen.

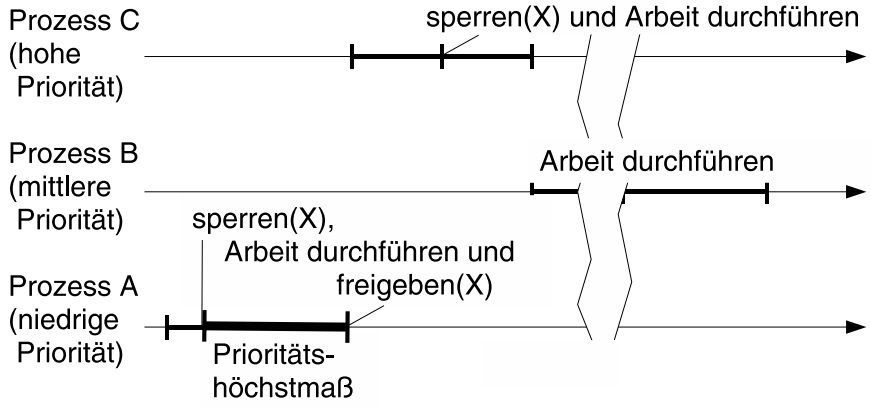

Abbildung 4.3: Prioritätshöchstmaß zur Vermeidung von Prioritätsumkehr und Verklemmungen

hindert. Der Prozess *A* muss beim Sperren von Monitor *X* die höchste Priorität aller jemals zugreifenden Prozesse besitzen, um bei der Abarbeitung nicht unterbrochen zu werden. Der Zugriff auf alle Monitore muss deshalb im Vorfeld der Ausführung statisch berechnet werden und eine Vergabe der Prioritäten geplant werden. Zusammen mit der Analyse auf Einhaltung aller Echtzeitbedingungen und Zeitfristen kann dies von Werkzeugen wie zum Beispiel MAST geleistet werden (vgl. Abschnitt 4.5).

Ein weiterer kritischer Punkt für die Auswahl der Laufzeitumgebung ist die Verwaltung der Ressource Speicher und eine eventuelle automatische Speicherbereinigung. Java unterstützt dies zwar, aber in der Echtzeiterweiterung ist automatische Speicherbereinigung durch Einführung neuer Speicherbereiche reduziert worden. Alle Speicherbereiche werden dafür unter der Klasse `javax.realtime.MemoryArea` in der RTSJ Programmbibliothek zusammengefasst. Der schon in Standard-Java verfügbare Speicherbereich der Halde ("heap") ist als `javax.realtime.HeapMemory` verfügbar. Hier wird weiterhin automatische Speicherverwaltung unterstützt. Keine solche Speicherbereinigung wird im dauerhaften ("immortal", Klasse: `javax.realtime.ImmortalMemory`) und in kurzlebigeren bereichsdefinierenden ("scoped", Klasse: `javax.realtime.ScopedMemory`) Speicherbereichen verlangt. Alle Objekte im dauerhaften Speicherbereich werden zur Laufzeit der Applikation nicht wieder freigegeben. Ein `ScopedMemory`-Bereich wird, wenn der letzte Kontrollfaden, der in diesem Bereich Objekte erzeugt oder verwendet, den Bereich verlässt[2], dieser Speicherbereich

[2]Die RTSJ Programmbibliothek sieht für Betreten und Verlassen von solchen Speicher-

vollständig freigegeben. Die Zusicherung einer linearen Allokationszeit[3] erlaubt es, die Ausführungszeit bei der Objekterzeugung in solchen Speicherbereichen nach oben zu begrenzen. Unterschiedliche Speicherbereinigungsverfahren (keine, vollständige oder automatische Freigabe) in den mit RTSJ verwalteten Speicherbereichen bestimmen die Verwendung von Objekten in diesen Bereichen. Grundsätzlich gilt, dass ein Objekt nur auf andere Objekte verweisen darf, wenn diese eine längere Lebensdauer besitzen. Auf Objekte im `ImmortalMemory`-Bereich darf also immer verwiesen werden und der Austausch von Daten über solche Objekte ist sicher. Die Dauerhaftigkeit dieser Objekte erfordert aber ihre Wiederverwendung und der Austausch von Informationen kann nur über das Kopieren und Ändern der Attributwerte erfolgen. Die Verwendung von `ImmortalMemory`- und `ScopedMemory`-Speicherbereichen und Pufferobjekten für die Implementierung der Rahmenarchitektur wird im Abschnitt 4.3 im Detail beschrieben. Da Verfahren der automatischen Speicherbereinigung im Umfeld sicherheits- und geschäftskritischer Systeme mit harten Echtzeitbedingungen (noch) nicht akzeptiert sind [34], beschränkt sich die Implementierung für harte Echtzeitbedingungen auf die Verwendung dieser Speicherbereiche.

Das Verfahren der Ablaufkoordinierung wird in RTSJ mit einem Fabrikmuster in der Klasse `javax.realtime.Scheduler` und ihren Unterklassen definiert. Die Spezifikation sieht standardmäßig nur eine Unterklasse `PriorityScheduler` mit einem Verfahren für Ablaufkoordinierung mit festen Prioritäten vor. Andere Verfahren wie z.B. die Ausführung von Kontrollfäden mit frühen Zeitgrenzen zuerst ("Earliest Deadline First") sind denkbar [28], aber die Java-Implementierungen der Rahmenarchitektur verwendet eine Steuerung mit `PriorityScheduler`.

Echtzeitkontrollfäden sind in RTSJ mit `javax.realtime.RealtimeThread` und (deren Unterklasse) `javax.realtime.NoHeapRealtimeThread` verfügbar. Kontrollfäden des Typs `NoHeapRealtimeThread` können nicht auf den Speicherbereich der Halde zugreifen und Objekte, die sie verwenden, unterliegen nicht der automatischen Speicherbereinigung. Die Implementierung der Rahmenarchitektur für harte Echtzeitbedingungen verwendet diese Echtzeitkontrollfäden. Bei der Erzeugung eines Kontrollfadens wird ein Parameterobjekt für die Priorität (`PriorityParameters`) angegeben. Dieser Parameter enthält einen Zahlenwert für die Priorität des Echtzeitkontrollfadens und wird im Verfahren der Ablaufkoordinierung mit festen Prioritäten genutzt. Durch Angabe eines `ReleaseParameters`-Objekts wer-

bereichen spezielle Methoden vor.

[3] linear zur Objektgröße

den die Häufigkeit und andere Parameter (z.B. Zeitfrist, maximale Kosten, etc.) für die periodische (Unterklasse `PeriodicParameters`) oder sporadische (Unterklasse `SporadicParameters`) Wiederholung der Aktivierung des Kontrollfadens bestimmt. Bei periodischen Echtzeitkontrollfäden wird diese Aktivierung durch die Laufzeitumgebung gesteuert. Der Kontrollfaden wird periodisch als "ausführbar" markiert und wenn er die höchste Priorität der ausführbaren Kontrollfäden besitzt, dann wird er durch den Prozessor ausgeführt. Diesen gibt er ab, sobald er selbst die Methode `waitForNextPeriod()` aufruft oder er seine Ausführungsfrist überschritten hat. Bei Überschreiten einer Zeitfrist wird eine bei der Erzeugung registrierte Fehlerbehandlungslogik (`AsyncEventHandler`) gestartet. In der Methode `handleAsyncEvent()` wird zum Beispiel die Neuberechnung eines Ablaufplans implementiert. Wie dieser Mechanismus sich für die Implementierung der Rahmenarchitektur mit harten und weichen Echtzeitbedingungen eignet, wird in Abschnitt 4.3 genauer beschrieben. Für eine detailliertere Einführung in den Einsatz der RTSJ für nebenläufige Programmierung mit Echtzeitkontrollfäden und dem Verfahren der Ablaufkoordinierung mit festen Prioritäten wird zum Beispiel "Concurrent and Real-Time Programming in Java" von Andy Wellings [117] empfohlen.

Beim Einsatz von Netzkommunikation wird auf die Standard-Objektserialisierung – zugunsten einer auf Speicherplatz und Rechenaufwand optimierten Serialisierung der Inhalte von Objekten – verzichtet. Haumacher [49] beschreibt die Laufzeit der Java-Objektserialisierung für die Datenübertragung in einer verteilten Umgebung für Rechnerbündel als inakzeptabel langsam. Dies liegt hauptsächlich am aufwändig zu erzeugenden selbstbeschreibenden Übertragungsformat und der relativ teuren dynamischen Typintrospektion für das Auslesen und Zurückschreiben der Daten. Um dies zu umgehen und die Kosten für Objekterzeugung und -freigabe nach oben abschätzen zu können, wird für harte Echtzeitbedingungen ein eigenes Serialisierungsverfahren unter Verwendung von Pufferobjekten auf Empfängerseite ohne Typintrospektion implementiert. Auch für den Austausch von Informationen zwischen Kontrollfäden innerhalb der Laufzeitumgebung eines Kommunikationsknotens werden nur primitive Typen und wiederverwendete Objektstrukturen eingesetzt.

Für die Garantie harter Echtzeitbedingungen werden getrennte Phasen für Initialisierung und Aufgabenausführung ("mission") [88] definiert. Nach der Initialisierungsphase sind alle Kontrollfäden für die Durchführung der folgenden Aktivitäten erzeugt.

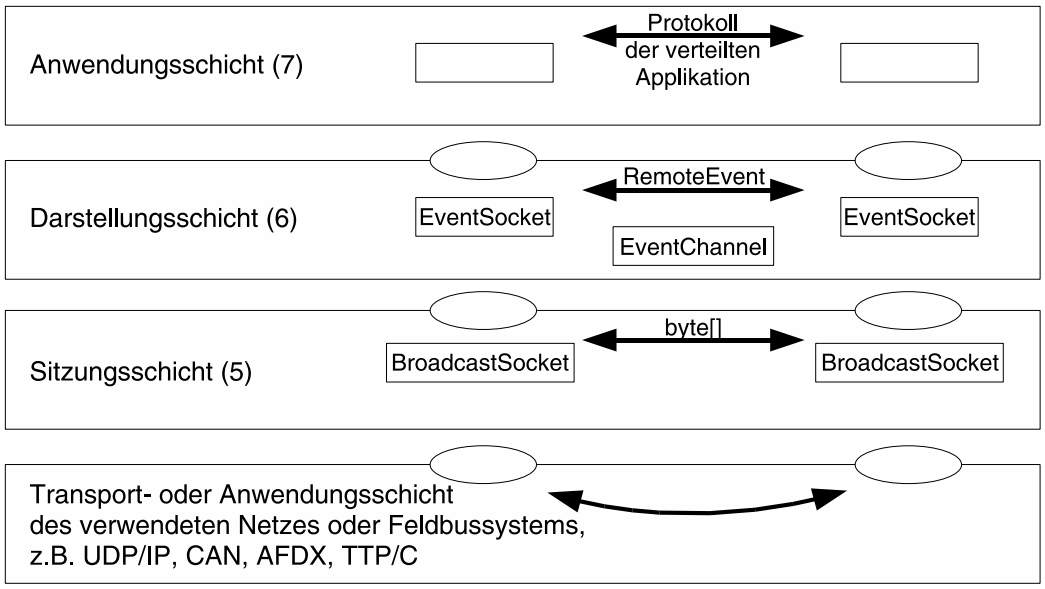

Abbildung 4.4: Schichtenmodell für Nachrichtenkommunikation

4.2 Implementierung der asynchronen Nachrichtenverarbeitung

Die Implementierung der vorgestellten Rahmenarchitektur verwendet nur Standard-Java-Klassen für den Zugriff auf das verwendete Kommunikationsnetz. Die Kommunikation erfolgt dabei über ein Netz, das Rundruf- oder Gruppenruflogik unterstützt. Beispiel für nicht-echtzeitfähige Netze, basierend auf einer Ethernet-Vernetzung, ist das Protokoll UDP/IP [86, 87] mit Gruppenruf ("multicast"). Als Treiberklasse zum Zugriff auf den Netzzugangspunkt wird `java.net.MulticastSocket` der Standard-Java-Programmbibliothek verwendet. Diese bietet für die Datenübertragung die Funktionalität der Transportschicht. Die Treiberklasse verwendet eine synchrone Form der Kommunikation: beim Empfang von Nachrichten wird blockiert. Für andere (echtzeitfähige) Kommunikationsnetze ist die Unterstützung durch entsprechende Treiberklassen notwendig. Die Abbildung 4.4 zeigt das Nachrichtenkanal-Netz als Schichtenmodell – analog zum ISO/OSI Schichtenmodell für Kommunikation. Die Kommunikation des Nachrichtendienstes wird in zwei Schichten organisiert. In der Sitzungsschicht erlaubt die Implementierung der Schnittstelle `BroadcastSocket` den Byte-orientierten Zugriff auf das unterliegende (Echtzeit-) Netz.

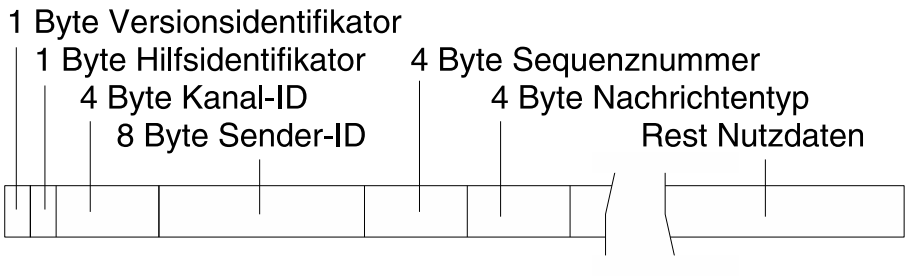

Abbildung 4.5: Serialisierung einer Nachricht

Die Darstellungsschicht kennt die Abstraktion von Nachrichten und Nachrichtenkanälen. Eine Instanz der Nachrichtensockelklasse (“`EventSocket`“) unterstützt die (De-)Kodierung von Nachrichten mit der Byte-Feld-Schnittstelle der Sitzungsschicht. Ein Objekt der Schnittstelle `BroadcastSocket` wird verwendet, um die Nutzdaten einer Nachricht zusammen mit Verwaltungsinformationen (wie z.B. Nachrichtenkanalbezeichner und laufender Nummer) zu übermitteln. Die Serialisierungsform einer Nachricht, wie sie in Abbildung 4.5 beschrieben wird, enthält dabei:

- Ein **Versionsindentifikator** (1 Byte) unterscheidet die verwendete Version des Nachrichtendienstes. Der Prototyp unterstützt dazu eine Version für harte, für weiche und eine ohne Echtzeitbedingungen; vergleiche dazu die einzelnen Unterabschnitte im Abschnitt 4.3.

- Mit einem **Hilfsidentifikator** (1 Byte) wird derzeit nur für harte Echtzeitbedingungen eine spezielle Form der Nachrichtenserialisierung unterschieden (vgl. Unterabschnitt 4.3.1).

- Die **Kanal-ID** (4 Byte) repräsentiert den verbundenen Nachrichtenkanals.

- Mit einem eindeutigen Bezeichner **Sender-ID** (8 Byte) wird der Sender der Nachricht identifiziert.

- Die **Sequenznummer** (4 Byte) ist dabei eine laufende Nummer für alle Nachricht dieses Senders.

- Das Feld **Nachrichtentyp** (4 Byte) wird beim Nachrichtenempfang verwendet um der Nachricht einen Typ innerhalb des Protokolls eines Nachrichtenkanals zuzuordnen.

- Der Rest der Nachrichtendaten enthält die serialisierten[4] **Nutzdaten** der Nachricht.

Nachrichten, die so über das Kommunikationsnetz übertragen werden, sind vom Typ `RemoteEvent`. Diese Klasse ist eine Abstraktion der konkreten verwendeten Java-Klassen. Der Quellcodeausschnitt 4.1 zeigt wesentliche Attribute.

- Mit einer Referenz auf ein Nachrichtenkanalobjekt wird die Zugehörigkeit der Nachricht zu einem Nachrichtenkanal geschrieben. Bei der Serialisierung als Byte-Feld und Übertragung über das Kommunikationsnetz wird diese Referenz als Integer-Zahlenwert kodiert.
- Zur Identifikation hat jeder Kommunikationsknoten in der verteilten Applikation einen eindeutigen Bezeichner. Dieser Long-Zahlenwert wird bei der Zuordnung der Nachrichten zu einem von mehreren Sendern eines Nachrichtenkanals verwendet.
- Zusammen mit der laufenden Nummer wird die Nachricht eindeutig identifiziert und eventuell notwendige zeitliche Reihenfolgen können bestimmt werden.
- Der Nachrichtentyp stellt neben den eigentlichen Nutzdaten die Information der Nachricht dar, mit der die verteilte Applikation Programmlogik steuern kann.
- Unterklassen von `RemoteEvent` unterschieden sich stark durch die Repräsentation der Nutzdaten. Mehr Informationen für die unterschiedlichen Implementierungen der Rahmenarchitektur mit harten und weichen Echtzeitbedingungen finden sich im Abschnitt 4.3.

[4] Für die unterschiedlichen Versionen des Nachrichtendienstes unterscheiden sich die bei der Serialisierung verwendeten Verfahren.

```
public abstract class RemoteEvent {

  /** Nachrichtenkanal*/
 protected EventChannel channel;

 /** Sender-ID */
 protected long transmitterID;

 /** Laufende Nummer */
 protected int seqNum;

 /** Nachrichtentyp */
 protected int type;

  // ...
```

Quellcode 4.1: Basisklasse für Nachrichten der Java-Implementierungen

Der Versand und Empfang dieser Nachrichten wird von Objekten erledigt, deren Klassen die Schnittstellen `PushEventTransmitter` und `PushEventReceiver` implementieren. Jeder Kommunikationsknoten besitzt für jedes unterstützte Kommunikationsnetz genau eine Empfängerkomponente. Sie werden im Kollektiv (vgl. Abschnitt 3.5.2) verwaltet. Die Implementierung erbt von einer Echtzeitkontrollfaden-Klasse und in der notwendigen `run()`-Methode wird die `receive()`-Methode des verwendeten `EventSocket`-Objekts aufgerufen. Solange keine Daten am Netzzugangspunkt verfügbar sind blockiert diese. Der Empfänger arbeitet mit der höchsten in RTSJ verfügbaren Priorität und garantiert so, dass Daten, sobald diese zur Verfügung stehen, auch empfangen werden können. Nach Rückkehr der blockierten `receive()`-Methode wird die empfangene Nachricht durch einen Empfängerkontrollfaden direkt in das prioritätssortierte Warteschlangensystem eingereiht und der Kontrollfaden der `ActivityManager`-Instanz benachrichtigt. Die Arbeit für den Empfängerkontrollfaden wird dabei auf die Datenentgegennahme und Kopierarbeit beschränkt. Über die schlechteste (d.h. größte) anzunehmende Nachrichtengröße kann der Aufwand dieser Arbeit nach oben abgeschätzt werden. Sofort danach beginnt der Kontrollfaden des Empfängers eine neue Schleife für den Empfang neuer Nachrichten. Solange er auf diese Daten warten muss, ist er blockiert und andere Kontrollfäden werden von der Laufzeitumgebung auf dem Prozessor ausgeführt.

Die `ActivityManager`-Komponente ist ebenfalls ein Objekt einer Unterklasse von `javax.realtime.RealtimeThread`. Seine Priorität ist um 1 geringer als die maximale verfügbare Priorität in RTSJ. Der Kontrollfaden wird mit sporadischer Wiederholung initialisiert. In der aktuellen Version der Echtzeiterweiterung RTSJ bedeutet dies, dass ein Echtzeitkontrollfaden erzeugt wird und durch Mechansimen der Inter-Kontrollfadenkommunikation seine Ausführung zu sporadischen Zeitpunkten aktiviert wird. Die Mindestzwischenzeit von zwei Aktivierungen des Kontrollfadens – notwendig für die Überprüfung eines Ablaufplans – errechnet sich als größter gemeinsamer Teiler (ggT) aus den Perioden der zu empfangenen Nachrichtenkanäle[5].

Mindestzwischenzeit: $ggT(600, 1200, 400) = 200$ Millisekunden zwischen zwei Aktivierungen.

Da das Kollektiv der Empfängerkomponenten als zentrale Verwaltungseinheit (für den Zugriff auf das Warteschlangensystem und die Registrierung von Behandlungslogiken) innerhalb des Kommunikationsknotens genutzt wird, erhält der `ActivityManager`-Kontrollfaden von dort seine Konfigurationsparameter. Dort werden Nachrichtenkanäle und deren Behandler (`PushEventHandler`-Objekte) registriert. `PushEventHandler`-Objekte beinhalten die Verarbeitungslogik für die empfangenen Nachrichten. Um diese Verarbeitungslogik zu verändern, unterstützt die Behandlerklasse eine Schablonenmethode `handleEvent()`, die durch Vererbung von der Klasse `PushEventHandler` überschrieben oder durch eine zusätzliche Logikklasse (Unterklasse von `AbstractAction`) von außen bereitgestellt werden kann. Wie für die Programmierung mit Kontrollfäden in Java üblich, wird die Logik in `AbstractAction` durch die Schnittstelle `java.lang.Runnable` in einer `run()`-Methode implementiert. `AbstractAction` und seine Unterklassen bieten zusätzlich eine Schnittstelle mit einer Fabrikmethode:

```
public static AbstractAction getAction(String[] params);
```

Durch Aufruf der Methode `fire()` der Klasse `PushEventHandler` wird eine empfangene Nachricht aus dem Warteschlangensystem in die Eingangswarteschlange des Behandlers verschoben. Der Manager für Aktivitäten (`ActivityManager`) ist dafür verantwortlich und verwaltet dafür vor-

[5] Verschiebungen durch unterschiedliche erste Eintreffzeitpunkte bei den Nachrichtenkanälen sind für diese Berechnung nicht relevant.

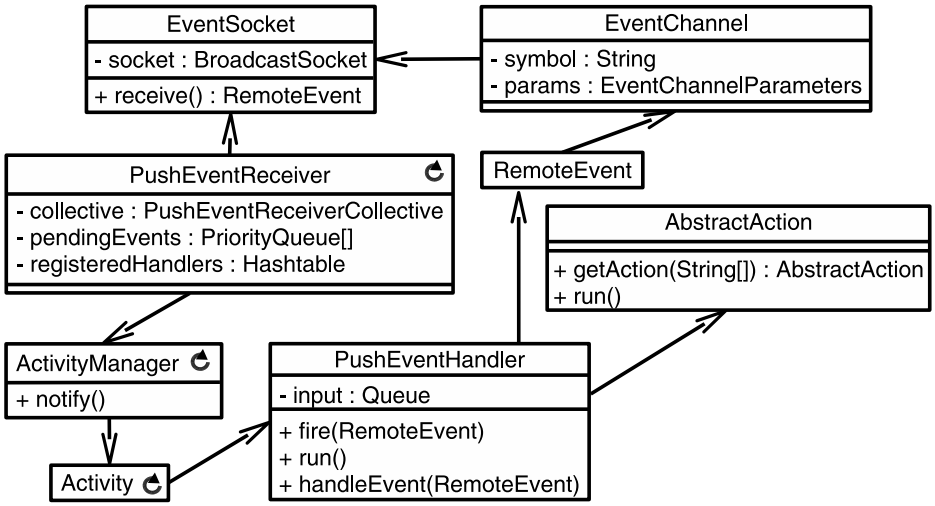

Abbildung 4.6: Implementierung des Nachrichtenkanal-Netzes mit Java-Klassen

gesehene Echtzeitkontrollfäden (sogenannte Aktivitäten, `Activity`-Objekte), die zur Ausführung mit passenden Prioritäten bereitliegen. Die `Activity`-Kontrollfäden werden bei der Initialisierung der Kommunikationsinfrastruktur erzeugt, gestartet und direkt schlafen gelegt. Der Aktivitätsmanager aktiviert sie zur Ausführung der registrierten Behandler und stellt ihre Fristen entsprechend ein.

Abbildung 4.6 illustriert die Beziehung zwischen den notwendigen Klassen in einem Klassenstrukturdiagramm. Das Diagramm vereinfacht die dargestellten Komponenten auf die notwendige Funktionalität beim Empfang von Nachrichten. Die Klasse `EventSocket` unterstützt natürlich auch das Senden mit `send(RemoteEvent)`. Abbildung 4.7 verdeutlicht den Fluss der zu verarbeitenden Nachrichten durch die beteiligten Komponenten der Rahmenarchitektur. Nach Eintreffen von Daten in der Netzzugangsschnittstelle werden diese durch einen zugehörigen Empfängerkontrollfaden (`Receiver`) über einen Kommunikationssockel mit `BroadcastSocket`- und `EventSocket`-Objekten gelesen und als Nachrichten für die weitere Verarbeitung in einer prioritätssortierten Warteschlange zwischengespeichert.

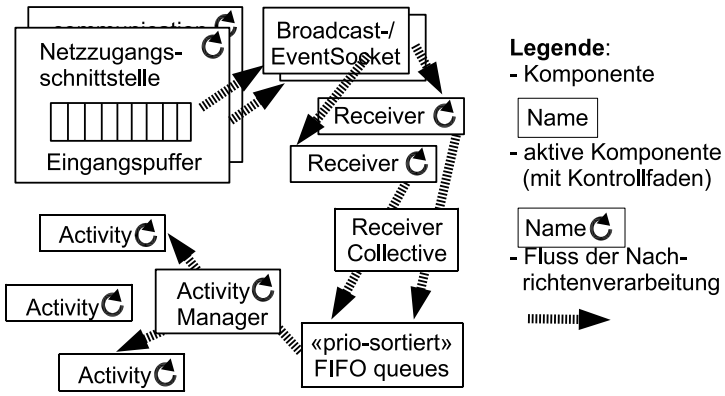

Abbildung 4.7: Fluss der empfangenen Nachrichten durch die Komponenten der Rahmenarchitektur

4.3 Besonderheiten durch harte und weiche Echtzeitbedingungen

Das Nachrichtenaustauschformat kann durch die Serialisierung (vgl. Abbildung 4.5 auf Seite 85) als einfache Byte-Folge zwar plattformübergreifend interpretiert werden, aber die Java-Implementierung der Rahmenarchitektur serialisiert Nachrichten mittels Identifikatoren für unterschiedliche Ausprägungen. Die Festlegung auf eine Version muss durch den Entwickler für alle Kommunikationsknoten einer verteilten Applikation einheitlich erfolgen.

Einheitlich für jede Java-Version des Nachrichtenkanal-Netzes ist die Erzeugung eines Einzelstücks der Klasse `EventChannelNetwork` zu Beginn. Durch die Festlegung auf ein Echtzeitprofil (vgl. die Entwicklung im Rahmen des EU-Forschungsprojektes HIJA, Abschnitt 5.1) wird die Erzeugung weiterer Komponenten konfiguriert. Das jeweilige `EventChannelNetwork`-Objekt stellt – analog zum Entwurfsmuster einer abstrakten Fabrik – den zentralen Zugangspunkt zu allen anderen Komponenten der Rahmenarchitektur dar. Abbildung 4.8 zeigt die Klasse `EventChannelNetwork` mit ihren Beziehungen zu anderen Komponenten[6].

Erforderliche Echtzeitbedingungen, ob harte oder weiche, ergeben unterschiedliche Anforderungen an die Implementierung der Rahmenarchitektur des Nachrichtenkanal-Netzes. Wie in Abbildung 3.7 auf Seite 67 beschrieben, existieren zwei Versionen, eine für harte und eine für weiche

[6] Für unterschiedliche Versionen existieren spezielle Implementierungen dieser Komponenten, die in den folgenden Abschnitten beschrieben werden.

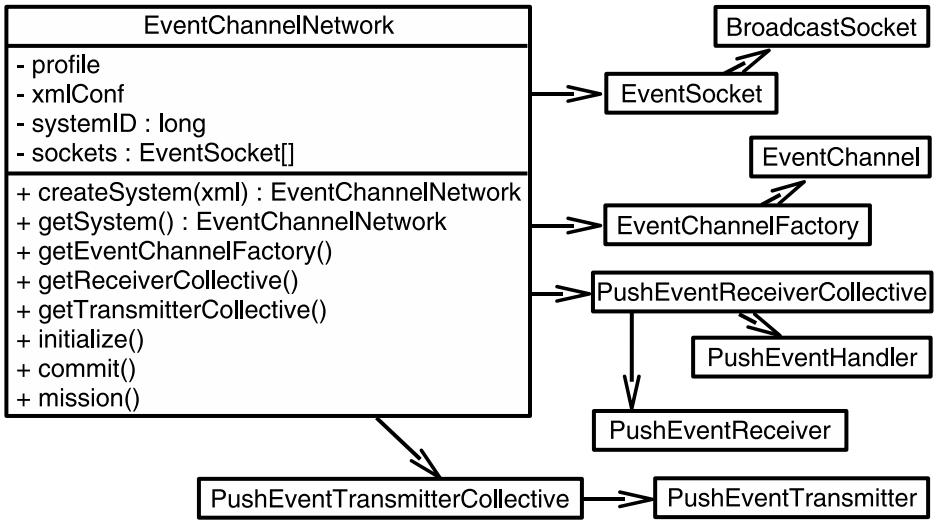

Abbildung 4.8: Abstrakte Fabrik des Nachrichtenkanal-Netzes

Echtzeitbedingungen. Beide Versionen folgen den in Kapitel 3 vorgestellten Entwurfsentscheidungen, orientieren die Implementierung aber an Anforderungen des jeweiligen Einsatzgebietes. Eine dritte hier vorgestellte Implementierung ist nicht alleinstehend gedacht. Sie basiert auf Standard-Java und kann deshalb keine Echtzeitbedingungen unterstützen. Ihre Entwicklung ist von der Notwendigkeit motiviert, auch mit der herkömmlichen Java-Laufzeitumgebung (JVM) und dort verfügbaren Bibliotheken (z.B. zur Programmierung von grafischen Benutzeroberflächen mit Swing oder AWT) auf den Nachrichtendienst zugreifen zu können.

Nachrichtenkanal-Netz HRT ("hard real-time") für harte Echtzeitbedingungen erlaubt die Nachrichtenkommunikation in statisch vorhersagbaren Zeitgrenzen. Die Implementierung verwendet nur Echtzeitkontrollfäden von RTSJ außerhalb der Halde. Für die verwendeten Nachrichten wird ein eigenes Verfahren der Serialisierung implementiert und für die maximale Anzahl der gleichzeitig zu verarbeitenden Nachrichtenobjekte werden ausreichend Pufferobjekte in der Initialisierungsphase angelegt. Auch für die Interaktion zwischen Echtzeitkontrollfäden (z.B. Empfänger, Aktivitätsmanager und Aktivität) werden entsprechende Pufferobjekte verwendet.

Nachrichtenkanal-Netz FRT ("flexible real-time") für weiche Echtzeitbedingungen unterstützt ein dynamisches Verwaltungsprotokoll. Echtzeitbedingungen werden ähnlich der Bedingungen in der HRT-Implementierung gefordert. Bei einer Nichteinhaltung von Zeitfristen

durch die beteiligten Echtzeitkontrollfäden werden aber die dynamischen Möglichkeiten von RTSJ für die Neuberechnung von Ablaufplänen der Ablaufkoordinierung verwendet und das Gesamtsystem kann weiterarbeiten. Prioritäten bei der Verarbeitung werden vom Entwickler definiert und die Nichteinhaltung von Verarbeitungsfristen muss dieser bei Bedarf durch die Behandlung von Ausnahmen beachten. Aufbauend auf das dynamische Verwaltungsprotokoll der FRT-Version wird es möglich, dynamisch Kommunikationsknoten und neue Nachrichtenkanäle in eine verteilte Applikation zu integrieren.

Nachrichtenkanal-Netz JVM ("Java Virtual Maschine") erlaubt die Interaktion mit Kommunikationsknoten, die als Laufzeitumgebung Standard-Java verwenden. Komponenten auf diesen Knoten können zwar keine Echtzeitbedingungen erfüllen, aber das Byte-orientierte Modell für die Nachrichtenserialisierung erlaubt, dass Nachrichten von HRT- oder FRT-Nachrichtenkanälen empfangen werden können. Die Verarbeitung in der JVM-Komponente der Applikation darf nicht zeitkritisch sein (z.B. Visualisierung einer Messgröße für den Benutzer) und Echtzeit-Komponenten dürfen ihrerseits nicht von dieser Verarbeitung abhängen. Wenn diese Bedingungen erfüllt sind, vereinfacht die JVM-Version der Rahmenarchitektur die Entwicklung verteilter Applikationen und Integration mit bestehenden Java-Applikationen.

4.3.1 Nachrichtenkanal-Netz HRT

Für die Beschreibung der HRT-Implementierung des Nachrichtenkanal-Netzes wird in diesem Abschnitt wieder das in Abschnitt 1.2 vorgestellte Beispielszenario eines Produktionssystems mit drei Sender- und einem Empfängerknoten verwendet.

Weil die Initialisierungsphase von Systemen mit harten Echtzeitanforderungen noch nicht den gleichen (periodischen) harten Zeitfristen wie in der Ausführungsphase unterliegt, werden hier alle notwendigen Kontrollfäden, Warteschlangen und Datenpufferobjekte erzeugt. Die Erzeugung von Objekten in der Ausführungsphase soll weitgehend vermieden werden. Für den Empfang der Nachrichten und die lokale Kommunikation zwischen Kontrollfäden (Empfänger, Manager, Aktivitäten) werden Objekte im dauerhaften Speicherbereich (`ImmortalMemory`) wiederverwendet. Für die periodische Verarbeitung der eingetroffenen Nachrichten werden

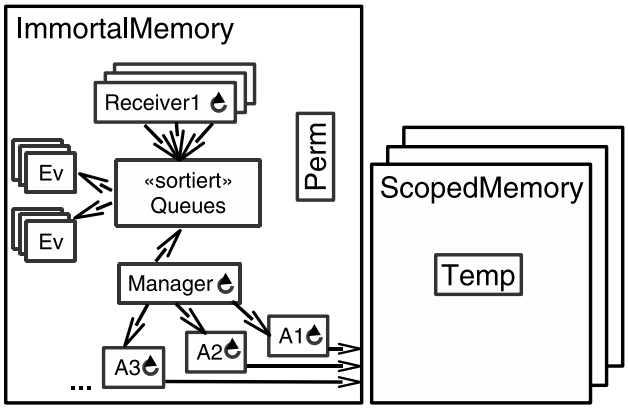

Abbildung 4.9: Speicherbereiche und Objekte in der HRT-Version

`ScopedMemory`-Bereiche verwendet. Die Laufzeitumgebung sorgt für die periodische Aktivierung und die vollständige Bereinigung nach Ende der Arbeit. Abbildung 4.9 illustriert die konzeptionelle Verwendung der Speicherbereiche und zeigt die Aufteilung der notwendigen Objekte. Die Echtzeitkontrollfäden für Empfang (`Receiver1`, ...) der Nachrichten sowie Ausführung (`A1-A3`) und Verwaltung (`Manager`) der Verarbeitungslogik werden als dauerhafte Objekte im `ImmortalMemory`-Bereich erzeugt. Für die Zwischenspeicherung der empfangenen Nachrichten werden leere Puffer-Objekte (`Ev`) im dauerhaften Speicherbereich alloziert. Notwendige permanente Objekte (`Perm`) liegen ebenfalls hier, im Gegensatz zu temporären Objekten (`Temp`), die bei der Verarbeitung im `ScopedMemory`-Speicher der verwendeten Aktivitätskontrollfäden angelegt werden.

Als weitere besondere Einschränkung der HRT-Version wird bei der Kommunikation mit Nachrichten nur die Form von `PrimitiveRemoteEvent`-Objekten, einer Unterklasse von `RemoteEvent` (vgl. Quellcodeausschnitt 5.1), unterstützt. `PrimitiveRemoteEvent` und spezielle Unterklassen erlauben den Versand von Nutzdaten, die sich in Werttypen zerlegen lassen. Für den Versand von Objekten bedeutet dies, dass ein Verfahren zur Serialisierung der Objektattribute notwendig ist. Hierfür ist die Schnittstelle `Copyable` definiert. Klassen, die diese Schnittstelle implementieren, unterstützen das Kopieren der eigenen Attributwerte in ein zweites Objekt gleichen Typs. Der Quellcodeausschnitt 4.2 verdeutlicht dies am Beispiel der Klasse `PrimitiveRemoteEvent`. Die Nutzdaten (`value`) sind nicht vom allgemeinen Typ `Object`, sondern implementieren die `Copyable`-Schnittstelle.

Der in Abschnitt 4.2 beschriebene Zugriff auf die Netzzugangsschnittstel-

le wird für den Einsatz einer besser planbaren Nachrichtenserialisierung etwas erweitert. Bei der Deserialisierung wird zwischen Nachrichtentypen unterschiedlicher Nachrichtenkanäle unterschieden. Anders als der direkte Zugriff von `EventSocket` auf `BroadcastSocket` (vgl. Abbildung 4.4 auf Seite 84) wird ein Manager für die Benutzung des `BroadcastSocket`-Objekts (eines Kommunikationsnetzes) mit mehreren `WrapperSocket`-Instanzen verwendet. Die Klasse `StaticEventSocket` (Unterklasse der Klasse `EventSocket`) fungiert als `ManagerSocket` und verwaltet unterschiedliche Dekorierer (`WrapperSocket`) für den Zugriff auf den Kommunikationssockel. Abbildungen 4.10 und 4.11 illustrieren die Beziehung von Klassen und Schnittstellen für das Beispielszenario. Da von den drei Senderknoten unterschiedliche Nachrichten über ein einziges Rundrufnetz verschickt werden, muss der Empfänger diese wieder trennen und den jeweiligen Nachrichtenkanälen zuordnen. Die Einbindung unterschiedlicher Dekorierer (`WrapperSocket`, vgl. Abbildung 4.10) im Kommunikationssockel werden dabei verwendet, um die nutzdatenabhängige Serialisierung sicherzustellen. Abbildung 4.11 zeigt passend diese Nachrichten und Nutzdaten, die das Kopieren von Attributen (`Copyable`) unterstützen. Die Klasse `CopyableInt` bietet einem Integer-Zahlenwert und wird zum Beispiel für die Nachrichten des Temperatursensors verwendet. Die Klasse `Data` ist ein Beispiel für eine komplexere Datenstruktur und repräsentiert drei Zahlenwerte für Höhe/Breite/Länge der Stichprobe bei der Funktionsanalyse. Beide Klassen implementieren neben der Schnittstelle Copyable für das Kopieren der eigenen Attributwerte auch die Schnittstelle `StaticExternalizable`. Angelehnt an `java.io.Externalizable` wird eine Schnittstelle für die applikationsspezifische Serialisierung definiert. Von einer implementierenden Klasse müssen die Methoden `read-` und `writeExternal()` angeboten werden. Beide Methoden haben als Parameter eine Byte-Feld das Quelle bzw. Ziel und einen Indexwert der den Beginn der Serialisierung im Feld beschreibt. Für das verwendete Serialisierungsverfahren lässt sich abhängig von der Struktur und Größe eines zu serialisierenden Objekts der Zeitaufwand abschätzen. In der vorgestellten Kommunikationsrahmenarchitektur wird dies ausgenutzt, um mit der größten anzunehmenden Länge einer Nachricht die Zeit für die Serialisierungsarbeit nach oben abzuschätzen. Beispiele für die Serialisierung von unterschiedlichen Nachrichten sind mit der Implementierung der Rahmenarchitektur verfügbar.

Außer für den Nachrichtenversand und -empfang wird der Mechanismus des Kopierens von Attributwerten auch bei der Interaktion von Kontrollfäden mittels Zugriff auf gemeinsame Speicherbereiche verwendet. In der

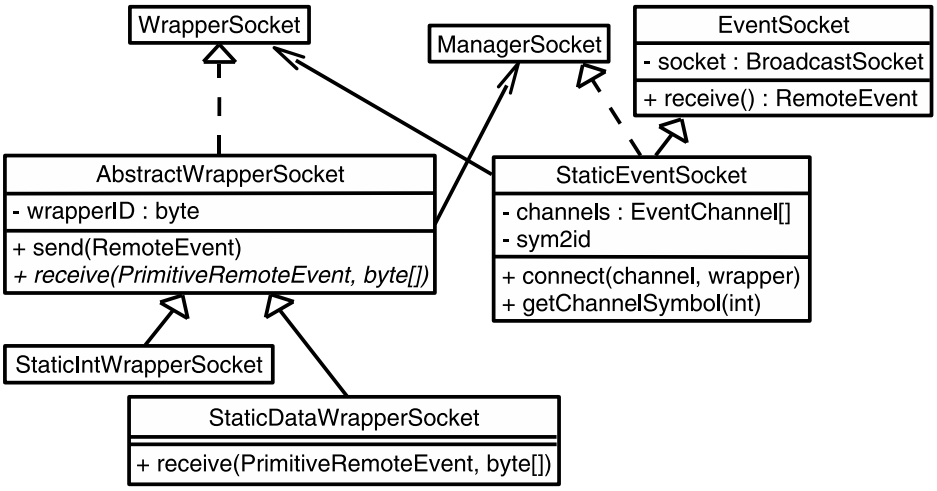

Abbildung 4.10: Klassen für Kommunikationssockel des Beispielszenarios in der HRT-Version

HRT-Version der Implementierung werden deshalb im `ImmortalMemory`-Speicherbereich Pufferobjekte für alle notwendigen Warteschlangen erzeugt. Dies sind Nachrichten-Objekte für die empfangenen Nachrichten der Nachrichtenkanäle und Pufferobjekte bei der Verwaltung von Aktivitätskontrollfäden.

Da diese Parameter einer Applikation mit harten Echtzeitbedingungen statisch vor der Ausführung bekannt sind, ist die Erzeugung aller notwendigen Objekte, zusammen mit den Objekten der Kontrollfäden in der Initialisierungsphase der Applikation möglich.

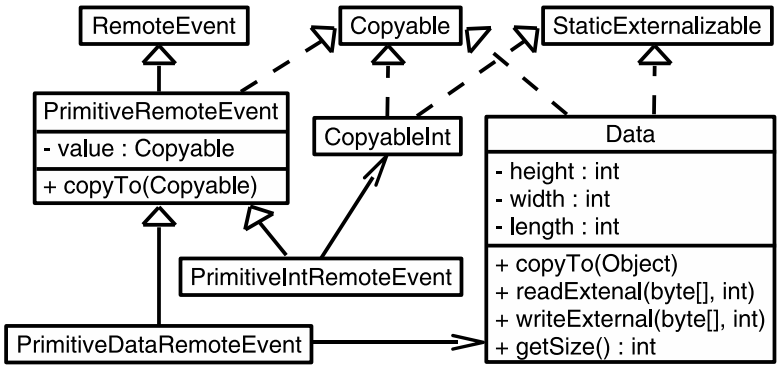

Abbildung 4.11: Klassen für Nachrichten des Beispielszenarios in der HRT-Version

```
public class PrimitiveRemoteEvent extends RemoteEvent
                                  implements Copyable {
  /** Nutzdaten der Nachricht */
  protected Copyable value;

  /** Zeitstempel beim Eintreffen der Nachricht*/
  protected AbsoluteTime time;

  /** Interface Copyable */
  public void copyTo(Copyable target) {
    if (target instanceof PrimitiveRemoteEvent) {
      PrimitiveRemoteEvent event =
        (PrimitiveRemoteEvent) target;
      event.channel = channel;
      event.transmitterID = transmitterID;
      event.seqNum = seqNum;
      event.type = type;
      if (value != null && event != null) {
        value.copyTo(event.value);
      }
    }
  }

  // ...
```

Quellcode 4.2: Basisklasse für Nachrichten der HRT-Version der Java-Implementierung

Für die Aktivitätskontrollfäden gibt es in der HRT-Implementierung eine weitere Besonderheit. Sie sind analog zu periodischen Nachrichtenkontrollfäden[7] als periodische Echtzeitkontrollfäden implementiert. Der Kontrollfaden des Aktivitätsmanagers regelt die Zuteilung der empfangenen Nachrichten und ist auch für die Synchronisation der erforderlichen Abarbeitungsfrist mit dem asynchronen Empfangszeitpunkt verantwortlich. Der Quellcodeausschnitt 4.2 der Nachrichtenklasse zeigt hierfür das Attribut eines Zeitstempels, den der Empfängerkontrollfaden bei Empfang der Nachricht hier einträgt.

[7]Die HRT-Version der Rahmenarchitektur mit Echtzeit-Java unterstützt nur periodische Nachrichtenkanäle, um die statische Analyse des Ablaufplans der Echtzeitkontrollfäden sicherzustellen.

Das Beispiel zeigt einen Ausschnitt ohne andere nebenläufige Empfangs- und Verarbeitungsaktivitäten. Sollte zum Beispiel bei Schritt 5 eine weitere Nachricht im Eingangspuffer der Netzzugangsschnittstelle verfügbar sein, dann wiederholen sich direkt die Schritte 3 und 4. Da der Empfängerkontrollfaden mit höchster Priorität arbeitet, können die Schritte 2-4 die Ausführung aller anderen Kontrollfäden jederzeit verzögern.

Um die Einhaltung aller Verarbeitungsfristen der nebenläufig eintreffenden Nachrichten zu ermöglichen, wird die freie Prozessorkapazität – zwischen Empfang durch den Empfängerkontrollfaden mit höchster Priorität und Zuteilung an Aktivitätskontrollfäden durch den Kontrollfaden der Aktivitätsmanagerkomponente mit zweithöchster Priorität – bestmöglich an alle Aktivitätskontrollfäden verteilt. Wie in Abschnitt 3.4.1 bei der Einführung von Ablaufkoordinierungsverfahren beschrieben, heißt bestmöglich dabei nicht mit voller Auslastung des Prozessors. Sichergestellt werden kann nur, dass bei erfolgreicher statischer Prüfung des Ablaufplans mit einem Werkzeug (z.B. MAST, vgl. Abschnitt 4.5) alle Kontrollfäden die Zeitfristen der Nachrichtenkanäle einhalten. Monoton nach Länge der Verarbeitungsfrist[8] wird den Aktivitätskontrollfäden dafür eine Priorität zugeordnet. Konkret für das Beispielszenario bedeutet das eine Zuordnung wie in Tabelle 4.1 angegeben. Die maximale bzw. minimale Priorität für Echtzeitkontrollfäden in der RTSJ-Implementierung lässt sich dabei wie folgt ermitteln:

`PriorityScheduler.instance().get[Max|Min]Priority();`[9]

Der Einsatz des `PriorityScheduler`-Verfahrens für die Ablaufkoordinierung regelt die Ausführung aller Echtzeitkontrollfäden. Die Nichteinhaltung der Verarbeitungsfrist für Nachrichten eines Nachrichtenkanals wird durch die Laufzeitumgebung erkannt. Wenn ein Aktivitätskontrollfaden seine Verarbeitungsfrist überschreitet, wird ein von der Kommunikationsinfrastruktur registrierter Behandlerkontrollfaden ausgeführt. Die `handleAsyncEvent()`-Methode wird ausgeführt und die Fristüberschreitung führt zu einem Ausführungsfehler des Kommunikationsknotens.

Eine genauere Beschreibung dieses Verhaltens und der notwendigen statischen Analyse zur Vermeidung von fehlerhaften Applikationen bietet der Abschnitt 4.5.

[8] Kürzere Verarbeitungsfrist bedeutet höhere Priorität, aber geringer als die des Aktivitätsmanagerkontrollfadens.

[9] In Tabelle 4.1 werden die tatsächlichen Werte 38 und 11 verwendet.

Kontrollfaden für ...	Frist	Priorität
... Empfänger		38
... Aktivitätsmanager		37
... Aktivität von K3	200	32*
... Aktivität von K1	300	27*
... Aktivität von K2	1200	22*

*Diese Prioritäten sind frei gewählt und erstrecken sich über den verfügbaren Bereich von Prioritäten für Echtzeitkontrollfäden. Dynamische Systeme mit weichen Echtzeitbedingungen können so leichter nachträglich zusätzliche Kontrollfäden mit mittleren Prioritäten hinzuzufügen.

Tabelle 4.1: Prioritäten der Echtzeitkontrollfäden in der Beispielanwendung

```
public abstract class FlexibleAbstractAction
                      extends AbstractAction {

  /** Kontrollfaden der Aktivität ausführt */
  protected FlexibleActivityManager.Activity activity;

  /** Behandlungslogik */
  public abstract void handleDeadlineMiss();
}
```

Quellcode 4.3: Basisklasse für Logik in FRT-Version

4.3.2 Nachrichtenkanal-Netz FRT

Die FRT-Version der Java-Implementierung basiert ebenfalls auf Echtzeit-Java. Da im Forschungsprojekt HIJA (vgl. Unterabschnitt 5.1) zwei Profile für die Entwicklung sicherheits- und geschäftskritischer verteilter eingebetteter Systeme ermittelt wurden, existieren auch zwei Implementierungen der vorgestellten Rahmenarchitektur. Anders als das HRTJ-Profil ("Hard Real-Time Java"), das Grundlage für die HRT-Version der Rahmenarchitekturimplementierung ist, werden für Systeme mit vorwiegend weichen Echtzeitanforderungen andere Voraussetzungen identifiziert.

Ein wesentlicher Unterschied zwischen beiden Versionen ist die Behandlung der Ressource Speicher. Das im Rahmen von HIJA definierte FSRTJ-Profil ("Flexible Soft Real-Time Java") unterstützt ein Verfahren der automatischen Speicherbereinigung unter Echtzeitbedingungen. Die hier vor-

gestellte Implementierung nutzt diese Eigenschaft aus und die Verwaltung von Nachrichten und anderen Objekten bei der Interaktion zwischen Kontrollfäden unterliegt nicht mehr den Einschränkungen, wie sie im letzten Abschnitt beschrieben wurden. Die FRT-Version verwendet den Speicherbereich der Halde um neue Objekte dynamisch mit dem `new`-Schlüsselwort zu erzeugen und überlässt die Speicherfreigabe der Laufzeitumgebung.

Außer dieser "internen" Veränderungen ist das Nachrichtenkanal-Netz FRT jetzt auch für Nutzung mit Standardkommunikationsnetzen ohne Unterstützung von Echtzeitbedingungen gedacht. Der Einsatz von zum Beispiel UDP/IP mit Gruppenruf-Kommunikation über Ethernet-Verkabelung eignet sich für die Entwicklung verteilter Systeme. Wenn das Nachrichtenaufkommen solcher verteilter Systeme relativ gering ist, kann auch ein Netzzugriff mit CSMA/CD-Algorithmus ("Carrier Sense Multiple Access with Collision Detection") einen hohen Datendurchsatz unter Einhaltung von Zeitfristen erreichen. Das Nachrichtenkanal-Netz FRT stellt dies zwar nicht sicher, aber Nachrichtendienst und Methodik unterstützen die Softwareentwicklung verteilter Systeme im Allgemeinen. Im Fehlerfall (d.h. bei Nichterfüllung einer Frist) muss der Entwickler Vorkehrungen in der Applikationsschicht treffen. Dazu zeigt der Quellcodeausschnitt 4.3 eine Spezialisierung der Klasse `AbstractAction`, die zusätzlich zur `run()`-Methode von `AbstractAction` noch Logik zur Fehlerbehandlung (`handleDeadlineMiss()`) enthält. Ein Feld `activity` enthält den Verweis auf den ausführenden Kontrollfaden der Aktion und seine Ausführung kann dynamisch gesteuert werden.

Der große Vorteil dieser Implementierung der Rahmenarchitektur ist deshalb auch nicht die Garantie von Verarbeitungsfristen, sondern die flexible Beschreibung einer verteilten Applikation mit Publiziere-/Abonniere-Kommunikation zwischen Sender- und Empfängerknoten. Durch Festlegung und Beschreibung von Nachrichtenkanälen und Aktionen erlaubt die Methodik die einfache Entwicklung verteilter Applikationen. Die verwendete XML-Beschreibung wird für alle Java-Implementierungen gemeinsam in Abschnitt 4.4 eingeführt.

Eine weitere Besonderheit der FRT-Version ist die Unterstützung eines dynamischen Verwaltungsprotokolls. Bei der Erzeugung von neuen `EventChannel`-Objekten für die Beschreibung von Nachrichtenkanälen wird nicht nur auf statische Beschreibungen in einer XML-Datei zurückgegriffen. Eine spezielle Implementierung der Fabrik zur Erzeugung von Nachrichtenkanal-Objekten (`AdminChannelEventChannelFactory` als Unterklasse von `EventChannelFactory`) unterstützt ein Protokoll dynamischer Erzeugung,

Ankündigung und Suche von Nachrichtenkanälen. Diese Möglichkeiten werden zum Beispiel auch im Rahmen der Evaluierung der Kommunikationsinfrastruktur (vgl. 5.2) mit der Implementierung einer Gleiche-zu-Gleichen-Kommunikationsinfrastruktur ("Peer-to-Peer") verwendet [65]. Das `AdminChannel`-Protokoll ist Grundlage für die Implementierung eines Überlagerungsnetzes ("overlay network"), das den Industriestandard Pastry [6, 4] auf Basis von Rundrufnetzen unterstützt.

4.3.3 Nachrichtenkanal-Netz JVM

Die Standard-Java-Version des Nachrichtenkanal-Netzes ist nicht alleinstehend gedacht. Diese Implementierung der Rahmenarchitektur unterstützt keine Echtzeitbedingungen, aber durch den Empfang von Nachrichten der HRT- und FRT-Version ist die Integration von Empfängerknoten mit der Nutzung von Bibliotheken auf Basis von Standard-Java möglich. Die Serialisierungsverfahren für HRT- und FRT-Nachrichten werden beide unterstützt und Nachrichten können dekodiert werden. Da die Laufzeitumgebung mit Standard-Java aber keine Echtzeitkontrollfäden unterstützt, können die Verarbeitungsfristen, die mit einem Nachrichtenkanal verbunden sind, nicht garantiert werden. Eventuell ist es nicht einmal möglich, alle Nachrichten eines Kanals fristgerecht entgegen zu nehmen. Die Nutzung von umfangreichen Java-Bibliotheken zum Beispiel für die grafische Darstellung der Nachrichtenkommunikation ist aber als Schnittstelle zum Benutzer gut geeignet.

4.3.4 Zusammenfassung

Die in Kapitel 3 vorgestellte Rahmenarchitektur für asynchrone Nachrichtenkommunikation in verteilten Systemen eignet sich für die Implementierung von verteilten Applikationen unterschiedlicher Einsatzszenarien. Durch die Bereitstellung von zwei echtzeitunterstützenden Java-Implementierungen wird die Umsetzung der vorgestellten Software-Entwurfsmuster demonstriert. Die HRT-Version ist Grundlage für die Entwicklung sicherheits- und geschäftskritischer verteilter Applikationen mit harten Echtzeitbedingungen. Diese Implementierung nutzt die Möglichkeiten der RTSJ aus und die verteilte Applikation kann mittels statischer Analyse (vgl. Abschnitt 4.5) auf die Einhaltung von Echtzeitbedingungen hin untersucht werden. Bei Implementierung des Nachrichtendienstes der Rahmenarchitektur für weiche Echtzeitanforderungen werden zukünftige Wei-

terentwicklungen von Profilen in der Spezifikation zu Echtzeit-Java, wie sie im Forschungsprojekt HIJA ermittelt wurden, schon eingesetzt. Echtzeitfähige Speicherbereinigung der Laufzeitumgebung vereinfacht die Implementierung und das Ergebnis ist ein Nachrichtendienst mit einem dynamischen Verwaltungsprotokoll, das die statische Beschreibung (vgl. Abschnitt 4.4) von Kommunikationskanälen erweitert. Durch die Implementierung der Rahmenarchitektur für eine Laufzeitumgebung mit Standard-Java zeigt sich die Plattformunabhängigkeit der dargestellten Softwareentwurfsmuster. Weil die Unterstützung von Echtzeitbedingungen hier nicht möglich ist, wird nur die Entwicklung heterogener verteilter Systeme mit Kommunikationsknoten auch ohne Echtzeit-Java deutlich.

Die in den nächsten Abschnitten vorgestellte XML-Beschreibung verteilter Systeme mit Nachrichtenkanälen, die Beschreibung von Echtzeitbedingungen und die mögliche statische Analyse vervollständigt die durchgängige Methodik. Das für die Entwicklung von verteilten Systemen unter Echtzeitbedingungen einfach verständliche Beschreibungsverfahren unterstützt diese und verdeutlicht den Nutzen eines Nachrichtendienstes. Die vorgestellten Implementierungen basieren zwar auf Java und Echtzeit-Java, aber der Transfer auf andere objektorientierte Laufzeitumgebungen ist möglich.

4.4 XML-Beschreibung verteilter Systeme

Den im letzten Abschnitt vorgestellten Java-Implementierungen der Rahmenarchitektur für asynchrone Nachrichtenkommunikation in verteilten Systemen fehlt noch die angestrebte Integration in einen durchgängigen Softwareentwicklungsprozesses. Grundlage für diese Entwicklungsmethodik ist eine Beschreibung der eingeführten Nachrichtenkanäle mit Sendern und Empfängern. Die hierfür in Abschnitt 3.6 beschriebene Methode zur Systembeschreibung wird in den Prototypen mit XML implementiert. Außer der statischen Systemkonfiguration werden in einer XML-Datei Echtzeitbedingungen und Zeitzusicherungen der referenzierten Java-Verarbeitungslogik beschrieben. Diese Fristen sind Voraussetzung für die Generierung eines Analysemodells, das generierten Programmcode und Komponenten der Rahmenarchitektur repräsentiert.

Dieser Abschnitt führt die wesentlichen Einheiten der XML-Datei ein. Eine DTD für das verwendete XML-Dateiformat der Version 2.1 des Nachrichtenkanal-Netzes zeigt Anhang B dieser Ausarbeitung. Die Beschreibung erfolgt für jeden Knoten in einer eigenen XML-Datei. Außer den für

XML notwendigen Informationen enthält jede dieser Dateien das Wurzelelement <node>. Quellcodeausschnitt 4.4 zeigt den 1. Teil einer Beschreibung für einen Kommunikationsknoten, der alle Sensorknoten und die Steuereinheit des Beispielszenarios aus Abschnitt 1.2 in einem Kommunikationsknoten vereinigt. Dieser Knoten heißt AllInOne und verwendet HRTJ als Profil der RTSJ-Implementierung[10].

In Quellcodeausschnitt 4.5 wird der 2. Teil der Beschreibungsdatei eines Kommunikationsknotens gezeigt. Zur Anbindung eines Knotens an zugrundeliegende Kommunikationsnetze werden <socket>-Elemente verwendet. Da das Beispielszenario auf harte Echtzeitbedingungen ausgelegt ist, werden die in Abschnitt 4.2 eingeführten Dekorierer des Kommunikationssockels ebenfalls hier beschrieben.

Zur Vereinfachung werden nur zwei <wrapper>-Elemente für die Nachrichtenkanäle der Temperatursensor- und Funktionsprüfungskomponenten im Detail dargestellt. Das <socket>-Element erhält in Zeile 7 einen Namen für die Referenzierung innerhalb der XML-Datei und der Name der verwendeten Treiberklasse wird als code-Attribut beschrieben. Für die Erzeugung des Treiberklassenobjekts ist eine optionale Liste von Attributwerten gedacht (vgl. Zeile 8-9). Für die Interpretation dieser Zeichenketten ist die Erzeugungsfabrikmethode der Treiberklasse (Unterklasse von BroadcastSocket) verantwortlich.

```
001: <?xml version="1.0" encoding="UTF-8"?>
002: <!DOCTYPE node SYSTEM
      "http://www.eventchannelnetwork.org/ecn2.dtd">
003: <node name="AllInOne" id="1">
004:  <rtsj profile="HRTJ">
005:   <factory>
        de.fzi.ecn.hrt.StaticEventChannelFactory
       </factory>
006:  </rtsj>
```

Quellcode 4.4: XML-Beschreibung der Steuerung eines Produktionssystems (1-Knoten-Version, Teil 1 von 5)

[10]Die verwendete Java-Laufzeitumgebung ist die JamaicaVM der Aicas GmbH [8]. Das in HIJA entwickelte HRTJ-Profil für sicherheits- und geschäftskritische Systeme mit harten Echtzeitanforderungen bestimmt die Implementierung der Rahmenarchitektur.

```
007: <socket name="Socket" code="BroadcastSocketImpl">
008:  <attributes>
009:   <value>14474</value>
010:  </attributes>
011:  <buffer>12288</buffer>
012:  <wrapper name="TemperatureSocket" id="1">
013:   <class>
        de.fzi.ecn.hrt.StaticIntWrapperSocket
       </class>
014:   <channelref
        ref="AllInOne.Server.TemperatureChannel"
        id="1" />
015:  </wrapper>
016:  <wrapper name="CheckSocket" id="2">
017:   <class>
        de.fzi.ecn.test.control.StaticDataWrapperSocket
       </class>
018:   <channelref
        ref="AllInOne.Server.CheckChannel"
        id="2" />
019:  </wrapper>
020:  <wrapper> ... </wrapper>
024: </socket>
```

Quellcode 4.5: XML-Beschreibung der Steuerung eines Produktionssystems (1-Knoten-Version, Teil 2 von 5)

Zur Referenzierung erhalten auch die Dekorierer des Kommunikationssockels (Zeilen 12 u. 16) eindeutige Namen. Ebenfalls notwendig ist die Deklaration von Zahlenbezeichnern, die als Hilfsidentifikatoren die Zuordnung der Nachrichten zu einem Dekorierer (mit speziellen Serialisierungsmethode) ermöglichen. Die verwendeten Klassen der Dekorierer werden im jeweiligen `<class>`-Element beschrieben. Mit dem XML-Element `<channelref/>` werden die weiter unten folgenden Deklarationen der verwendeten Nachrichtenkanäle referenziert. Die ebenfalls in den Zeilen 14 und 18 angegebenen Bezeichner kodieren diese Nachrichtenkanäle über den Kommunikationssockel und müssen für diesen eindeutig sein.

Die detaillierte Beschreibung der Nachrichtenkanäle ist Teil der Beschreibung des Senderknotens im Nachrichtenkanal-Netz. Der Quellcodeausschnitt 4.6 enthält den 3. Teil für die Beschreibungsdatei des Beispiels und deklariert die Senderfunktionalität in einem `<server>`-Element. Für die

bessere Übersicht wird lediglich der Nachrichtenkanal des Temperatursensors im Detail dargestellt. Da in der HRT-Version das Senderverhalten für den Nachrichtenkanal die harten Echtzeitverarbeitungsbedingungen auf Empfängerseite bestimmt, muss der Kanal "exklusiv" von einem Knoten genutzt werden (vgl. Zeile 26)[11]. Sein eindeutiger Name (vgl. Referenz in Zeile 14 im Quellcodeausschnitt 4.5) ergibt sich aus Name des Knotens, Name des `<server>`-Elements (vgl. Zeile 25) und Name der Kanalbeschreibung. Die Referenz auf den Dekorierer und Kommunikationssockel in Zeile 28 wird analog gebildet. Außer diesen Konfigurationsparametern enthält die Kanalbeschreibung genaue Angaben zum Nachrichtensendeverfahren. Mit der Angabe von `<start>`-, `<periodic>`-, `<jitter>`- und `<deadline>`-Elementen werden diese Zeitfristen beschrieben. Mit der Angabe von möglichen Nachrichtentypen wird die Kanalbeschreibung vollständig.

Das XML-Element `<events>` enthält die verwendete Klasse der übertragenen Nachrichten und deklariert Integer-Bezeichner für die Kodierung unterschiedlicher Nachrichtentypen. Die Angabe der Größe erlaubt der Implementierung der Rahmenarchitektur den Empfangs- und Kopieraufwand zeitlich zu begrenzen.

Neben der Beschreibung von diesem Senderknoten angebotener Nachrichtenkanäle enthält das `<server>`-Element ein optionales XML-Element zur Deklaration einer Aktionsklasse. Diese Klasse kann die Programmlogik des Knotens enthalten und unterstützt damit die Programmcodegenerierung des deklarativen und komponentenbasierten Entwurfs verteilter Systeme. Quellcodeausschnitt 4.7 zeigt den 4. Teil der XML-Datei mit dem dazu notwendigen `<action>`-Element. Dieses Element enthält den Klassennamen der auszuführenden `AbstractAction`-Unterklasse und optional Attribute für die Erzeugungsfabrikmethode.

Da die Implementierung des Beispielszenarios auf nur einem Kommunikationsknoten ausgeführt werden soll, ist die Empfänger- bzw. Steuerkomponente (`Controller`) ebenfalls Teil der XML-Datei (vgl. Quellcodeausschnitt 4.8). Notwendig ist ein `<client>`-Element, das Behandler und Parameter der Aktivitätskontrollfäden beschreibt. Der Übersichtlichkeit wegen ist nur die Beschreibung für den Nachrichtenkanal des Temperatursensors angegeben.

Der Behandler besteht hauptsächlich aus der Referenz auf eine Aktionsklasse. Klassenname und Parameter für die Erzeugung werden hier angegeben (vgl. Zeile 152-157). Des Weiteren wird die Kapazität (`capacity`) der

[11] Mit der FRT-Version können sich mehrere Sender einen Nachrichtenkanal teilen.

Eingangswarteschlange der Verarbeitungslogik beschrieben und die Zeit für den schlechtesten Ausführungsfall (`wcet`) angegeben. Das Beispiel verarbeitet jede Nachricht direkt, die Eingangswarteschlange ist deshalb nur ein Element groß, und die Verarbeitung soll im schlechtesten Fall 50 Millisekunden nicht überschreiten. Diese Zeitgrenze wird im nächsten Abschnitt 4.5 beim Einsatz von MAST praktisch verwendet.

```
025: <server name="Server">
026:  <channel name="TemperatureChannel" exclusive="true">
027:   <description>Temperatursensor</description>
028:   <wrappersocket>
        AllInOne.Socket.TemperatureSocket
       </wrappersocket>
029:   <start>
030:    <time>
031:     <millis>100</millis>
032:     <nanos>0</nanos>
033:    </time>
034:   </start>
035:   <periodic>
036:    <time>
037:     <millis>600</millis>
038:     <nanos>0</nanos>
039:    </time>
040:   </periodic>
041:   <jitter>
042:    <time>
043:     <millis>50</millis>
044:     <nanos>0</nanos>
045:    </time>
046:   </jitter>
047:   <deadline>
048:    <time>
049:     <millis>300</millis>
050:     <nanos>0</nanos>
051:    </time>
052:   </deadline>
```

Quellcode 4.6: XML-Beschreibung der Steuerung eines Produktionssystems (1-Knoten-Version, Teil 3 von 5)

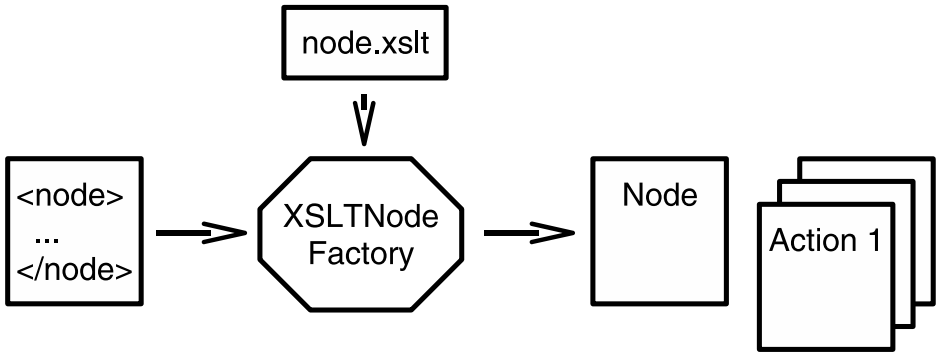

Abbildung 4.12: XSL Transformation für die Programmcodegenerierung

```
053:   <events>
054:    <class>
         de.fzi.ecn.hrt.PrimitiveIntRemoteEvent
        </class>
055:    <event name="VALUE" id="1">
056:     <size>10</size>
057:    </event>
058:   </events>
059:  </channel>
```

Quellcode 4.6 (Fortsetzung): XML-Beschreibung der Steuerung eines Produktionssystems (1-Knoten-Version, Teil 3 von 5)

Die Beschreibung des notwendigen Aktivitätskontrollfadens beginnt ab Zeile 195. Für die HRT-Version wird genau ein Kontrollfaden (vgl. Zeile 198) für die Verarbeitung reserviert. Seine Priorität ist, wie oben beschrieben, abhängig vom zugeordneten Nachrichtenkanal (`AllInOne.Server.TemperatureChannel`), hier wird die Kapazität (`capacity`) und das Verhalten bei Überlauf (`overwrite`) des Warteschlangensystems konfiguriert.

```
140:  <action>
141:   <class>
        de.fzi.ecn.test.control.AllInOneAction
       </class>
142:   <attributes>
143:    <value>AllInOne Server</value>
144:   <attributes>
145:  </action>
146: </server>
```

Quellcode 4.7: XML-Beschreibung der Steuerung eines Produktionssystems (1-Knoten-Version, Teil 4 von 5)

Da das Beispiel zur einfachen Vorstellung des XML-Formats mit nur einem Sender- und gleichzeitigen Empfängerknoten implementiert wurde, fehlen einige Elemente, die sonst in der XML-Beschreibung genutzt werden können. Jeder Kommunikationsknoten benötigt die Beschreibung der verteilten Applikation in einer XML-Datei. Sender- und Empfängerknoten müssen dabei die gleiche Beschreibung des gemeinsamen Nachrichtenkanals verfügen. Für die Beschreibung des Nachrichtenkanals ist aber nur der Sender verantwortlich. Auf Empfängerseite wird ein Stellvertreterelement (`<proxy>`) als Kopie der Konfiguration verwendet. Quellcodeausschnitt 4.9 zeigt ein entsprechendes Beispiel einer solchen Kopie für einen FRT-Nachrichtenkanal. Zusätzliche Besonderheit ist die explizite Angabe der Priorität (vgl. Zeile 4) in der FRT-Version für die verarbeitenden Aktivitätskontrollfäden. Die Angaben im Element `<wrappersocket>` beschreiben anders als für die HRT-Version auch nicht einen Dekorierer, sondern referenzieren direkt das `BroadcastSocket`-Objekt des Kommunikationssockels.

Durch den Einsatz einer XSL Transformationsbibliothek (z.B. Saxon) kann aus den XML-Beschreibungsdateien Standard-Programmcode des Kommunikationsknotens generiert werden. Abbildung 4.12 zeigt den Generationsprozess. Aus der XML-Beschreibung eines Knotens wird mit einer XSLT-Datei Programmcode für die Kommunikation der Hauptsteuerklasse (`Node`) und eventuell Rahmen für abhängige Aktionsklassen (`Action 1,...`) erzeugt. Der Vorteil für Systeme mit harten Echtzeitbedingungen liegt insbesondere darin, dass keine Interpretation der Konfiguration zur Laufzeit mehr notwendig ist. Außerdem stellt dieses Verfahren sicher, dass die Implementierung dem ebenfalls aus der Konfiguration generierten Analysemodelle (vgl. Abschnitt 4.5) entspricht.

```
150:  <client name="Controller">
151:   <handler name="Temperature">
152:    <action>
153:     <class>
          de.fzi.ecn.test.control.TemperatureAction
         </class>
154:     <attributes>
155:      <value>AllInOne</value>
156:     </attributes>
157:    </action>
158:    <capacity>1</capacity>
159:    <wcet>
160:     <time>
161:      <millis>50</millis>
162:      <nanos>0</nanos>
163:     </time>
164:    </wcet>
165:   </handler>

195:   <activity
        channel="AllInOne.Server.TemperatureChannel">
196:    <capacity>10</capacity>
197:    <overwrite>false</overwrite>
198:    <num>1</num>
199:    <handlerref
         ref="AllInOne.Controller.Temperature" />
200:   </activity>
```

Quellcode 4.8: XML-Beschreibung der Steuerung eines Produktionssystems (1-Knoten-Version, Teil 5 von 5)

```
01: <proxy ref="Server" id="2">
02:  <channel name="Channel" exclusive="true">
03:   <description>FRT Example Channel</description>
04:   <priority>20</priority>
05:   <wrappersocket>Node.Socket</wrappersocket>
06:   <start> ... </start>
07:   <periodic> ... </periodic>
08:   <jitter> ... </jitter>
09:   <deadline> ... </deadline>
10:   <events> ... </events>
11:  </channel>
12: </proxy>
```

Quellcode 4.9: Beispiel der XML-Beschreibung eines <proxy>-Elements eines FRT-Beispiels

4.5 Analyse mit MAST

Die Rahmenarchitektur für asynchrone Nachrichtenkommunikation ermöglicht bei harten Echtzeitbedingungen die statische Analyse der notwendigen Echtzeitkontrollfäden auf Ablauffähigkeit und Einhaltung aller Zeitfristen. Für sicherheits- und geschäftskritische Systeme ist eine durchgängige Entwicklungsmethodik, die neben Entwurf auch Generierung von Programmcode und Analysemodell unterstützt, hilfreich. Voraussetzung ist zum einen ein Verfahren der Ablaufkoordinierung mit festen Prioritäten wie es in Abschnitt 3.4.1 beschrieben wird und zum anderen Bearbeitungszeiten im schlechtesten Ausführungsfall.

Die Implementierung der Rahmenarchitektur mit Echtzeit-Java für harte Echtzeitbedingungen unterstützt eine solche Analyse mit der Werkzeugsuite MAST [29]. Da alle Kontrollfäden für den Anteil der Kommunikation aus der XML-Beschreibung der verteilten Applikation zu ermitteln sind, wird für diese ein Modell als Eingabe für das MAST Werkzeug erzeugt. Der verwendete Algorithmus für die Berechnung der Modellinformationen wird im Anhang C mit Java-Programmcode verdeutlicht.

In Abschnitt 3.4.1 wurde bereits die Notwendigkeit der Abstraktion von den tatsächlichen Echtzeitkontrollfäden verdeutlicht. Wie die Abbildung 3.9 auf Seite 74 zeigt, werden dafür Pseudo-Empfänger- und Pseudo-Aktivitätsmanagerkontrollfäden ermittelt. Diese ergeben sich direkt aus den Parametern der zu empfangenden Nachrichtenkanäle (vgl. Tabelle 1.1 auf

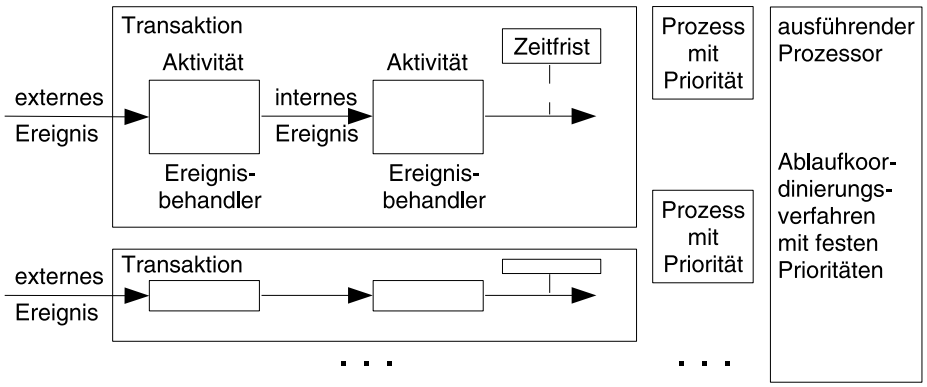

Abbildung 4.13: Echtzeitsystem modelliert mit Transaktionen

Seite 6). Das eingesetzte Werkzeug der MAST-Suite basiert auf einem Analyseverfahren mit linearem Modell [47]. Ähnlich einer Nachricht im Empfänger der vorgestellten Rahmenarchitektur wird ein periodisches, externes Ereignis verwendet um Transaktionen im Analysemodell zu starten. Eine lineare Folge vom externen und internen Ereignisse mit Aktivitäten zur Ereignisbehandlung (vgl. Abbildung 4.13) wird definiert und die Ausführbarkeit eines Echtzeitsystems mit Prozessen[12] unterschiedlicher Priorität und einem Prozessor simuliert. Die Abbildung 4.13 illustriert ein Modell mit mehreren nebenläufigen Transaktionen, die von Prozessen mit unterschiedlicher Priorität auf einem Prozessor ausgeführt werden. Als Verfahren zur Ablaufkoordinierung unterstützt dieser Prozessor FPS (vgl. Abschnitt 3.4.1).

Anders als der vorgestellte Nachrichtendienst, mit einem Echtzeitkontrollfaden höchster Priorität für den Empfang aller Nachrichten eines Kommunikationssockels, erlaubt das lineare Modell nicht die Verarbeitung mehrerer Ereignisse[13] durch eine Behandleraktivität. Für nebenläufige Nachrichten sind deshalb mehrere Transaktionen notwendig. Bei der Berechnung des ungünstigsten Zeitverhaltens der Ausführung beeinflussen sich diese Transaktionen. Um für den Arbeitsaufwand von Empfänger- und Aktivitätsmanagerkontrollfaden der Rahmenarchitektur eine realistische Nachbildung in MAST zu erreichen, werden die Aktivitäten dieser Kontrollfäden gesplittet und überlappungsfrei von externen periodischen Ereignissen im MAST-Modell ausgelöst. Diese Ereignisse starten Transaktionen, die sich bei der Empfangsarbeit (mit Kontrollfäden gleicher Prioritäten)

[12] Java-Kontrollfäden werden für dieses Modell wieder als Prozesse aufgefasst.
[13] Der Ablauf innerhalb einer Transaktion ist immer linear.

nicht überschneiden[14]. Nur die Behandlerlogik der Aktivitätskontrollfäden müssen sich den Prozessor bei überlappender Arbeit teilen.

Die Pseudo-Kontrollfäden wiederholen sich in Runden, die als kleinstes gemeinsames Vielfache (kgV) aller Perioden ermittelt werden.

Runde: $kgV(600, 1200, 400) = 1200$ Millisekunden entspricht der Periode der Pseudo-Kontrollfäden für Empfang und Verwaltung der Nachrichten.

Neben der Periode kann für jeden der Pseudo-Empfänger- und Pseudo-Aktivitätskontrollfäden der erste Aktivierungszeitpunkt innerhalb der ersten Runde (vgl. wieder Abbildung 3.9 auf Seite 74) errechnet werden.

Die maximale Arbeit eines Pseudo-Empfängerkontrollfäden hängt von der Anzahl und Art der durch ihn zusammen empfangenen Nachrichten ab. Die notwendige Arbeit für das Lesen des Eingangspuffers und das Kopieren in die Warteschlange muss hierfür bekannt sein. Außerdem kann das notwendige Aktivitätsfenster des Kontrollfadens für den ungünstigsten Fall aus dem Maximalwert der überlappenden Jitterzeiten der Nachrichtenkanäle bestimmt werden.

Der Quellcodeausschnitt 4.10 zeigt einen Ausschnitt der für das MAST-Werkzeug notwendigen Steuerdatei[15]. Die Transaktion des 1. Pseudo-Empfängerkontrollfadens (für Empfang von Kanal 1 und 3, `PseudoReceiver-K1K3`) wird hier beispielhaft erklärt. Der überlappende Eintreffzeitpunkt für Kanal 1 und 3 am Anfang der Runde ergibt diese Kombination und das resultierende Empfangsintervall.

Abhängig vom periodischen (`Periodic`) externen Ereignis (`ExternalEvent-ID`) mit erstem Aktivierungszeitpunkt 100ms, Periode 1200ms und Jitter von 50ms wird das interne Ereignis (`InternalEventID`) gestartet. Da erst für die Verarbeitung der Nachricht Zeitfristen deklariert sind, wird als Frist (`Deadline`) die Periode des Pseudo-Empfängerkontrollfadens verwendet. Wichtig ist der Abschnitt zur Ereignisbehandlung (`Event_Handlers`). Die beschriebene Aktivität (`Activity`) führt eine Operation (`Activity_Operation`) mit einem Prozess (`Activity_Server`) in fester Priorität aus.

[14] Überschneidende Arbeit mit gleicher Priorität ergäbe für die Verarbeitungszeit unter schlechtesten Bedingung bei der Anlyse in MAST immer die Blockierung eines Kontrollfadens, was nicht dem tatsächlichen Verhalten der Rahmenarchitektur mit nur einem Kontrollfaden entspricht.

[15] Das Dateiformat verlangt bei Zeitangaben keine Einheit und die Berechnung erfolgt immer auf der Basis von Nanosekunden.

```
Transaction {
 Type                  => Regular,
 Name                  => PseudoReceiverK1K3,
 External_Events       => {
   Type                => Periodic,
   Name                => ExternalEventID,
   Period              => 1200000000,
   Max_Jittr           =>   50000000,
   Phase               =>  100000000
 },
 Internal_Events       => {
   Type                => Regular,
   Name                => InternalEventID,
   Timing_requirements => {
    Type               => Hard_Global_Deadline,
    Deadline           => 1200000000,
    ReferencedEvent    => ExternalEventID
   }
 },
 Event_Handlers  => {
   Type                => Activity,
   Input_Event         => ExternalEventID,
   Output_Event        => InternalEventID,
   Activity_Operation  => Receive2EventsWCET,
   Activity_Server     => Receiver
 }
};
```

Quellcode 4.10: Transaktion im MAST-Modell für Echtzeitsysteme

	K1+K3 (1)	K2+K3 (2)	K1 (3)	K3 (4)
1. Aktivierungszeitpunkt	100	350	700	900
Periode	1200			
Jitter	50	200	50	50
Bearbeitungsfrist	-			
WCET	16	20	8	8

Tabelle 4.2: Parameter der Pseudo-Empfängerkontrollfäden in Millisekunden

```
Operation {
 Type                      => Simple,
 Name                      => Receive2EventsWCET,
 Worst_Case_Execution_Time => 16
};

Scheduling_Server {
 Type                       => Fixed_Priority,
 Name                       => Receiver,
 Server_Sched_Parameters    => {
  Type                      => Fixed_Priority_Policy,
  The_Priority              => 38
 },
 Server_Processing_Resource => CPU
};
```

Quellcode 4.11: Weitere Elemente des MAST-Modells

Die Referenzen auf eine Operation und einen Prozess sind durch Namen (`Receive2EventsWCET` u. `Receiver`) gegeben. Im Programmcodeausschnitt 4.11 werden die entsprechenden Beschreibungen in der Steuerdatei aufgeführt. Der Prozess (`Scheduling_Server`), der von allen Transaktionen der Pseudo-Empfängerkontrollfäden gemeinsam benutzt wird, hat die höchste in RTSJ verfügbare Priorität `38`. Ausgeführt wird er durch den Prozessor (`CPU`), der bei der Ablaufkoordinierung ein Verfahren mit festen Prioritäten unterstützt. Die Angabe in der Operationsbeschreibung (`Receive-2EventsWCET`) ist eine Zeitabschätzung für den Empfang von zwei Nachrichten (von Nachrichtenkanal 1 u. 3) im schlechtesten Fall (`Worst_Case_Execution_Time`).

Die Tabelle 4.2 zeigt erste Aktivierungszeitpunkte, Perioden und Jitter der ermittelten Pseudo-Empfängerkontrollfäden. Die Zeile WCET beschreibt

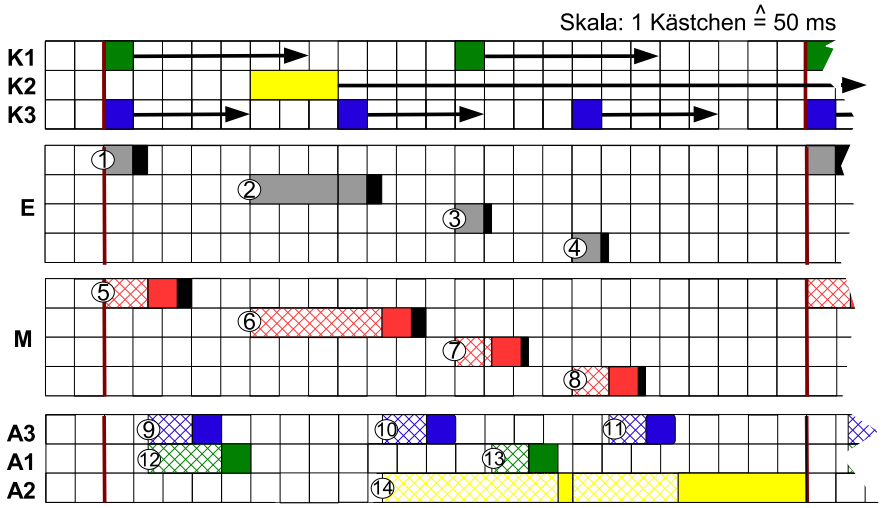

Abbildung 4.14: Modell der Echtzeitkontrollfäden

	K1+K3 (5)	K2+K3 (6)	K1 (7)	K3 (8)
1. Aktivierungszeitpunkt	100	350	700	900
Periode	1200			
Jitter	50			
Bearbeitungsfrist	-			
WCET	16	20	8	8

Tabelle 4.3: Parameter der Pseudo-Kontrollfäden des Aktivitätsmanagers in Millisekunden

dabei den maximalen Arbeitsaufwand für die Entgegennahme von einer bzw. zwei Nachrichten. Die Jitter-Zeitspannen der Nachrichtenkanäle ergeben eine maximale Aktivitätszeit für den Empfang.

Da der Aktivitätsmanagerkontrollfaden der Rahmenarchitektur durch den Empfängerkontrollfaden aktiviert wird, werden zur Repräsentation der notwendigen Arbeiten im Modell ebenfalls vier Pseudo-Kontrollfäden definiert. Ihre Ausführung im schlechtesten (d.h. arbeitsintensivsten) Fall beginnt deshalb mit der gleichen Startzeit wie der jeweilige Pseudo-Empfängerkontrollfaden. Die Arbeit der Pseudo-Aktivitätsmanagerkontrollfäden teilt sich in einen festen Standardaufwand (im Beispiel ca. 50ms) und einen variablen Aufwand zum Kopieren der Nachrichten.

Auch die Aktivitätskontrollfäden werden im Modell als Pseudo-Kontrollfaden dargestellt. Ihr Startzeitpunkt liegt zeitgleich zu Pseudo-Empfänger- und Pseudo-Aktivitätsmanagerkontrollfaden. Durch die geringere Pri-

	K1 (9)	K1 (10)	K1 (11)	K3 (12)	K3 (13)	K2 (14)
1. Aktivierung[A]	100	350	900	100	700	350
Periode	1200					
Jitter	-					
Frist	200	200	200	300	300	1200
WCET	50	50	50	50	50	250

[A] Die hier angegeben ersten Aktivierungszeitpunkte für Kontrollfäden entsprechen den zu Beginn geplanten ersten Eintreffzeitpunkten der einzelnen Nachrichten. Die dynamische Neuberechnung des Aktivierungszeitpunktes, die in Abschnitt 3.5.6 auf Seite 60 beschrieben wird, ist hier nicht eingerechnet.

Tabelle 4.4: Parameter der Pseudo-Aktivitätskontrollfäden

orität (vgl. Tabelle 4.1 auf Seite 98) ist sichergestellt, dass die Ausführung erst nach Empfang der zugeordneten Nachrichten erfolgen kann. Abbildung 4.14 verändert noch mal Abbildung 3.9 auf Seite 74 und zeigt alle 14 Pseudo-Kontrollfäden des Modells für Nachrichtenempfang, -verwaltung und -verarbeitung der drei Nachrichtenkanäle. Die Tabellen 4.3 und 4.4 zeigen die resultierenden Parameter der korrelierenden Transaktionen und Operationen der MAST-Steuerdatei. Titel für die einzelnen Spalten ergeben sich immer aus dem Bezeichner der Nachrichtenkanäle 1 bis 3 und der Nummerierung des Pseudo-Kontrollfadens in Abbildung 4.14.

Durch Ausnutzung der zusätzlichen Funktionalität des MAST-Werkzeuges kann als Ausgabe der Analyse auch eine Beschreibung der notwendigen Prioritätshöchstmaße für die nebenläufig verwendeten Monitore erzeugt werden. Diese Informationen sind Grundlage für eine Rückkopplung in der vorgeschlagenen Entwicklungsmethodik. Ähnlich wie die für eine Ausführungsplattform (virtuelle Maschine und Hardware) überprüfte Einhaltung von Echtzeitbedingungen geben diese Analyseergebnisse dem Entwickler konkrete Unterstützung bei der Implementierung verteilter Applikationen.

Bei der Evaluierung des Prototyps der Rahmenarchitektur in Abschnitt 5.4 wird die mögliche statische Ablaufplananalyse näher untersucht. Als Beispiel werden Ergebnisse der Analyse des Beispielszenarios mit MAST vorgestellt. Die erzeugten periodischen Pseudo-Echtzeitkontrollfäden werden für den schlechtesten Fall der Ausführung untersucht und das erzeugte Modell bewertet.

4.6 Zusammenfassung

Echtzeit-Java eignet sich als objektorientierte Laufzeitumgebung für die im letzten Kapitel entworfene Rahmenarchitektur eines asynchronen Nachrichtendienstes. Ausgehend von einer virtuellen Maschine, die RTSJ unterstützt, werden zwei echtzeitfähige Implementierungen der Rahmenarchitektur vorgestellt. Basierend auf Eigenschaften und Besonderheiten von Echtzeit-Java werden Anforderungen an eine HRT- und FRT-Version der Java-Implementierung des Nachrichtenkanal-Netzes gestellt. Die Umsetzung der Softwareentwurfsmuster wird eingeführt.

Für die Nutzung der durchgängigen Entwicklungsmethodik wird eine XML-Beschreibung verteilter Systeme vorgestellt. Mit Einsatz von XSLT können aus dieser Beschreibung Rahmen für die notwendigen Java-Klassen der Implementierung erzeugt werden. Die XML-Beschreibung erlaubt als domänenspezifische Sprache die Konfiguration der Kommunikationsanteile einer verteilten Applikation. Durch Beschreibung von Echtzeitbedingungen und Angabe von Ausführungszeiten im schlechtesten Fall kann die XML-Datei auch als Basis für eine statische Analyse der notwendigen Echtzeitkontrollfäden verwendet werden. Die Prototypimplementierung unterstützt dazu die MAST-Werzeugsuite zur Analyse von Echtzeitsystemen.

Kapitel 5

Evaluierung

Die Evaluierung einer neuen Methodik für die Softwareentwicklung erfordert eigentlich eine breite praktische Erprobung und den Einsatz in unterschiedlichen Softwareentwicklungsprojekten mit verschiedenen Entwicklungsteams. Der hier vorgestellte Ansatz für die Beschreibung verteilter Systeme mit Nutzung einer asynchronen Kommunikationsinfrastruktur konnte durch die Entwicklung in zwei europäischen Forschungsprojekten profitieren. Asynchrone Nachrichtenkommunikation wurde für HIDOORS [115] und HIJA [56] als Grundlage verteilter Systemen identifiziert und erfolgreich eingesetzt. Die vorgestellte Methodik resultiert aus Erfahrungen in diesen Projekten. Weitere Informationen zu beiden Forschungsprojekten finden sich in Abschnitt 5.1.

Da für die Evaluierung leider nicht auf äquivalente Parallelversuche zurückgegriffen werden konnte, wird die Eignung der vorgestellte Entwicklungsmethodik auf Basis der drei in Abschnitt 1.1 eingeführten Thesen beurteilt. Diese werden untersucht und im Abschluss in Abschnitt 5.5 ihr Nutzen in einer Methodik für die durchgängige Entwicklung verteilter Systeme mit Echtzeitbedingungen verdeutlicht. Die Notwendigkeit für eine durchgängige Methodik bei der Entwicklung verteilter Systeme für Rundrufnetze mit asynchroner Nachrichtenkommunikation und komponentenbasierter Systembeschreibung wird exemplarisch belegt.

5.1 Einsatz in EU-Forschungsprojekten

Die in diesem Abschnitt vorgestellten Forschungsprojekte wurden über das “Information Society Technologies”-Förderprogramm (IST) im 6. Rah-

menprogramm der Europäischen Union finanziell unterstützt [63]. Beide Forschungsprojekte basieren auf Echtzeit-Java und der Entwicklung für eingebettete Systeme. Asynchrone Nachrichtenkommunikation wurde dabei in HIDOORS und HIJA als eine Form der Interaktion in komponentenbasierten verteilten Systemen identifiziert und eine notwendige Rahmenarchitektur entwickelt.

5.1.1 HIDOORS

Das Forschungsprojekt HIDOORS ("High Integrity Distributed Object-Oriented Realtime Systems") [52] wurde von der Europäischen Union im IST-Förderprogramm unterstützt (IST 2001-32329). In einem Team aus bis zu acht unterschiedlichen europäischen Forschungseinrichtungen, Universitäten und klein- und mittelständischen Unternehmen wurde die Möglichkeit von Echtzeit-Java auf eingebetteten Systemen untersucht. Die Nutzung objektorientierter Techniken beim Entwurf mittels der UML erlaubte dabei die Steigerung des Abstraktionslevels bei der Entwicklung.

Das Nachrichtenkanal-Netz war Teil der Forschungsarbeit des Forschungszentrums für Informatik (FZI) in diesem Zusammenhang. Die Entwicklung einer plattformunabhängigen Beschreibungsform mit XML, sowie die Generierung von Programmcode für die Kommunikationsanteile wurden hier entwickelt.

5.1.2 HIJA

Die allgegenwärtige ("ubiquitous") Nutzung von eingebetteten Systemen ist auch das Thema des zweiten EU-Forschungsprojektes (IST 2003-511718). Mit HIJA ("High Integrity Java") [54] wird besonders die Entwicklung von ANRTS ("Architecturally Neutral, high-integrity Real-Time Systems") Applikationen unterstützt. Die Java-Sprachspezifikation in Version 5 [41] mit der Echtzeiterweiterung RTSJ ist Grundlage der Entwicklung für solche sicherheits- und geschäftskritischen Systeme. Neben sprachlichen Mitteln wie der Erweiterungen des Programmcodes durch Annotationen zur Unterstützung einer Entwicklungs- und Laufzeitumgebung werden Nebenläufigkeit und ein Verfahren zur Ablaufkoordinierung mit festen Prioritäten für die Entwicklung von hart und weich echtzeitfähiger und verteilter Applikationen untersucht.

Die Weiterentwicklung der Java-Implementierungen des Nachrichtenkanal-Netzes ist Grundlage für die Evaluierung der vorgestellten Rahmenarchitektur. Da eingebettete Systeme in sehr unterschiedlichen Umgebungen eingesetzt werden, ist das Ergebnis des Forschungsprojektes auch die Beschreibung von speziellen Profilen für RTSJ. Die Java-Implementierungen der Rahmenarchitektur (vgl. Kapitel 4) nutzen besondere Eigenschaften dieser Profile für harte bzw. weiche Echtzeitbedingungen aus. Für die Evaluierung der Rahmenarchitektur und der vorgeschlagenen Methodik wird in Abschnitt 5.4 besonders die Integration der Applikationsentwicklung mit dem Analysewerkzeug MAST herangezogen. Der Abschnitt 5.2 untersucht als Ergänzung dazu die Vorteile der Rahmenarchitektur beim Einsatz von Kommunikationsnetzen mit Rundruflogik. Ein Gleiche-zu-Gleichen-Kommunikationsprotokoll für die Implementierung von vollständig dezentralen, skalierbaren und selbstverwaltenden Überlagerungsnetzen wurde dazu implementiert.

5.2 Asynchrone Nachrichtenkommunikation mit Rundrufnetzen

Eine für Kommunikationsnetze mit Rundruflogik angepasste asynchrone Kommunikationsform ist zwar für den Einsatz in sicherheits- und geschäftskritischen verteilten Systemen noch ungewohnt, aber die Eignung des in Kapitel 3 vorgestellten plattformunabhängigen Entwurfs wird durch die in Kapitel 4 beschriebenen unterschiedlichen Java-Umsetzungen erfolgreichen demonstriert. Sie zeigen die Möglichkeit für asynchrone Kommunikation mit Echtzeitbedingungen bei der Verarbeitung.

Das dynamische Kommunikationsmodell (bei weichen Echtzeitbedingungen) unterstützt keine statische Programmablaufanalyse. Ziel ist es aber für die eingeführte Rahmenarchitektur den einfachen Einsatz mit verteilten Systemen (hier am Beispiel einer Bibliothek für Gleiche-zu-Gleichen-Systeme) zu demonstrieren.

In diesem Abschnitt wird die Eignung asynchroner Nachrichtenkommunikation bei der Entwicklung verteilter Systeme untersucht. Die Verwendung von Netzen mit Rundruf- oder Gruppenruflogik ergibt sich aus den im Bereich eingebetteter Systeme weit verbreiteten Verbindungsnetzen und Feldbussystemen (vgl. Abschnitt 2.6). Die einfache Verkabelung von günstigen und leistungsstarken Sensoren und Aktoren ist der Vorteil dieser Systeme. Diese Art der Systeme erlaubt überall vorhandenes (“pervasi-

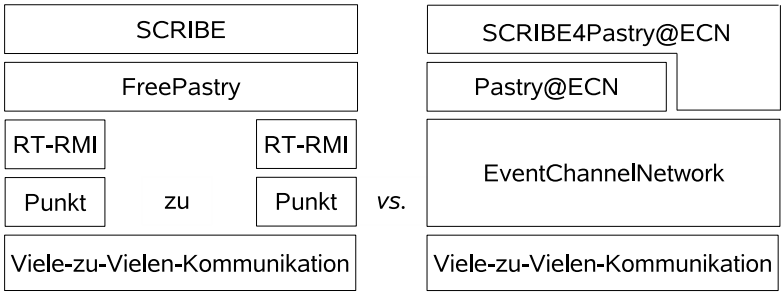

Abbildung 5.1: Stapel im Vergleich

ve") Rechnen. Die Integration in andere technische Systeme der Umgebung (vgl. Abschnitt 2.4 über eingebettete Systeme) ermöglicht dann allgegenwärtige Applikationen für die Vision "Ambient Intelligence".

Grundlage bei der Bewertung der Rahmenarchitektur für asynchrone Nachrichtenkommunikation ist die Implementierung einer Programmbibliothek für Überlagerungsnetze. *Pastry@ECN* unterstützt die von Microsoft Research in Cambridge entwickelte Gleiche-zu-Gleichen-Verteilungsplattform Pastry [89] auf Basis von Rundrufnetzen. Diese Adaption wurde innerhalb des Forschungsprojektes HIJA auf der Basis von ECN entwickelt [65]. Eigenarten des Nachrichtenkanal-Netzes werden dabei ausgenutzt, um den Kommunikationsaufwand in Netzen mit Viele-zu-Viele-Kommunikation zu reduzieren.

Ausgangspunkt der Implementierung ist die Java-Implementierung FreePastry auf Basis von RMI [4]. Liebrich [66] hat durch Vergleiche die Protokollkonformität der neuen Implementierung mit der Spezifikation und dieser Orginalversion von FreePastry belegt.

Die Verwendung einer Rahmenarchitektur für asynchrone Nachrichtenkommunikation auf Basis von Viele-zu-Viele-Kommunikation motiviert sich durch den Vergleich in Abbildung 5.1. Der notwendige Protokollstapel für Pastry in einem Rundrufnetz wird durch ECN vereinfacht. Eine aufwändige Emulation der Punkt-zu-Punkt-Verbindungskommunikation für den Einsatz entfernter Objekte in RMI wird eingespart. FreePastry verwendet verbindungsorientierte Kommunikation um auf Applikationsebene wieder ein Viele-zu-Viele-Protokoll bereitzustellen. Die Nachrichtenkommunikation der Rahmenarchitektur bietet dies ebenfalls und setzt direkt auf das Rundrufnetz auf.

Neben der Pastry-Bibliotheksschnittstelle zeigt die Abbildung 5.1 auch eine Schnittstelle des dezentrale Gruppenrufsystems SCRIBE [24]. *SCRIBE4-Pastry@ECN* eine Implementierung dieses Systems für das Nachrichtenka-

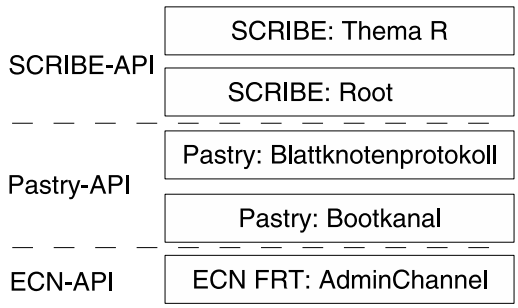

Abbildung 5.2: Nachrichtenkanäle für EventChannelNetwork-, Pastry- und SCRIBE-Verwaltung mit Kanälen der Beispielapplikation (Knoten 0)

nal-Netz ist zur gesteigerten Effizienz direkt mit der Nachrichtenkommunikation über Nachrichtenkanäle verbunden; die SCRIBE-Schnittstelle der Applikationsschicht ist wieder unverändert. Abbildung 5.2 zeigt die Implementierung von SCRIBE über Nachrichtenkanäle, die von ECN, Pastry und SCRIBE verwendet werden.

Bei einer Untersuchung zur Auslastung von Rundrufnetzen durch Applikationen mit Gleiche-zu-Gleichen-Protokollen wurden beide Protokollstapel verglichen. Die Idee war der Vergleich einer SCRIBE-Applikation bei gleicher Netzinfrastruktur mit beiden Stapeln.

Für Systeme mit weichen Echtzeitanforderungen und flexiblen Strukturen ergeben sich zwei Probleme bei der Entwicklung. Ständig neu angeschlossene Geräte, bzw. das Entfernen angeschlossener Geräte, ergeben verglichen zu Arbeitsplatzrechnern sich permanent ändernde Strukturen des Netzes. Dies muss trotz beschränkter Ressourcen von eingebetteten Systemen durch die Entwicklungsmethodik unterstützt werden. Aufwändige Wegewahlverfahren und Kommunikationsprotokolle können nicht verwendet werden. Im privaten Bereich, für zum Beispiel eine Haussteuerung, sind teure Wartungs- und Konfigurationsarbeiten bei der Integration neuer Geräte undenkbar. Aus diesem Problemszenario ergibt sich die Evaluierung der These 1 (vgl. Abschnitt 1.1 auf Seite 2).

Für eine SCRIBE-Applikation zur Überwachung von Temperatur und Lüftung eines Hauses werden unterschiedliche Testfälle definiert. Die Applikation besteht aus über das Haus verteilten Sensoren und Aktoren, die – zur einfachen Vernetzung – zwar nur über ein physikalisches Netz oder einen Feldbus (z.B. LON, "Local Operating Network" [70]) verbunden sind, aber unterschiedliche Gruppen/Themen bei der Steuerung verwenden. Diese Themen in SCRIBE sind als R, G und B benannt und Sensoren und Aktoren für sie sind über das Haus verteilt. Abbildung 5.4 zeigt die drei

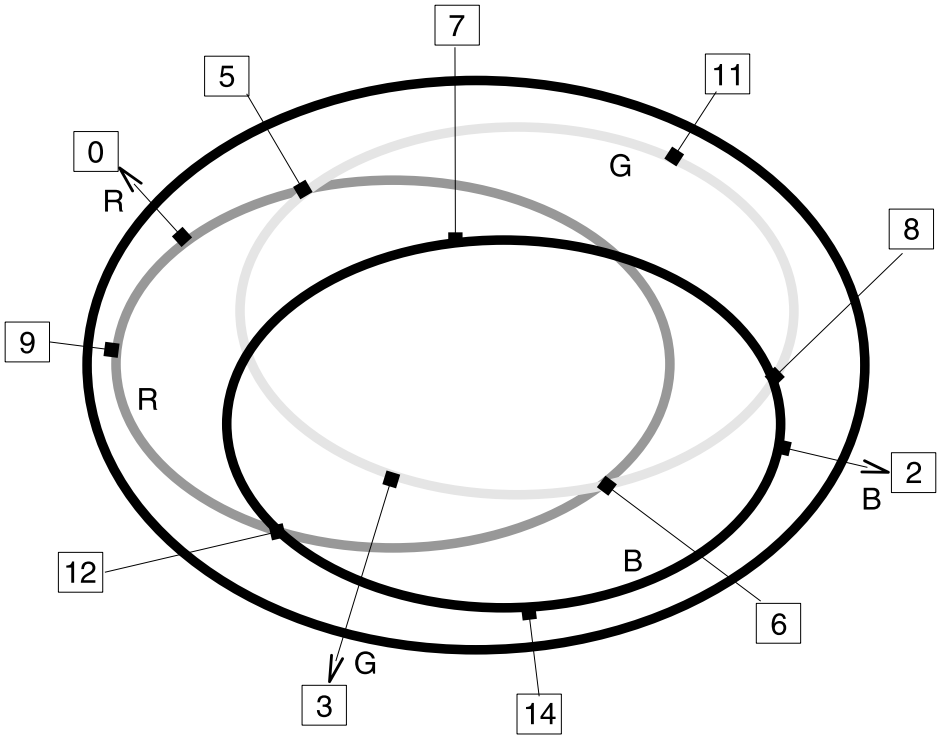

Abbildung 5.3: Pastry-Netz einer Haussteuerung mit drei SCRIBE-Nachrichtenkanälen (R,G,B)

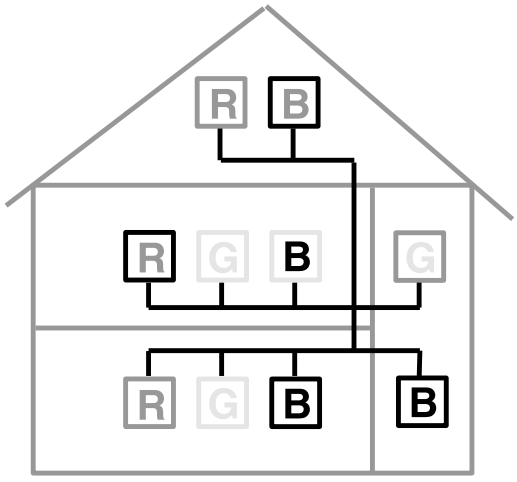

Abbildung 5.4: SCRIBE-Netz für Haustechnik

Funktion	<	ID	>
Empfänger R	14	0	2
Empfänger B	0	2	3
Empfänger G	2	3	5
Sender R, G	5	5	6
Sender R, G	5	6	7
Sender B	6	7	8
Sender B, G	7	8	9
Sender R	8	9	11
Sender G	9	11	12
Sender R, B	11	12	14
Sender B	12	14	0

Tabelle 5.1: Pastry Knoten-ID mit Listen für Blattknoten

Themen im Haus und Abbildung 5.3 illustriert das zugrundeliegende Pastry-Netz der 11 Knoten mit zufälligen Bezeichnern. Das Pastry-Protokoll verwaltet dezentral die Kommunikation der Knoten untereinander. Tabelle 5.1 listet die Knoten mit Funktion, Bezeichner und einer Blattmenge ("leaf set", beobachtete Pastry-Knoten), die eine dezentrale Systemüberwachung garantiert. Wenn ein Knoten verloren geht erkennt das Pastry-Netz dies und auf Applikationsschicht (in der SCRIBE-Applikation) kann dies behandelt werden.

Nur die Knoten 0, 2 und 3 verarbeiten Informationen der drei Themen und reagieren auf Sensorwerte. Die anderen Knoten senden themenbezogene Werte und tauschen Pastry-Verwaltungsnachrichten untereinander aus. Abbildung 5.2 zeigt ein Beispiel der in Knoten 0 verwendeten Nachrichtenkanäle[1].

Da dieser Abschnitt nicht die Funktionstüchtigkeit des Pastry-Protokolls beweisen kann, sondern nur die Eignung von asynchroner Nachrichtenkommunikation in der vorgestellten Rahmenarchitektur mit Rundrufnetzen evaluiert, wird für die Einführung zu Pastry und SCRIBE auf die umfangreiche Dokumentation im Web verwiesen [6].

Für verschiedene Testfällen, die mit der Beispielapplikation auftreten können, lassen sich unterschiedliche Messwerte für die Anzahl und Größe notwendiger Nachrichtenpakete ermitteln.

[1]Für Thema G und B existieren ähnliche Nachrichtenkanäle, die von den anderen Knoten aktiv empfangen werden.

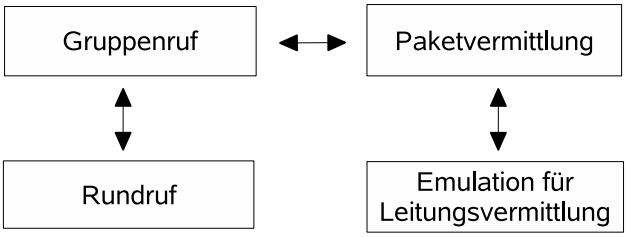

Abbildung 5.5: Vergleichbarkeit der Netzkommunikation am Beispiel von UDP/IP

- Initialisierung des Netzes — Ein Neustart des Netzes verlangt das gegenseitige Finden von Programmknoten, um ein stabiles Überlagerungsnetz zu bilden. Sollwerte an den Aktoren müssen hier verteilt werden.

- Normaler Betrieb — Sensoren senden gemessene Werte an das Netz, zuständige Aktoren verarbeiten diese und senden selbst Lebenszeichen zur Kontrolle des Netzes.

- Knoteneintritt — Ein zusätzlich verfügbarer Knoten integriert sich in das Netz. Die Struktur des Überlagerungsnetzes wird an den neuen Knoten angepasst und wenn es sich um einen Aktor handelt bekommt er seine Sollwerte zugesendet.

- Knotenausfall — Ein Knoten verliert die Verbindung zum Kommunikationsnetz. Das Überlagerungsnetz erkennt den Ausfall und repariert seine Verknüpfungen.

- Gruppenruf — Eine Gruppe von Aktoren bekommt aufgrund von gemessenen Werten neue Sollwerte zugeschickt.

Exemplarisch wird der Kommunikationsaufwand in einem UDP/IP-Netz für die Initialisierung der Pastry bzw. einer SCRIBE-Infrastruktur in den Tabellen 5.2 aufgelistet.

5.2.1 Ergebnis

Problem eines Vergleichs von SCRIBE auf Basis von Pastry mit RMI und Pastry@ECN ist der Einsatz eines vergleichbaren Rundrufnetzes. Für die

Initialisierung von Pastry-Netz	UDP/IP Multicast	UDP/IP Pakete
Pastry@ECN	23.265 Byte	
FreePastry/RMI		415.500 Byte
Initialisierung für SCRIBE	UDP/IP Multicast	UDP/IP Pakete
SCRIBE4Pastry@ECN	35.278 Byte	
SCRIBE mit FreePastry/RMI		522.000 Byte

Tabelle 5.2: Pastry@ECN im Vergleich zu FreePastry mit einem Netz aus 5 Knoten

Java-Implementierung der Rahmenarchitektur mit Pastry existiert ein Zugriffssockels für den Gruppenruf ("Multicast") mit UDP/IP. Für den Einsatz von FreePastry über RMI mit Echtzeit-Java fehlt aber die Emulation der Punkt-zu-Punkt-Verbindung über das gleiche Rundruf-Protokoll. Für die Evaluierung wurden stattdessen einfache Datenpakete mit UDP-Paketvermittlung verwendet.

Vorteil dieses in Abbildung 5.5 im Zusammenhang mit Rundruf und emulierter Leitungsvermittlung dargestellten Vergleichs von Kommunikation mit Gruppenruf und Kommunikation über Paketvermittlung ist, dass das gleiche UDP/IP-Netz eingesetzt werden konnte. Eine RMI-Implementierung auf Basis von UDP-Paketen erlaubt den Vergleich von UDP-Paketen des Gruppenrufs im Nachrichtenkanal-Netzes und der RMI-Kommunikation[2] von FreePastry.

Bei Einsatz mit UDP/IP belegen verschiedene Testläufe mit unterschiedlicher Knotenanzahl für die verschiedenen Testfälle die Eignung asynchroner Nachrichtenkommunikation als Basis einer Pastry-Implementierung. Tabelle 5.2 zeigt die notwendige Menge der Daten (in Byte) für die Initialisierung einer Applikation mit fünf Knoten. Auch wenn für reine Rundrufnetze der Protokollmehraufwand einer Emulation für Punkt-zu-Punkt-Verbindungen nicht überraschend ist, bleibt selbst bei UDP/IP die Datenmenge der reinen UDP/IP-Kommunikation deutlich über der für den Gruppenruf notwendigen Daten. Die Frage, in wie weit beim Rundruf der Filteraufwand auf Empfängerseite diesen Vorteil egalisiert, muss für den konkreten Einsatz (d.h. verteiltes System und Applikation) selbst beantwortet werden.

[2]Grundlage für diesen Vergleich ist eine RMI-Implementierung auf Basis einer Kommunikation mit UDP/IP Datenpaketen.

Beispiel	Anzahl XML-Tags	XML in Byte	Quellcode LOC	Bytecode in Byte
Sender	28	1135	57	1180
Empfänger	32	1493	55	1122
Sender/Empfänger	50	2833	170	7543

Tabelle 5.3: Quantitative Auswertung der XSL-Transformation zur Programmcodegenerierung in HIDOORS

5.3 Generierung von Standardprogrammcode für Kommunikation (HIDOORS)

Das Forschungsprojekt HIDOORS (vgl. Abschnitt 5.1.1) hatte die Nutzung von UML beim Entwurf eingebetteter verteilter Systeme als Ziel. Es wurde eine spezielle Profil-Erweiterung entwickelt und die notwendige Werkzeugunterstützung implementiert. Das sogenannte *HIDOORS-Profil* für die UML ist dabei von der SPT-Spezifikation (vgl. Abschnitt 2.1) abgeleitet. Durch Festlegung neuer Stereotype wurden Kommunikationsmuster in verteilten objektorientierten Echtzeitsystemen vorgegeben. Kommunikation über Puffer, eine schwarze Tafel und mit dem für die vorgestellte Methodik entwickelten Nachrichtendienst sind besondere Formen [58, 68].

Im Rahmen der Entwicklung des UML-Profils wurde die mit der Methodik verwendete XML-Beschreibung für das Nachrichtenkanal-Netz definiert. Sie war eine Vorstufe für die Modellbeschreibung mittels grafischer Notation in UML. Außerdem konnte die Programmentwicklung durch die Generierung von Standardprogrammcode für die Kommunikation mit Nachrichten vereinfacht werden.

5.3.1 Ergebnis

Einige Ergebnisse der Generierung von Standard-Kommunikationsprogrammcode fasst die Tabelle 5.3 quantitativ zusammen. Grundlage sind typische Applikationsbeispiele mit unterschiedlichen Kommunikationsknoten aus der Projekt-Evaluierung. Verglichen wird die Anzahl der XML-Elemente (Tags) mit der Länge des daraus generierten Programmcodes in Programmzeilen (LOC) und der Größe des übersetzten Bytecodes in Byte. Insbesondere das komplexe Beispiel mit Sender- und Empfänger-Komponenten in einem Knoten zeigt wie durch wenig Beschreibung mit

XML-Elementen viel notwendiger Standard-Kommunikationsprogrammcode erzeugt werden kann. Der Programmcode wird dabei zur einfachen Erweiterungen in unterschiedliche Klassen und Unterklassen aufgeteilt und die Größe von 7543 Byte ist die Summe von 4 Klassendateien[3]. Zusätzlich zu implementierende Verarbeitungs- und Programmlogik ist nicht mitgewertet. Diese Behandlerklassen demonstrieren aber die einfache Wiederverwendung von Programmmodulen im Nachrichtenkanal-Netz. Die deklarative Programmiermethode vereinfacht den Austausch und die Rekombination dieser Module auf den einzelnen Kommunikationsknoten.

Diese Evaluierung belegt die These 2 der Aufgabenstellung. Es zeigt sich, dass die asynchrone Nachrichtenkommunikation die Entkopplung von Programmkomponenten vereinfacht. Die Beschreibung von Komponenten und Kanälen in einer XML-Datei erlaubt die Generierung von Standard-Programmcode bei deren Verbindung und für die Kommunikation. Durch die Generierung wird der Entwickler entlastet und bei der Entwicklung verteilter Systeme unterstützt.

Die in dieser Dissertation vorgeschlagene Entwicklungsmethodik nutzt die erkennbare Vereinfachung bei der Implementierung von Programmcode auch für andere Phasen des Entwicklungsprozesses. Die Evaluierung im folgenden Abschnitt 5.4 zeigt den Nutzen besonders für Systeme mit harten Echtzeitbedingungen. Anders als ein dynamisches System, das seine Konfiguration aus einer XML-Datei liest und einstellt, kann aus der XML-Datei erzeugter Programmcode, mit Werkzeugen der WCET-Analyse, auf seine tatsächliche Ausführungszeit hin untersucht werden (vgl. Abschnitt 2.9). Diese Ergebnisse zusammen mit der Analyse des Programmcodes für spezielle Programmfunktionalität erlauben die vollständige statische Analyse eines verteilten Systems für harte Echtzeitbedingungen.

5.4 Modellerzeugung für Analyse von harten Echtzeitbedingungen (HIJA)

Für sicherheitskritische Systeme mit harten Echtzeitbedingungen, zum Beispiel in der Flugzeugtechnik, ist es notwendig, Steuerapplikation vor der Ausführung auf ihre Ablauffähigkeit prüfen zu können. Bei der in diesem

[3]Weitere Klassen mit applikationsabhängiger Verarbeitungslogik sind hier nicht berücksichtigt.

Umfeld notwendigen Zertifizierung kann die statische Prüfung des Ablaufplans aller Echtzeitkontrollfäden von Vorteil sein. Dies wird in These 3 auf Seite 3 für eine Methodik zur Unterstützung einer durchgängigen Entwicklung gefordert. Die Systementwicklung mittels Beschreibung und die Generierung von Programmcode wird hierfür genutzt. Struktur und Verhalten des generierten Programmcodes sind bekannt und es kann automatisch ein äquivalentes Modell für die Analyse erzeugt werden.

Durch Integration der Ergebnisse einer Analyse von Ausführungszeiten unter ungünstigsten Voraussetzungen (vgl. Abschnitt 2.9) in die Beschreibung der Behandlerlogik (vgl. Abschnitt 3.6) kann bei der Modellanalyse mit zum Beispiel MAST (vgl. Abschnitt 4.5) ein Modell mit Fristen für alle nebenläufigen Kontrollfäden untersucht werden.

Um die Machbarkeit dieses Ansatzes zu evaluieren, wird eine im Forschungsprojekt HIJA entwickelte HRT-Java-Version der Rahmenarchitektur eingesetzt. Der Prototyp unterstützt die Modellerzeugung für ein Werkzeug zur statischen Analyse des Ablaufplans von Kontrollfäden mit festen Prioritäten. Dieser Abschnitt ist auch als praktische Anleitung zur Modellanalyse mit MAST innerhalb der Methodik zu verstehen.

Bei der Evaluierung wird erneut auf das Beispielszenario aus Abschnitt 1.2 zurückgegriffen, um Möglichkeiten und Schwierigkeiten der Methodik und Kommunikationsrahmenarchitektur zu ermitteln. Die Generierung eines Teilmodells zur Prüfung des Ablaufplans der Kontrollfäden der Rahmenarchitektur ist Teil der vorgestellten Methodik.

Eine Analyse und Zeitabschätzung für die Ausführungszeit von Programmcode im ungünstigsten Fall ist notwendig. Dies ist mit unterschiedlichen kommerziellen und frei verfügbaren Werkzeugen grundsätzlich möglich. Das in den Forschungsprojekten HIDOORS und HIJA für Echtzeit-Java angepasste Analysewerkzeug Gromit besitzt bisher nur für zwei Prozessoren der PowerPC-Plattform (MPC750, PPC403) und den 16-Bit Mikrocontroller C167 von Infineon Modellbeschreibungen. Für die Evaluierung des Beispielszenarios stand diese Hardware leider nicht zur Verfügung und es musste mit Schätzungen durch den Entwickler[4] gearbeitet werden.

[4] Ausführungszeiten im ungünstigsten Fall abzuschätzen ist durchaus üblich und dieses Verfahren behindert deshalb nicht die eigentliche Evaluierung der vorgestellten Methodik.

Für die Analyse der Ablaufkoordinierung wurde im Rahmen des Forschungsprojektes HIJA die statische Analyse mit MAST erweitert [29, 57]. In der Flugzeugtechnik sind Systeme mit der Partitionierung des Prozessors in Raum und Zeit, die eine sichere Ausführung mehrerer Applikationen ("APplication EXecutive", APEX) garantieren, verbreitet. Hierfür wurde in HIJA ein zusätzliches Analyseverfahren entwickelt [55]. Am Beispiel von ARINC 653 [7] das einen Standard für die Partitionierung von Computerressourcen in Raum und Zeit definiert, kann so der generierte Ablaufplan für die verteilte Beispielanwendung überprüft werden. Das Verfahren wird mit dem Parameter `dm_apex` bei der Analyse ausgewählt und auf einem Entwicklungsrechner mit Linux-Betriebssystem ausgeführt.

```
linux[~] > mast_analysis dm_apex controller.txt out.txt
```

Die Evaluierung des Beispielszenarios verwendet dieses Verfahren und testet maximal mögliche Ausführungszeiten von Programmcode und HRT-Java-Rahmenarchitektur. Das Verfahren ist für die Evaluierung der Methodik hinreichend. Nicht die erfolgreiche Ausführung der zu entwickelnden Applikation für eine spezielle Hardware, sondern die erfolgreiche Integration des Analyseverfahrens soll exemplarisch überprüft werden. Die im Werkzeugaufruf verwendete Steuerdatei (`controller.txt`) wird als Ausgabe der Rahmenarchitektur generiert und Quellcodeabschnitt 5.1 und 5.3 zeigen Ausschnitte des generierten Codes. Der Programmierer muss Komponenten (Transaktionen und Operationen) der Applikationslogik ergänzen. Entsprechende Beschreibungen dazu finden sich schon im Abschnitt 4.5.

Im Quellcodeabschnitt 5.1 werden die notwendige Kontrollfäden der Rahmenarchitektur mit entsprechenden Prioritäten (38, 37, 27, 22, 32) modelliert[5]. Der Kontrollfaden mit höchster Priorität wird zur Ausführung der Pseudo-Empfänger-Transaktionen verwendet. Dieser und der Manager-Kontrollfaden (mit einer Priorität um 1 kleiner als die höchste mögliche Priorität) werden über Behandler in entsprechenden Transaktionen mit periodisch (`Type => Periodic`) geplanten Nachrichten verbunden. Die Periode entspricht der vorher definierten Runde (`Period => 1200000000`). Quellcodeabschnitt 5.2 zeigt beide Transaktionen für den Empfang und die Verwaltung von Nachrichten, die zum Zeitpunkt 100 (`Phase => 10000-0000`) gleichzeitig über Kanal 1 und 3 eintreffen können. Die für den Empfang und die Zuteilung zuständigen Kontrollfäden (`Receiver` und `Manager`)

[5] Die Auswahl der Prioritäten wurde bereits in Tabelle 4.1 auf Seite 98 eingeführt.

werden mit Ausführungszeiten im ungünstigsten Fall (`ReceiverWCET13`, `ManagerWCET13`) verbunden.

Wie in Abschnitt 4.5 erläutert werden zur Modellierung der Empfangs- und Verwaltungstätigkeit unterschiedliche Transaktionen mit periodischer Wiederholung und unterschiedlichen Startzeitpunkten verwendet. Die Aktivitätsfenster der Empfänger-Transaktionen überlappen sich nie und haben so nur Auswirkungen auf Manager- und Verarbeitungskontrollfäden. Dies entspricht dem tatsächlichen Ablaufplan.

```
Scheduling_Server (
  Type                       => Fixed_Priority,
  Name                       => Receiver,
  Server_Sched_Parameters    => (
    Type => Fixed_Priority_policy,
    The_Priority => 38
  ),
  Server_Processing_Resource => CPU
);
Scheduling_Server (
  Type                       => Fixed_Priority,
  Name                       => Manager,
  Server_Sched_Parameters    => (
    Type => Fixed_Priority_policy,
    The_Priority => 37
  ),
  Server_Processing_Resource => CPU
);
Scheduling_Server (
  Type                       => Fixed_Priority,
  Name                       => Channel1,
  Server_Sched_Parameters    => (
    Type => Fixed_Priority_policy,
    The_Priority => 27
  ),
  Server_Processing_Resource => CPU
);
```

Quellcode 5.1: Ausschnitt der MAST-Steuerdatei `controller.txt`

```
Scheduling_Server (
  Type                        => Fixed_Priority,
  Name                        => Channel2,
  Server_Sched_Parameters     => (
    Type => Fixed_Priority_policy,
    The_Priority => 22
  ),
  Server_Processing_Resource => CPU
);
Scheduling_Server (
  Type                        => Fixed_Priority,
  Name                        => Channel3,
  Server_Sched_Parameters     => (
    Type => Fixed_Priority_policy,
    The_Priority => 32
  ),
  Server_Processing_Resource => CPU
);
```

Quellcode 5.1 (Fortsetzung): Ausschnitt der MAST-Steuerdatei

Der Quellcodeabschnitt 5.3 ergänzt eine Transaktion für die Verarbeitung der Nachricht des Nachrichtenkanals 1 (Temperatursensor). Die Transaktion wird eigentlich parallel zu den Transaktionen `Receive13` und `Manage13` gestartet, die Ausführung kann aber erst nach Empfang und Zuteilung der Nachricht wirklich erfolgen. Im gleichen Empfangsfenster ist außer dem Empfang der Nachricht für Kanal 1 auch Zeit für eine Nachricht des 3. Kanals vorgesehen. Da die Priorität eines Lebenssignals höher ist[6] als die einer Temperaturnachricht, wird die Ausführung der Transaktion auch durch diese Verarbeitung verzögert[7].

MAST simuliert alle möglichen Ablaufpläne und prüft dabei die Einhaltung der geforderten Fristen. Die Ausgabe des Analysewerkzeugs zeigt den Fortschritt der Simulation und gibt am Ende das Analyseergebnis aus. Für die im Beispiel generierte Steuerdatei ist dies erfolgreich.

[6] Die Verarbeitungsfrist ist geringer.

[7] Der Übersichtlichkeit wegen wurde auf die Darstellung dieser Transaktion für Kanal 3 verzichtet. Sie unterscheidet sich außer Zeitfenstern und -fristen nicht von den anderen Beispielen.

```
*********************************************
--- HIJA Schedulability Analysis Tool (UoY) ---
*********************************************
MAST Version: 1.3.6x Parsing input file
After parsing ..................
Calculating Ceilings
---------------------------------------------
Final invocation of the analysis tool...
Printing results in file: out.txt
The system is schedulable
Final analysis status: DONE
```

Quellcode 5.2: Ausgabe des Werkzeugs bei erfolgreicher Analyse des Beispiels

Neben dem in HIJA entwickelten Verfahren für APEX-Systeme mittels einer "deadline-monotonic" Analyse sind auch andere Analyseverfahren wie zum Beispiel klassisches RMA (`classic_rm`) möglich. Sie erlauben die Bewertung des Modells für jeden einzelnen Kommunikationsknoten.

Außer der Ausgabe des Gesamtanalyseergebnisses erzeugt MAST eine Ausgabedatei `out.txt`. Hier werden Zeitabschätzungen (im günstigsten und ungünstigsten Fall) für die Verarbeitung der MAST-Steuernachrichten ausgegeben. Mit Angabe des Parameters `-c` beim Aufruf des Analysewerkzeugs können für jeden Kontrollfaden notwendige Prioritäten zur Vermeidung von Prioritätsumkehr und Verklemmungen bei der Verwendung eines Ablaufkoordinierungsverfahrens mit Prioritätshöchstmaß berechnet werden. Besonders für die applikationsspezifischen weiteren Transaktionen, die der Entwickler hinzufügt, ist diese Information interessant. Die Kontrollfäden in der Rahmenarchitektur selbst haben keine Abhängigkeiten über Datenstrukturen, die beim synchronisierten Zugriff zu Verklemmungen führen könnten.

Die Analyse des Ablaufplans nebenläufiger Kontrollfäden wird für jeden Kommunikationsknoten unterstützt. Um die Analyse auf das gesamte verteilte System zu erweitern, muss der Nachrichtenaufwand für das verwendete Netz und Kommunikationsprotokoll ebenfalls untersucht werden. Im Forschungsprojekt HIJA wurde dies ebenfalls mit MAST auf Basis eines AFDX-Netzes im Fugzeug demonstriert [57]. Die durch die Beschreibung mit Nachrichtenkanälen verfügbaren Informationen über die Nachrichtenhäufigkeit und Abschätzungen zur Nachrichtengröße im ungünstigsten Fall können auch hier genutzt werden. Vor der Ausführung kann die Verfügbarkeit einer ausreichenden Bandbreite verifiziert werden.

```
Operation (
  Type                      => simple,
  Name                      => ReceiverWCET13,
  Worst_Case_execution_Time => 50
);
Transaction (
  Type            => Regular,
  Name            => Receive13,
  External_Events => (
    (
      Type       => Periodic,
      Name       => E1122,
      Period     => 1200000000,
      Max_Jitter =>   50000000,
      Phase      =>  100000000
    )
  ),
  Internal_Events => (
    (
      Type               => Regular,
      Name               => O1122,
      Timing_Requirements => (
        Type             => Hard_Global_Deadline,
        Deadline         => 1200000000,
        Referenced_Event => E1122       )
    )
  ),
  Event_Handlers  => (
    (
      Type               => Activity,
      Input_Event        => E1122,
      Output_Event       => O1122,
      Activity_Operation => ReceiverWCET13,
      Activity_Server    => Receiver
    )
  )
);
```

Quellcode 5.3: MAST-Steuerdatei: Empfang und Verwaltung von zwei Nachrichten

```
Operation (
  Type                      => simple,
  Name                      => ManagerWCET13,
  Worst_Case_execution_Time => 50
);
Transaction (
  Type            => Regular,
  Name            => Manage13,
  External_Events => (
    (
      Type       => Periodic,
      Name       => E5617,
      Period     => 1200000000,
      Max_Jitter =>          0,
      Phase      =>  100000000
    )
  ),
  Internal_Events => (
    (
      Type                => Regular,
      Name                => O5617,
      Timing_Requirements => (
        Type             => Hard_Global_Deadline,
        Deadline         => 1200000000,
        Referenced_Event => E5617
      )
    )
  ),
  Event_Handlers  => (
    (
      Type               => Activity,
      Input_Event        => E5617,
      Output_Event       => O5617,
      Activity_Operation => ManagerWCET13,
      Activity_Server    => Manager
    )
  )
);
```

Quellcode 5.3 (Fortsetzung): MAST-Steuerdatei: Empfang und Verwaltung von zwei Nachrichten

Die vorgestellte Methodik integriert noch keine globale Analyse von Ende-zu-Ende-Kommunikationseigenschaften. Jeder einzelne Kommunikationsknoten wird unabhängig auf die Einhaltung seiner Echtzeitbedingungen geprüft. Die Beschreibung des verteilten Systems unterstützt dabei die Entkopplung zwischen Kommunikations- und sonstiger Verarbeitungslogik. Es ist möglich mittels lokaler Analyse die Ablauffähigkeit des gesamten verteilten Systems zu verifizieren.

5.4.1 Ergebnis

Auch wenn die Methodik die Unterstützung einer durchgängigen Entwicklung beschreibt, bietet der aktuelle Prototyp der Rahmenarchitektur und Werkzeugintegration noch kein einheitliches Verfahren. Bei der Evaluierung können einzelne Aspekte (Entwurf, WCETA des erzeugten Programmcodes, Analyse der Ablaufkoordinierung) noch nicht durchgängig untersucht werden. Die Bewertung der im Prototyp implementierten Funktionalität erfolgt deshalb in Schritten.

Dieser Abschnitt untersucht die Methodik bezüglich der in These 3 der Aufgabenstellung geforderten Anforderung einer statischen Analyse. Unter harten Echtzeitbedingungen muss dies entsprechend unterstützt werden. Die Evaluierung zeigt, dass die Integration eines Werkzeugs für die Analyse der Ablaufkoordinierung ist möglich. Wenn Abschätzungen für die Verarbeitung von Nachrichten im ungünstigsten Fall verfügbar sind, kann neben der Generierung von Programmcode für die Kommunikation auch ein äquivalentes Teilmodell für dessen Analyse erzeugt werden. Der Entwickler wird bei der Analyse seiner Applikation davon entlastet, ein Modell für diese Kommunikationsanteile zu entwerfen. Er kann sich wieder auf zusätzliche applikationsspezifische Funktionalität konzentrieren.

5.5 Evidenz einer durchgängigen Entwicklungsmethodik

Dieser Abschnitt verdeutlicht den offensichtlichen Nutzen einer durchgängigen Entwicklungsmethodik im Zusammenspiel mit einer objektorientierten Laufzeitumgebung, Komponenten einer Rahmenarchitektur, einer einfachen Systembeschreibung und der Implementierung von Applikationslogik durch den Entwickler.

Der Entwickler wird in unterschiedlichen Phasen der Softwareentwicklung unterstützt:

- beim Entwurf
  Mit der Abstraktion von Nachrichtenkanälen wird ein netzunabhängiger Komponentenentwurf zur Modellierung sicherheitskritischer, verteilter Echtzeitsysteme geboten;

- bei der Implementierung
  Durch eine Systembeschreibung (XML) für Kommunikation und Echtzeitbedingungen ist automatische Codegenerierung möglich;

- mit Analyse statt Tests
  Mit generierten Modellen für Ablaufpläne aller notwendigen (d.h. ebenfalls generierten) Kontrollfäden ist es möglich statische Analysen statt aufwändiger Tests am Prototyp einzusetzen.

Diese durchgängige Unterstützung bei der Entwicklung ergibt den Bezug zu objektorientierter Analyse und Design, Programmcodegenerierung und Verfahren der Programmanalyse. Die Verantwortlichkeiten innerhalb der Methodik werden klar getrennt. Abbildung 5.6 zeigt die notwendigen Funktionen einer objektorientierten Laufzeitumgebung im Zusammenspiel mit der Rahmenarchitektur und zusätzlicher Applikationslogik.

Mit der Ausrichtung auf die Entwicklung verteilter Systeme werden die Rahmenbedingungen eingeschränkt. Ein Vergleich mit dem Konzept domänenspezifischer Sprachen zeigt, dass eine durchgängige Methodik für die Entwicklung verteilter Systeme mit Echtzeitbedingungen die Entwicklung hoch-qualitativer Applikationen für sicherheits- und geschäftskritische Systeme verbessert.

Zentrale Komponente des mit der Methodik eingeführten Nachrichtendienstes ist, wie in Abbildung 3.2 auf Seite 57 dargestellt, ein logischer Nachrichtenkanal. Er wird beim Entwurf verteilter Systems genutzt, um die Kommunikation zwischen Netzknoten zu beschreiben. Seine Parameter werden bei der Erzeugung von Kontrollfäden in der Rahmenarchitektur genutzt. Ein äquivalentes Modell zur statischen Analyse des Ablaufplans auf Einhaltung angegebener Zeitfristen kann erzeugt werden.

Die durchgängige Verwendung dieser Beschreibung eines verteilten Systems bei der Entwicklung unterstützt den Entwickler. Mittels der Systembeschreibung steuert er das Zusammenspiel von Komponenten der

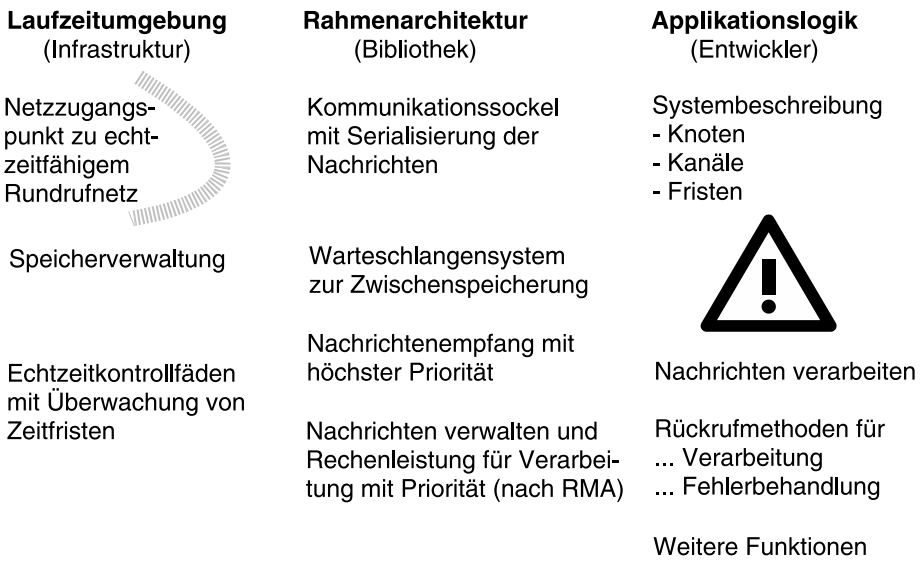

Abbildung 5.6: Zusammenspiel von Laufzeitumgebung, Rahmenarchitektur und Applikationslogik bei der vorgestellten Entwicklungsmethodik

Rahmenarchitektur, einer zugrundeliegenden objektorientierten Laufzeitumgebung und zusätzlich entwickelter Komponenten mit eigener Programmlogik.

Objektorientierte Laufzeitumgebungen für die Entwicklung sicherheits- und geschäftskritische Systeme sind neu. Durch den Einsatz einer echtzeitfähigen Laufzeitumgebung kann aber sowohl die Steuerung von Echtzeitkontrollfäden, der Zugriff auf Kommunikationsnetze, als auch die Verwaltung von Speicherbereichen und automatische Speicherbereinigung bereitgestellt werden.

Die vorgestellte Methodik unterstützt die Trennung von Zuständigkeiten zwischen Infrastruktur, Bibliothek und Entwickler. In der Auflistung werden zur Verdeutlichung Symbole der Darstellung des Beispielszenarios in Abschnitt 1.2 wiederholt. Das echtzeitfähige Netzwerk ist Teil der Infrastruktur. Es wird über die Laufzeitumgebung angesprochen und für die Verarbeitung der Nachrichten muss vom Entwickler Applikationslogik bereitgestellt werden.

Grundlage für die Entwicklung verteilter Systeme mit Echtzeitbedingungen ist die Beschreibung der verteilten Applikation durch den Entwickler. Die verbindende Rahmenarchitektur nutzt den Zugriff auf das echtzeitfähige Netz (Laufzeitumgebung) und gewährleistet die Trennung von

Empfang und Verarbeitung einer Nachricht[8]. Die Speicherverwaltung der Laufzeitumgebung wird genutzt um die Interaktion zwischen den Echtzeitkontrollfäden der Rahmenarchitektur über gemeinsamen Speicher und entsprechende Warteschlangen zu gewährleisten. Die Ausführung der Kontrollfäden wird durch die Laufzeitumgebung gesteuert.

Die objektorientierte Laufzeitumgebung mit asynchroner Kommunikationsinfrastruktur erlaubt die Integration von Softwarekomponenten auf einem relativ kleinen Nenner. Ähnlich der Entwicklung im Bereich der Geschäftssoftwareapplikationen hin zu einer diensteorientierte Architektur (vgl. Abschnitt 2.8.2) kann dieses Kommunikationsschema einer offenen Plattform die Integration unterschiedlicher Systembestandteile verbessern. Insbesondere bei eingebetteten Systemen ist dies für die Entwicklung komplexerer neuer Applikationen notwendig und noch nicht üblich (vgl. Abschnitt 2.4).

5.5.1 Harte vs. weiche Echtzeitbedingungen

Die Garantie von Echtzeitbedingungen bei der Kommunikation erfordert den Einsatz von echtzeitfähigen Netzen. Die asynchrone Nachrichtenkommunikation der Rahmenarchitektur nutzt die Garantien der Laufzeitumgebung und des Kommunikationsnetzes. Der Nachrichtenempfang ist Teil der Rahmenarchitektur und wird von der Verarbeitungslogik getrennt. Der Entwickler kann sich auf die Programmlogik ohne Kommunikationsanteile konzentrieren.

Vereinfachungen im Softwareentwicklungsprozess werden bei der Entwicklung verteilter Systeme mittels Entwurf und Analyse von Bedingungen der lokalen Komponenten erreicht.

Der Empfang mit höchster Priorität stellt sicher, dass verfügbare Nachrichten entgegengenommen werden. Die Überwachung von Ausführungsfristen periodischer oder sporadischer Verarbeitungskontrollfäden wird genutzt, um erforderliche Fristen bei Ausführung der Verarbeitungslogik der Applikation mit den empfangenen Nachrichten sicherzustellen. Beliebige a-periodische Nachrichten sind nicht mit vorhersagbarer Verarbeitungsfrist zu garantieren, da nicht planbar.

Die Umsetzung von Methodik und Rahmenarchitektur für Echtzeit-Java (vgl. Kapitel 4) mit unterschiedlichen Echtzeitbedingungen demonstriert,

[8] Nachrichtenempfang, -verwaltung und Bereitstellung von Rechenleistung für die eigentliche Verarbeitung ist Teil der Rahmenarchitektur.

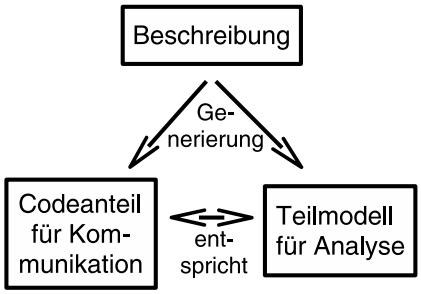

Abbildung 5.7: Automatisierte Generierung

die Machbarkeit des Entwurfs aus Kapitel 3. Für die Implementierung der Rahmenarchitektur mit harten Echtzeitbedingungen wird die Methodik erweitert. Der Entwickler wird auch bei der Analyse seines Programmcodes unterstützt.

5.5.2 Domänenspezifische Sprache

Die Methodik betrifft unter harte Echtzeitbedingungen neben dem Systementwurf auch die Ablaufanalyse eines verteilten Systems. Ähnlich einer domänenspezifischen Sprache unterstützt sie Konzepte einer verteilten Applikation und unterstützt die Generierung ausführbaren Programmcodes und eines äquivalenten Analysemodells.

Bei der Entwicklung sicherheits- und geschäftskritischer Systeme mit Echtzeitbedingungen folgt die Methodik einem deskriptiven Ansatz. Ausgehend von Beschreibungsdateien für Kommunikationsknoten werden Programmcode und Ausführungsmodelle für die Kommunikation erzeugt. Abbildung 5.7 illustriert die automatische Generierung von Programmcode und Analysemodell für den Anteil der Kommunikation. Wie in Abschnitt 3.2 als Lösungsansatz beschrieben, nutzt dieser Entwicklungsschritt eine Beschreibung des verteilten Systems, um die weitere Implementierung und Analyse der verteilten Applikation zu vereinfachen.

Die Anforderungen an Entwurfsmethoden für eingebettete Systeme (vgl. Abschnitt 2.4) können mit der umgesetzten Methodik erfüllt werden. Eine firmenübergreifende und verteilte Entwicklung ist leicht möglich. Die Funktionalität von Kommunikationsknoten wird über Komponenten (`AbstractAction`-Objekte) implementiert und die Verknüpfung der Systemknoten erfolgt mittels abgestimmter XML-Beschreibungen. Die Entwicklung von Komponenten mit klar definierten Schnittstellen (z.B. Fabrikmus-

ter für Verarbeitungslogik, `AbstractAction`) unterstützen die Wiederverwendung sowie die Entwicklung von Spezialisierungen und Varianten. WCETA wird für die Deklaration der Applikationslogik in der Rahmenarchitektur verlangt. Durch Analyse der Verarbeitungslogik auf die schlechteste anzunehmende Ausführungszeit können harte Echtzeitbedingungen in der Systembeschreibung verifiziert werden.

Weiche Echtzeitbedingungen reduzieren zwar diese Anforderungen an die Beschreibung des verteilten Systems, aber die Methodik unterstützt auch hier die automatische Generierung von verbindendem Kommunikationsprogrammcode und notwendigen Programmrahmen.

Als domänenspezifische Sprache zur Beschreibung von verteilten Systemen vereinfacht das Nachrichtenkanal-Netz die Entwicklung von Applikationen für sicherheits- und geschäftskritische verteilte Applikationen. Wie Booch und andere [19] feststellen, besteht trotz UML und MDA noch immer eine große Lücke zwischen domänenspezifischen Konzepten und hochsprachlichen Programmiertechniken. Das Nachrichtenkanal-Netz unterstützt für die Domäne der verteilten Systeme mit Echtzeitbedingungen eine Methodik, die eine Beschreibung von Applikationen mittels eines domänennahen Konzeptes von Nachrichtenkanälen mit asynchroner Nachrichtenkommunikation verwendet. Die Beschreibung ist Basis für die Generierung von plattform- und implementierungsabhängigem Programmcode. Industrielle Erfahrungen mit ähnlicher domänenspezifischer Modellierung[9] [30] versprechen eine deutliche Beschleunigung der Entwicklung.

5.5.3 Vorteile der Methodik

Die eingeführte Methodik unterstützt unterschiedliche Formen der Automatisierung. Neben der Generierung von Standard-Programmcode, wie sie Abschnitt 5.3 näher untersucht wird, untersucht Abschnitt 5.4 auch die statische Analyse von Ablaufplänen verwendeter Kontrollfäden. Neben der Implementierung und Analyse kann der Entwickler aber auch in der Entwurfsphase unterstützt werden. Der programmgesteuerte Entwurf der XML-Beschreibungen für einzelne Knoten eines verteilten Systems vereinfacht die Entwicklung. Mit dem Einsatz eines Programmier- und Entwicklungsassistenten beim Entwurf (z.B. eine Plug-In-Komponente in Eclipse) kann der Entwickler einfach eine konsistente Beschreibung der verteilten

[9] "Industrial experiences at leading-edge players like Nokia and Lucent have consistently shown development becoming 5-10 times faster."

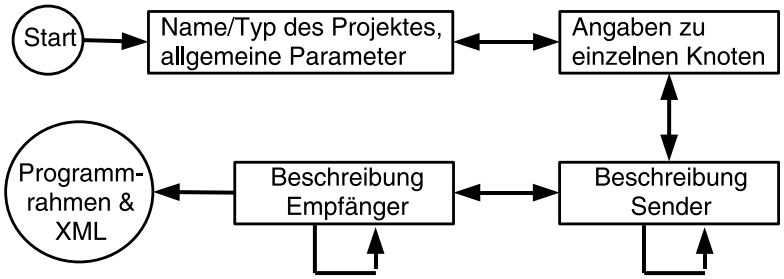

Abbildung 5.8: Geführter Entwurfsprozess

Applikation erzeugen. Wie Januar [60] feststellt, sind "gleiche" Informationen in der XML-Beschreibung für Sender und Empfänger eine Quelle für Kopieren-und-Einfügen-Fehler. Der Einsatz eines Assistenten hilft bei der Evaluierung der Entwicklungsmethodik und bestätigt den erfolgreichen Einsatz.

Abbildung 3.8 auf Seite 71 zeigt eine Abfrageseite des entwickelten Assistenten. In der dargestellten Knotenübersicht werden Angaben zu einzelnen Knoten abgefragt. Diese Informationen sind notwendig um den Entwurfsprozess zu leiten. Abbildung 5.8 illustriert diesen Prozess, wie er durch die Eclipse-Plug-In-Komponente implementiert wird.

- Mit Start beginnt die Entwicklung eines verteilten Systems.
- Im 1. Schritt werden allgemeine Parameter des neu zu erzeugenden Eclipse-Programmierprojekts abgefragt. Die verteilte Applikation unterstützt dabei die unterschiedlichen Profile der RTSJ-Laufzeitumgebung wie sie in HIJA (vgl. Abschnitt 5.1.2) identifiziert wurden.
- Eine Knotenübersicht erlaubt im nächsten Schritt die Definition der verwendeten Kommunikationsknoten. Hier werden allgemeine Informationen wie zum Beispiel die Anbindung an physikalische Kommunikationsnetze erhoben. Es ist aber auch die Unterscheidung zwischen Sender- und Empfängereigenschaften erforderlich.
- Für die erfolgreiche weitere Konfigurationserfassung werden im 3. Schritt die Senderknoten einzeln beschrieben. Teil der Senderkonfiguration ist die Beschreibungen für Nachrichtenkanäle.
- Die Konfiguration der Empfängerknoten nutzt die bisher ermittelten Beschreibungen. Für einen Empfänger können mögliche Nachrichtenkanäle abhängig von physikalischen Netzen des Knotens zur Auswahl angeboten werden.

Durch den Einsatz eines solchen Assistenten wird die Methodik für den Entwurf verteilter Systeme unterstützt. Der Assistent stellt die vollständige Konfigurationsdatenerfassung sicher und kann für die Beschreibung in jedem Knoten notwendige XML-Dateien konsistent generieren.

5.5.4 Ergebnis

Die Eignung und Erweiterung einer objektorientierte Laufzeitumgebung für die Entwicklung verteilter Systeme mit asynchroner Nachrichtenkommunikation ist Grundlage der eingeführten Methodik. Durch Trennung zwischen Infrastruktur, Bibliothek und Applikationslogik wird dem Entwickler eine definierte Methodik für den Entwurf verteilter Systeme mit harten oder weichen Echtzeitbedingungen geboten.

Die verwendete deskriptive Entwicklungsmethodik nutzt eine XML-Datei um Applikationen aus der Domäne verteilter Systeme mit Echtzeitbedingungen zu konfigurieren. Die Modellierung verteilter Applikationen mit Konzepten wie Nachrichtenkanal, Kommunikationsknoten, Sender oder Empfänger und Bearbeitungsfristen vereinfacht dabei die Entwicklung.

Das Ergebnis der in Abschnitt 5.2, 5.3 und 5.4 evaluierten drei Thesen aus Abschnitt 1.1 und weist die Evidenz einer Methodik zur Unterstützung der durchgängigen Entwicklung verteilter Systeme mit Echtzeitbedingungen nach.

5.6 Zusammenfassung

Dieses Kapitel zeigt den Nutzen der vorgestellten Methodik. Der Einsatz einer Systembeschreibung für verteilte Applikationskomponenten erlaubt die Abstraktion von konkreter Kommunikationsprogrammierung. Kontrollfäden und Zwischenspeicher für die Kommunikation können direkt erzeugt werden. Die Entwicklung von applikationspezifischer Programmlogik wird hiervon getrennt.

Die vorgestellte Methodik unterstützt den Entwickler durchgängig vom Entwurf der verteilten Applikation, über die automatische Generierung von Standard-Kommunikationsprogrammcode bis hin zur statischen Analyse des Ablaufplans mit einem äquivalenten Teil-Modell der Kommunikationskontrollfäden.

EU-Forschungsprojekte haben den Einsatz des Prototyps mit Echtzeit-Java untersucht. Mit Tests im Forschungsprojekt HIDOORS wurde der Nutzen von XSLT bei der Generierung von Standard-Programmcode bestätigt. Durch die Verwendung von Aktionsklassen mit einer Fabrikmethode für den Zugriff wird applikationsspezifischer Programmcode von Logik für die Kommunikation und Echtzeit getrennt. HIJA erforderte die Entwicklung von Prototypen der Rahmenarchitektur für harte und weiche Echtzeitbedingungen.

Weiche Echtzeitbedingungen erlauben flexible Systeme und die Implementierung einer Pastry-Bibliothek für Rundrufnetze auf Basis des Prototyps. Die Protokollkonformität wurde durch Tests und die Wiederverwendung bestehender Applikationen bei der Anpassung der dezentralen Gruppenrufsystem-Bibliothek SCRIBE belegt. Der Entwurf der P2P-Bibliothek profitiert von der deskriptiven Entwurfsmethode und die Effizienz für die Nutzung mit Rundrufnetzen wird durch den Vergleich von notwendigen Nachrichten für die SCRIBE-Kommunikation deutlich.

Mit harten Echtzeitbedingungen garantiert der RTSJ-Prototyp der Rahmenarchitektur die Generierung eines Analysemodells für jeden Kommunikationsknoten. Die Analyse dieser Modelle, die um Teile der Applikationslogik (d.h. lokale Verarbeitungslogik) erweitert werden können, erlaubt so die Analyse der Ablaufpläne eines verteilten Systems durch die Verifikation lokaler Modelle und lokaler Verarbeitungsfristen.

Kapitel 6

Zusammenfassung und Ausblick

Die in dieser Arbeit vorgestellte Methodik für die durchgängige Entwicklung verteilter Systeme mit Echtzeitbedingungen nutzt asynchrone Nachrichtenkommunikation in Rundrufnetzen für die Entwicklung verteilter Applikationen. Die endgültige Akzeptanz für eine neue Entwicklungsmethodik durch Entwickler verteilter Echtzeitsysteme kann durch eine Dissertation aber kaum erreicht werden. Diese Arbeit stellt dar, wie asynchrone Verarbeitung durch Unterstützung einer Rahmenarchitektur mit herkömmlichen synchronen Hardware-Komponenten und einer objektorientierten Laufzeitumgebung in Echtzeitfristen möglich ist.

Eine deskriptive Entwurfsmethode für Komponenten der verteilten Applikation unterstützt Echtzeitbedingungen im Entwicklungsprozess. Für die Domäne verteilter Systeme wird eine spezielle Beschreibung (domänenspezifische Sprache) vorgegeben, die die Generierung von Standard-Programmcode für die Kommunikation ermöglicht. Eine Abstraktion mit Konzepten aus der Domäne verteilter Systeme (Knoten und Nachrichtenkanäle) vereinfacht die Entwicklung verteilter Systeme und der Entwickler wird vom umfassenden Verständnis über konkrete Umsetzungen (Sprachen und Bibliotheken für Kommunikation und Nebenläufigkeit) entlastet.

Ausgehend von Programmrahmen sieht die Methodik für Systeme mit harten Echtzeitbedingungen auch die Erzeugung und Analyse eines Ablaufplans vor. Die von der eingesetzten Rahmenarchitektur verwendeten Echtzeitkontrollfäden müssen durch die genutzte Lauzeitumgebung gesteuert werden. Verfahren der Ablaufkoordinierung mit festen Prioritäten erlauben die statische Analyse vor Ausführung. Die Methodik sieht die automatische Erzeugung eines Echtzeitmodells für Kontrollfäden der

Rahmenarchitektur vor und der Entwickler kann dieses Teilmodell um applikationsspezifische Komponenten und Echtzeitkontrollfäden erweitern. Die Analyse mittels eines Werkzeugs für die Ablaufprüfung wird möglich. Der Entwickler kann sich wie bei der restlichen Implementierung auf Probleme der Applikation konzentrieren und muss keine Kommunikation oder Infrastruktur für Echtzeitfristen mit notwendigen Kontrollfäden und Warteschlangen entwerfen und implementieren.

Die aus der Motivation der Arbeit ermittelten Thesen (vgl. Abschnitt 1.1) erlauben die Bestätigung der vorgestellten durchgängigen Entwicklungsmethodik.

- Asynchrone Kommunikation vereinfacht die Entwicklung komponentenorientierter verteilter Applikationen. Durch Deklaration von Echtzeitbedingungen bei der Verarbeitung von Nachrichten werden Anforderungen für einzelne Kommunikationsknoten festgelegt. Der Einsatz einer Rahmenarchitektur unterstützt hier die Entwicklung.

- Eine beschreibende Entwurfsmethode verwendet Konzepte der Applikationsdomäne. Beim Applikationsentwurf ist deshalb weniger internes Wissen über Kommunikationsbibliotheken und geeignete Entwurfsmuster verteilter Echtzeitapplikationen notwendig. Durch Generierung von Standard-Programmcode bei der Implementierung unterstützt die Entwicklung.

- Für die Analyse werden aus den Applikationsbeschreibungen der Entwurfsphase Ablaufpläne für die Echtzeitkontrollfäden der generierten Programmteile erzeugt. Diese Analysemodelle unterstützt bei der Entwicklung die statische Prüfung der Einhaltung von harten Echtzeitbedingungen.

Alle drei Thesen werden durch die Evaluierung der vorgestellten Methodik belegt. Die Entwicklung verteilter Systeme mit Echtzeitbedingungen für Rundrufnetze wird durchgängig im gesamten Entwicklungsprozess verbessert.

Eine erste Implementierungen der notwendigen Rahmenarchitektur existieren für die Echtzeit-Java-Plattform. Diese sind Teil der Ergebnisse von zwei europäischen Forschungsprojekten für die Nutzung von objektorientierten Laufzeitumgebungen mit sicherheits- und geschäftskritischen eingebetteten Systemen.

Die Mitarbeit in beiden Forschungsprojekten hat Folgen für die Entwicklung eingebetteter verteilter Systeme. Die Nutzung asynchroner Kommunikation in zukünftigen Spezifikationen der Sprache Java für Echtzeitsysteme ist denkbar. Die mögliche Integration der deskriptiven Entwurfsmethodik in Spracherweiterungen der Java-Plattform, wie zum Beispiel Annotationen und Metadaten des Programmcodes ist offen. Ihre Nutzung würde es dem Entwickler erlauben bekannte Kommunikationseigenschaften, notwendige Echtzeitbedingungen und zusätzliche Beschreibungen der Laufzeitumgebung mit möglichen Ausführungszeitabschätzung in seinen Programmcode aufzunehmen.

Das Forschungsprojekt HIDOORS demonstrierte die äquivalente Aussagekraft von XML-Beschreibungen und Darstellungen in UML-Modellen. Die Erweiterung der hier vorgestellten Methodik um einen Modellierungphase mit UML ist möglich. Dies kann die weitere Vereinfachung des Entwurfs verteilter Systeme mit Nutzung asynchroner Kommunikationsformen fördern.

Sämtliche Prototypimplementierungen der Rahmenarchitektur, sowie die Erweiterungen für die Gleiche-zu-Gleichen-Programmbibliothek für Rundrufnetze und Erweiterung der integrierten Programmierumgebung Eclipse für die Unterstützung im Systementwurf sind als offene Produktlösungen verfügbar[1] erlauben die zukünftige Anpassung und Integration in Forschungsprojekte, sowie kommerzielle Produkte.

[1] ECN, abhängige Bbliotheken und entstandene Prototypen sind derzeit im Quellcode über SourceForge.net, einen Anbieter freier Softwareentwicklungsprojekte, online verfügbar: http://sourceforge.net/projects/ecn/

Anhang A

Glossar

Erster Eintreffzeitpunkt Zeitspanne bis zur ersten Nachricht eines Nachrichtenkanals, dieser Wert wird zur Berechnung des ersten Aktivierungszeitpunktes notwendiger Echtzeitkontrollfäden verwendet

APEX "APplication EXecutive", sichere Ausführungsumgebung für mehrere Applikationen auf einem Prozessor, die in Raum und Zeit voneinander getrennt werden und sich nicht gegenseitig beeinflussen dürfen

AUTOSAR "AUTomotive Open System ARchitectures", eine Standardisierungsbemühung der Automobilindustrie um die Spezifikation einer herstellerübergreifenden Softwareinfrastruktur

Bearbeitungsfrist Zeitspanne innerhalb der die Verarbeitung einer empfangenen Nachricht abgeschlossen sein muss

DSM "Domain-specific Modeling" Domänenspezifische Modellierung

DSL "Domain-specific Language" Domänenspezifische Sprache

ECN "EventChannelNetwork", Nachrichtenkanal-Netz API und Methodik zur durchgängigen Entwicklung verteilter Systeme mit Echtzeitbedingungen für Rundrufnetze

FSRTJ "Flexible Soft Real-Time Java", Profil in HIJA für die Weiterentwicklung von RTSJ bei Systemen mit weichen Echtzeitbedingungen

HRTJ "Hard Real-Time Java", Profil in HIJA für die Weiterentwicklung von RTSJ bei sicherheitskritischen Systemen mit harten Echtzeitbedingungen

Jitter Zittern oder maximale Verzögerung bei der Bestimmung des genauen Eintreffzeitpunktes einer Nachricht eines Nachrichtenkanals

Mindestzwischenzeit Zeitspanne, die mindestens zwischen zwei aufeinanderfolgenden Nachrichten eines sporadischen Nachrichtenkanals liegt

OSEK/VDX "Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen" und "Vehicle Distributed eXecutive" beschreibt ein herstellerübergreifendes Standardisierungsgremium für die Entwicklung von Softwareschnittstellen im Automobilbereich

Pastry Überlagerungsnetz für die Implementierung von Gleiche-zu-Gleichen-Applikationen

Pastry@ECN Implementierung der Pastry-Bibliothek auf Basis des flexiblen Nachrichtenkanal-Netzes

Periode Zeitspanne zwischen zwei aufeinanderfolgenden Nachrichten eines regelmäßigen Nachrichtenkanals

RTSJ "Real-Time Specification for Java", Echtzeiterweiterung für Java

SCRIBE Dezentrale Gruppenrufsystembibliothek auf Basis des Pastry-Protokolls

SCRIBE4Pastry@ECN SCRIBE-API auf Basis von Pastry@ECN

WCET "Worst Case Execution Time", Ausführungszeit im schlechtesten anzunehmenden Fall

WCETA "WCET Analysis", Analyse/Ermittlung der WCET ausgehend vom Programmcode oder Modell

Anhang B

DTD für XML-Beschreibungsdatei

Die nachfolgende Dokumenttypdefinition beschreibt die Syntax für XML-Beschreibungsdateien für die Java-implementierung der Rahmenarchitektur für das Nachrichtenkanal-Netz. Dieses Dateiformat beschreibt Kommunikationsknoten in verteilten Systemen mit harten oder weichen Echtzeitbedingungen.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DTD version 2.1 for EventChannelNetwork -->
<!ELEMENT node (rtsj, socket+, server?, proxy*, client?)>
<!ATTLIST node
  name CDATA #REQUIRED
  id CDATA #IMPLIED
>
<!ELEMENT rtsj (factory)>
<!ATTLIST rtsj
  profile (HRTJ | FMRTJ | FSRTJ | J2SE) #REQUIRED
>
<!ELEMENT factory (#PCDATA)>
<!ELEMENT socket (class, attributes, buffer?, (wrapper | channelref)+)>
<!ATTLIST socket
  name CDATA #REQUIRED
>
<!ELEMENT wrapper (class, channelref+)>
<!ATTLIST wrapper
  name CDATA #REQUIRED
  id CDATA #REQUIRED
>
<!ELEMENT channelref EMPTY>
```

```
<!ATTLIST channelref
  ref CDATA #REQUIRED
  id CDATA #REQUIRED
>
<!ELEMENT class (#PCDATA)>
<!ELEMENT capacity (#PCDATA)>
<!ELEMENT attributes (value*)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT buffer (#PCDATA)>
<!ELEMENT time (millis, nanos)>
<!ELEMENT millis (#PCDATA)>
<!ELEMENT nanos (#PCDATA)>
<!ELEMENT server (commit*, channel+, action?)>
<!ATTLIST server
  name CDATA #REQUIRED
>
<!ELEMENT commit EMPTY>
<!ATTLIST commit
  system-id CDATA #REQUIRED
>
<!ELEMENT proxy (channel+)>
<!ATTLIST proxy
  ref CDATA #REQUIRED
  id CDATA #REQUIRED
>
<!ELEMENT channel (description?, priority?, wrappersocket,
  start, (periodic | sporadic), jitter?, deadline, events)>
<!ATTLIST channel
  name CDATA #REQUIRED
  id CDATA #IMPLIED
  exclusive CDATA #IMPLIED
  silent CDATA #IMPLIED
>
<!ELEMENT description (#PCDATA)>
<!ELEMENT priority (#PCDATA)>
<!ELEMENT wrappersocket (#PCDATA)>
<!ELEMENT start (time)>
<!ELEMENT periodic (time)>
<!ELEMENT sporadic (time)>
<!ELEMENT jitter (time)>
<!ELEMENT deadline (time)>
<!ELEMENT events (class?, event+)>
<!ELEMENT event (size)>
<!ATTLIST event
  name CDATA #REQUIRED
  id CDATA #REQUIRED
>
<!ELEMENT size (#PCDATA)>
<!ELEMENT client (handler+, activity+)>
```

```
<!ATTLIST client
  name CDATA #REQUIRED
>
<!ELEMENT handler (action, capacity, wcet)>
<!ATTLIST handler
  name CDATA #REQUIRED
>
<!ELEMENT action (class, attributes)>
<!ELEMENT wcet (time)>
<!ELEMENT activity (capacity, overwrite, num, handlerref+)>
<!ATTLIST activity
  channel CDATA #REQUIRED
>
<!ELEMENT overwrite (#PCDATA)>
<!ELEMENT num (#PCDATA)>
<!ELEMENT handlerref EMPTY>
<!ATTLIST handlerref
  ref CDATA #REQUIRED
>
```

Anhang C

Modellgenerierung für Ablaufplan

Unter harten Echtzeitbedingungen nutzt die Methodik die Beschreibungen für Nachrichtenkanäle auch zur automatischen Generierung eines Modells für den Ablaufplan der notwendigen Echtzeitkontrollfäden für jeden Kommunikationsknoten. Dieser Anhang beschreibt den dafür verwendeten Algorithmus zur Bestimmung der notwendigen Parameter von Pseudo-Empfänger-, Pseudo-Aktivitätsmanager- und initialen Aktivitätskontrollfäden.

Das Empfängerkollektiv des Kommunikationsknotens erzeugt ein Objekt der Klasse `FeasibilityModel` und die Methode `getMastModel()` dieses Objekts erzeugt eine entsprechende Steuerungsdatei für die Analyse mit dem MAST-Werkzeug. Wesentliche Methoden und Datenstrukturen, die für die iterative Berechnung von Zeitintervallen notwendig sind, werden nachfolgend im Java-Programmcode der `FeasibilityModel`-Implementierung aufgeführt.

Eingabeparameter zur Modellerzeugung sind, für jeden Nachrichtenkanal der empfangen wird,

- Name (und Identifikator),
- Kommunikationseigenschaften (d.h. erster Eintreffzeitpunkt, Periode und Ungenauigkeit),
- Verarbeitungsfrist bzw. eine Priorität für den Aktivitätskontrollfaden und

- die Ausführungszeit der Behandlerlogik unter ungünstigsten Bedingungen.

Aus diesen Parametern werden in einem iterativen Prozess die notwendigen Empfangsintervalle und Einträge von Pseudo-Kontrollfäden für das Modell des Ablaufplans erzeugt. Tabelle 4.2 auf Seite 113 und Abbildung 4.14 auf Seite 114 zeigen wie für die Zeitintervalle des möglichen Jitters der einzelnen Nachrichtenkanäle eine Aktivitätszeit für die Pseudo-Empfängerkontrollfäden abgeleitet wird. Die Datenstruktur der Klasse Slot wird genutzt um mögliche Empfangsintervalle einer Runde als Objekte zu beschreiben und durch die Aufrufe der Methoden `initialize()` und `integrate()` wird die Liste dieser Objekte für alle möglichen Überlappungen zusammengefasst.

Außer den so berechneten Zeitintervallen werden Parameter von abhängigen Kontrollfäden für Pseudo-Aktivitätsmanger und Aktivitäten bestimmt.

```
/** Representation of a jitter time slot */
public static class Slot implements Comparable
{
  /** Start of the slot. */
  BigInteger _start;
  /** Maximal jitter = end of slot. */
  BigInteger _jitter;
  /** Old index for re-ordering depending on priority. */
  int _channelIdx;
  // ...
}
/** Global parameters */
/** Minimum start of all releases. */
private BigInteger _minStart;
/** Calculated round for all releases. */
private BigInteger _round;
/** Calculated greatest common devisor for all releases. */
private BigInteger _gcd;
/** Slots to integrate. */
private Vector _slots;

/** Initialize slots from channel parameters. */
public void initialize(
              EventChannelPeriodicParameters[] releases)
{
```

```
  BigInteger[] starts = new BigInteger[releases.length];
  BigInteger[] periods = new BigInteger[releases.length];
  BigInteger[] jitters = new BigInteger[releases.length];
  _slots = new Vector();
  for (int i = 0; i < releases.length; i++)
  {
    BigInteger start = time2int(releases[i].getStart());
    starts[i] = start;
    BigInteger period = time2int(releases[i].getPeriod());
    periods[i] = period;
    BigInteger jitter = time2int(releases[i].getJitter());
    jitters[i] = jitter;
  }
  // Calculate global parameters.
  _minStart = Euklid.getMinStart(starts);
  _round = Euklid.lcm(periods);
  _gcd = Euklid.gcd(periods);
  for (int i = 0; i < releases.length; i++)
  {
    _slots.addAll(Slot.createSlots(_minStart, _round,
                  starts[i], periods[i], jitters[i], i));
  }
}

/** Integrate overlapping slots. */
private void integrate()
{
  for (int i = 0; i < _slots.size(); i++)
  {
    Slot j = (Slot) _slots.elementAt(i);
    Vector del = new Vector();
    for (int l = i+1; l < _slots.size(); l++)
    {
      Slot k = (Slot) _slots.elementAt(l);
      sum(j, k, del);
    }
    for (int d = 0; d < del.size(); d++)
    {
      _slots.removeElement(del.elementAt(d));
    }
  }
  for (int i = 0; i < _slots.size(); i++)
  {
```

```
      Slot j = (Slot) _slots.elementAt(i);
      Vector del = new Vector();
      for (int l = i+1; l < _slots.size(); l++)
      {
        Slot k = (Slot) _slots.elementAt(l);
        sum(j, k, del);
      }
      for (int d = 0; d < del.size(); d++)
      {
        _slots.removeElement(del.elementAt(d));
      }
    }
    Collections.sort(_slots);
  }
  /** Help method to summarize Slot objects. */
  public void sum(Slot j, Slot k, Vector del)
  {
    BigInteger wcet = null;
    BigInteger jBeg = j.getStart();
    BigInteger jEnd = jBeg.add(j.getJitter());
    BigInteger kBeg = k.getStart();
    BigInteger kEnd = kBeg.add(k.getJitter());
    boolean match = true;
    if ((jBeg.compareTo(kBeg) == 0) ||
        (jBeg.compareTo(kBeg) < 0 &&
         jEnd.compareTo(kEnd) >= 0) ||
        (jBeg.compareTo(kBeg) > 0 &&
         jEnd.compareTo(kEnd) <= 0))
    {
      wcet = j.getJitter().max(k.getJitter());
    }
    else if (jBeg.compareTo(kBeg) <= 0 &&
             jEnd.compareTo(kBeg) >= 0 &&
             jEnd.compareTo(kEnd) <= 0)
    {
      wcet = kBeg.subtract(jBeg).add(k.getJitter());
    }
    else if (jBeg.compareTo(kBeg) >= 0 &&
             jBeg.compareTo(kEnd) <= 0 &&
             jEnd.compareTo(kEnd) >= 0)
    {
      wcet = jBeg.subtract(kBeg).add(j.getJitter());
    }
```

```
  else
  {
    match = false;
  }
  if (match)
  {
    BigInteger start = jBeg.min(kBeg);
    j.setStart(start);
    j.setJitter(wcet);
    del.addElement(k);
  }
}
/** Generate controller for MAST tool. */
public String getMastModel()
{
  // Output of thread parameters with aggregated slots.
}
```

Anhang D

Eigene Veröffentlichungen

[1] Marc Schanne, Dr. James J. Hunt. Remote Event Service Design. Technical Report. Deliverable D4.2 describing the HIDOORS event channel network. FZI Forschungszentrum Informatik, 2004.

[2] Marc Schanne. Real-Time Communication with a Receiver Collective, Activity Manager, and Queues. In *Proceedings of IADIS International Conference Applied Computing 2005*, Algarve, 2005.

[3] Marc Schanne. Anforderungsanalyse für asynchrone Kommunikation beim Entwurf von objektorientierten, verteilten Systemen unter Echtzeitbedingungen. Workshop zu Zuverlässigkeit in eingebetteten Systemen. Ada Deutschland Tagung 2005.

[4] Marc Schanne. Real-Time Communication with Direct Publish/Subscribe Event Service. 3rd Workshop on Java Technologies for Real-time and Embedded Systems (JTRES), OOPSLA 2005.

[5] Marc Schanne. Asynchrone Kommunikation bei objektorientierten verteilten Systemen mit Echtzeitbedingungen. In *GI Softwaretechnik-Trends*, Band 25, Heft 4, November 2005.

[6] Marc Schanne. EventChannelNetwork — Asynchronous with Real-Time. Poster presentation at Microsoft Research Summer event for PhD students, 2006.

[7] Marc Schanne. EventChannelNetwork. Tool and API Documentation. Technical Report. Deliverable D6.3.L, Version 2.5 für das

HIJA Projekt. FZI Forschungszentrum Informatik, September 2006.

[8] Marc Schanne. Integrated Development of Distributed Real-Time Applications with Asynchronous Communication. 5th Workshop on Java Technologies for Real-time and Embedded Systems (JTES), Vienna, Austria, 2007.

Literaturverzeichnis

[1] AUTOSAR. Project Website. http://www.autosar.org/.

[2] OSEK/VDX. Project Website. http://www.osek-vdx.org/.

[3] JGroups. Project Website. http://www.jgroups.org/, 2002.

[4] FreePastry. Project Website. http://freepastry.rice.edu/, November 2005.

[5] JBoss Application Server. Website. http://www.jboss.org, 2005.

[6] Pastry. Website. http://research.microsoft.com/ antr/Pastry/, Dezember 2005.

[7] AEEC. ARINC653: Avionics Application Software Standard Interface (Draft 15). Technical report, Airline Electronic Engineering Committee, 1996.

[8] aicas GmbH. The Jamaica Virtual Machine, 2001-2005. http://www.aicas.com/products/jamaica.html.

[9] ARINC. *ARINC Standard. 629 Part 1-5 - Multi-Transmitter Data Bus, Part 1-Technical Description*, 1999. http://www.arinc.com.

[10] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, pages 264–292, September 1993.

[11] Neil C. Audsley, Alan Burns, M. F. Richardson, and Andy J. Wellings. Hard Real-Time Scheduling: The Deadline-Monotonic Approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems*, 1991.

[12] Jakob Axelsson. Holistic Object-Oriented Modelling of Distributed Automotive Real-Time Control Applications. In *Proceedings of Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, 1999.

[13] Bela Ban. Adding Group Communication to Java in a Non-Intrusive Way Using the Ensemble Toolkit. Technical report, Dept. of Computer Science, Cornell University, November 1997.

[14] Bela Ban. JavaGroups - Group Communication Patterns in Java. Technical report, Dept. of Computer Science, Cornell University, July 1998.

[15] Daniel J. Barrett, Lori A. Clarke, Peri L. Tarr, and Alexander E. Wise. A framework for event-based software integration. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(4):378 – 421, October 1996.

[16] Florian Bogenberger, Bernd Müller, and Thomas Führer. Flexray. protocol overview, April 2002. Presentation on the FlexRay International Workshop.

[17] Greg Bollella. The Real-Time Specification for Java. *IEEE Computer*, 33(6):47–54, June 2000.

[18] Greg Bollella, Ben Brosgol, Peter Dibble, Steve Furr, James Gosling, David Hardin, Mark Turnbull, and Rudy Belliardi. *The Real-Time Specification for Java*, 1.0 edition, June 2000.

[19] Grady Booch, Alan Brown, Sridhar Iyengar, ames Rumbaugh, and Bran Selic. An MDA Manifesto. *MDA Journal*, pages 2–9, May 2004.

[20] Uwe Brinkschulte, A. Bechina, Florentin Picioroaga, and Etienne Schneider. Distributed Real-Time Computing for Microcontrollers - the OSA+ Approach. In *Proceedings of the International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2002)*, 2002.

[21] Uwe Brinkschulte and Theo Ungerer. *Mikrocontroller und Mikroprozessoren*. Springer, 2002. http://ipr.ira.uka.de/perso/brinks/books/microcontroller/.

[22] Daniel Brookshier, Darren Govoni, Navaneeth Krishnan, and Juan Carlos Soto. *JXTA: Jave P2P Programming*. Sams, 2002.

[23] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages. Ada 95, Real-Time Java and Real-Time POSIX*. Addison-Wesley, third edition, 2001.

[24] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication*, 20(8):1489–1499, October 2002.

[25] Alexander Christoph. *Beschreibung von Software-Entwurfstransformationen mit Graphersetzungssystemen*. PhD thesis, Universität Karlsruhe, März 2004.

[26] AEEC Airlines Electronic Engineering Committee. *Circulation Prior to Adoption Consideration Draft 4 of ARINC Project Paper 664: Aircraft Data Network, Part 7 - Avionics Full Dulex Switched Ethernet (AFDX) Network*. ARINC Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401-7465 USA, February 2005.

[27] Condor Engineering, Inc., Santa Barbara, CA 93101. *AFDX / ARINC 664 Tutorial (1500-049)*, 1.0 edition, November 2004.

[28] Peter C. Dibble. *Real-Time Java Platform Programming*. Sun Microsystems Press, 2002.

[29] Jose Maria Drake, Michael Gonzalez Harbour, Jose Javier Gutierrez, Patricia Lopez Martinez, Julio Luis Medina, and Jose Carlos Palencia. *Modeling and Analysis Suite for Real Time Applications (MAST 1.3.6). Description of the MAST Model*. Universidad de Cantabria, June 2004.

[30] DSM Forum. What is Domain-Specific Modeling? http://www.dsmforum.org/why.html, December 2006.

[31] Eclipse Foundation. Eclipse - An Open Development Platform. Project Website. http://www.eclipse.org, 2006.

[32] ECN. Event Channel Network. Project Website. http://www.eventchannelnetwork.org, 2005.

[33] Wilfried Elmenreich and Richard Ipp. Introduction to TTP/C and TTP/A. Technical report, Vienna University of Technology and TT-Tech ComputerTechnik, 2003.

[34] Andy J. Wellings et al. D1.2a. Technical report, University of York (on behalf of the HIJA project), 2005. Deliverable D1.2a describing the analysis of requirements for high-integrity systems.

[35] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[36] Wolfgang Fleisch. Validierung komponentenbasierter Software für verteilte Echtzeitsysteme. In *VDI/VDE GMA-Kongress '98*, 1998.

[37] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster*. Addison-Wesley, 1996.

[38] J.J. Gutierrez Garcia, J.C. Palencia Gutierrez, and M. Gonzalez Harbour. Schedulability Analysis of Distrbuted Hard Real-Time Systems with Multiple-Event Synchronization. In *Proceedings of 12th Euromicro Conference on Real-Time Systems*, 2000.

[39] Abdelouahed Gherbi and Ferhat Khendek. UML Profiles for Real-Time Systems and their Applications. *Journal of Object Technology*, 5(4), May-June 2006.

[40] Robert Bosch GmbH. *CAN Specification*. Robert Bosch GmbH, Postfach 50, D-7000 Stuttgart 1, 2.0 edition, September 1991.

[41] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison-Wesley, 3rd edition, 2005.

[42] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded Software Engineering: The State of the Practice. *IEEE Software*, 20(6):61–69, November/December 2003.

[43] Zonghua Gu, Shige Wang, and Kang G. Shin. Issues in Mapping from UML Real-Time Profile to OSEK API. In *Proceedings of the Workshop on Specification and Validation of UML models for Real-Time and Embedded Systems (SVERTS 2003), San Francisco, CA*, 2003.

[44] Graham Hamilton. Javabeans components architecture. Presentation at JavaOne 1998, 1998.

[45] Ulrike Hammerschall. *Verteilte Systeme und Anwendungen*. Pearson Studium, 2005.

[46] M. Gonzalez Harbour, J.J. Gutierrez Garcia, J.C. Palencia Gutierrez, and J.M Drake Moyano. MAST: Modeling and Analysis Suite for Real Time Applications. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 125–134, 2001.

[47] Michael Gonzalez Harbour, J.J. Gutierrez Garcia, J.C. Palencia Gutierrez, and J.M. Drake Moyano. On the Schedulability Analysis for Distributed Hard Real-Time Systems. In *Proceedings of the 9th Euromicro Workshop on Real Time Systems (euromicro-rts 97)*, page 136, 1997.

[48] Michael Gonzalez Harbour, Mark H. Klein, and John P. Lehoczky. Timing Analysis for Fixed Priority Scheduling of Hard Real-Time Systems. *IEEE Transactions on Software Engineering*, 1994.

[49] Bernhard Haumacher. *Plattformunabhängige Umgebung für verteilt paralleles Rechnen mit Rechnerbündeln*. PhD thesis, Universität Karlsruhe, Oktober 2005.

[50] Harald Heinecke, Klaus-Peter Schnelle, Helmut Fennel, Jürgen Bortolazzi, Lennart Lundh, Jean Leflour, Jean-Luc Mate, Kenji Nishikawa, and Thomas Scharnhorst. AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *Proceedings of the Convergence Transportation Electronics Association*, 2004.

[51] A. Hergenhan and W. Rosenstiel. Static Timing Analysis of Embedded Software on Advanced Processor Architectures. In *Proceedings of the Design, Automation and Test in Europe (DATE '00)*, page 552, 2000.

[52] HIDOORS. High Integrity Distributed Object-Oriented Realtime Systems. Project Website. http://www.hidoors.org, 2002.

[53] High-level Group on Embedded Systems. Building artemis. report. Luxembourg: Office for Official Publications of the European Communities, 2004.

[54] HIJA. High Integrity Java Applications. Project Website. http://www.hija.info, 2004.

[55] HIJA Project Partners. D3.1b - HIJA Schedulability Analysis. Technical report, The Open Group, 2005.

[56] HIJA Project Partners. D8.1 - HIJA White Paper. Technical report, The Open Group, June 2005.

[57] HIJA Project Partners. D7.4b - HIJA Methodology Handbook. Technical Report 1.4, The Open Group, September 2006.

[58] Dr. James J. Hunt, editor. *The HIDOORS Methodology. Using Java in Realtime and Embedded Systems*. aicas GmbH, Karlsruhe, Germany, 2004.

[59] Information Society Technologies. ARTIST Year 2 Roadmap. http://www.artist-embedded.org, May 2004.

[60] Mohammad Athar Januar. Bewertung von Programmier- und Entwicklungsassistenten am Beispiel eines Eclipse Plug-Ins für den Entwurf verteilter Systeme mit der EventChannelNetwork-Kommunikationsinfrastruktur, 2006.

[61] J. Kaiser and M. Mock. Implementing the Real-Time Publisher/Subscriber Model on the Controller Area Network (CAN). In *Proceedings of the Conference on Object-oriented Real-time distributed Computing*, May 1999.

[62] Mark H. Klein, Thomas Ralya, Bill Pollak, Ray Obenza, and Michael GonzÃ¡lez Harbour. *A Practitioner's Handbook for Real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems*. Kluwer Academic Publishers, 1993.

[63] Europäische Kommission. Information Society Technologies. Website. http://cordis.europa.eu/ist/, 2002-2006.

[64] Hermann Kopetz. *Real-Time Systems. Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[65] Alexander Liebrich. Gleiche-zu-Gleiche-Protokolle auf Basis asynchroner Nachrichtenkommunikation mit prototypischer Implementierung eines Pastry-Netzwerkes, 2005.

[66] Alexander Liebrich. Peer-to-Peer Protocols based on asynchronous message communication with prototype-like implementation of a Pastry network over the EventChannelNetwork. Technical report, FZI Karlsruhe, February 2006.

[67] Peter Liggesmeyer and Dieter Rombach, editors. *Software Engineering eingebetteter Systeme. Grundlagen - Methodik - Anwendungen*. Spektrum Akademischer Verlag, 2005.

[68] Frank Lippert. Ein RealTime Java UML-Profil und sein Einsatz in Analyse und Codeerzeugung. presentation, Aonix GmbH, 2003.

[69] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), January 1973.

[70] LonMark Deutschland. Interessengemeinschaft für LON-Technologie. Website. http://www.lonmark.de/, 2006.

[71] Ludwig D. J. Eggermont (ed.). Embedded Systems Roadmap 2002, March 2002.

[72] MAST. Modeling and Analysis Suite for Real-Time Applications. Project Website. http://mast.unican.es/, September 2005.

[73] Sun Microsystems. Core J2EE Patterns - Service Activator. http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceActivator.html, December 2005.

[74] Martin Naedele. A Survey of Real-Time Scheduling Tools. Technical Report 72, ETH Zurich, Computer Engineering and Networks Laboratory, CH-8092 Zurich, 1998.

[75] Nicolas Navet, Yeqiong Song, Francoise Simonot-Lion, and Cedric Wilwert. Trends in Automotive Communication Systems. In *Proceedings of the IEEE*, volume 93, June 2005.

[76] Object Management Group, Inc. *CORBA Messaging*, orbos/98-05-06 edition, May 1998.

[77] Object Management Group, Inc. *CORBA Event Service Specification*, 1.1 edition, March 2001.

[78] Object Management Group, Inc. *Real-Time CORBA v2.0: Dynamic Scheduling Specification*, ptc/01-08-34 edition, September 2001.

[79] Object Management Group, Inc. *Minimum CORBA Specification*, version 1.0, formal/02-08-01 edition, August 2002.

[80] Object Management Group, Inc. *Real-Time CORBA Specification (Static Scheduling)*, version 1.1, formal/02-08-02 edition, August 2002.

[81] Object Management Group (OMG). MDA Specifications. http://www.omg.org/mda/specs.htm, 2006.

[82] Bernd Oestereich. *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*. Oldenbourg Verlag, 4 edition, 1998.

[83] OSEK/VDX. *Communication*, 3.0.3 edition, July 2004.

[84] OSEK/VDX. *Network Management. Concept and Application Programming Interface*, 2.5.3 edition, July 2004.

[85] OSEK/VDX. *System Generation OIL: OSEK Implementation Language*, 2.5 edition, July 2004.

[86] J. Postel. *RFC 760. Internet Protocol*, January 1980.

[87] J. Postel. *RFC 768. User Datagram Protocol*, August 1980.

[88] Peter Puschner and Andy J. Wellings. A Profile for High-Integrity Real-Time Java Programs. In *Proceedings of the 4th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC)*, 2001.

[89] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing fpr large-scale peer-to-peer systems. Technical report, Microsoft Research, 2001.

[90] RTCA. Radio Technical Commission for Aeronautics. Website. http://www.rtca.org/, 2006.

[91] Jörg Schäuffele and Thomas Zurawka. *Automotive Software Engineering*. Vieweg, 2003.

[92] Douglas C. Schmidt. Reactor. An Object Behavioral Pattern for Demultiplexing and Dispatching Handles for Sychronous Events. In *Proceedings of the First Pattern Languages of Programs Conference*, 1994.

[93] Douglas C. Schmidt, David L. Levine, and Sumedh Mungee. The Design of the TAO Real-Time Object Request Broker. *Computer Communications, Elsivier Science*, 21(4), April 1998.

[94] Douglas C. Schmidt, Michael Stal, Hans Rohnert, and Frank Buschmann. *Pattern-oriented Software Architecture. Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons Ltd., April 2001.

[95] Peter Scholz. *Softwareentwicklung eingebetteter Systeme. Grundlagen, Modellierung, Qualitätssicherung*. Springer, 2005.

[96] Rainer Seck. *Skripte und Umdrucke zur Vorlesung Verteilte Systeme 2004*, chapter Kommunikation. http://cd1.ee.fhm.edu/, 2004.

[97] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[98] D. C. Sharp. Object-Oriented Real-Time Distributed Computing. In *Proceedings of 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 185–192, 2001.

[99] Fridtjof Siebert. *Hard Realtime Garbage Collection in Modern Object Oriented Programming Languages*. aicas GmbH, Mai 2002.

[100] Fridtjof Siebert, editor. *JamaicaVM 3.0 User Manual. Java Technology for Critical Embedded Systems*. aicas GmbH, 2006.

[101] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *The Journal of Real-Time Systems*, 1:27–60, 1989.

[102] Stephan Eberle und Peter Göhner. Softwareentwicklung für eingebettete Systeme mit strukturierten Komponenten. Teil 1: Komponentenorientierte Zerlegung. *Automatisierungstechnische Praxis (atp)*, (3):41–52, 2004.

[103] Stephan Eberle und Peter Göhner. Softwareentwicklung für eingebettete Systeme mit strukturierten Komponenten. Teil 2: Komponentenorientierte Modellierung und Realisierung. *Automatisierungstechnische Praxis (atp)*, (4):61–73, 2004.

[104] Sun Microsystems. *Java Remote Method Invocation (RMI). Specification*, 1999.

[105] Sun Microsystems. *Java Message Service*, 1.1 edition, April 2002.

[106] Sun Microsystems. *JavaSpaces Service Specification*, 2.0 edition, June 2003.

[107] Sun Microsystems. *Jini Specification*, 2.0 edition, June 2003.

[108] Sun Microsystems. *InfoBus 1.2 Specificaton*, 1.2 edition, February 2004.

[109] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems. Principles and Paradigms*. Prentice Hall, Inc, 2002.

[110] Daniel Tejera, Ruth Tolosa, Miguel A. de Miguel, and Alejandro Alonso. Two Alternative RMI Models for Real-Time Distributed Applications. In *Proceedings of the Eigth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, 2005.

[111] Jean Pierre Thomesse. A review of the fieldbuses. *Annual Reviews in Control*, 22:35–45, 1998.

[112] Walter F. Tichy. A catalogue of general-purpose software design patterns. In *Proceedings of Technology of Object-Oriented Languages and Systems, 1997 (TOOLS-23)*, pages 330–339, 1997.

[113] Juha-Pekka Tolvanen. Domain-specific modeling: Making code generation complete. http://www.devx.com/architect/Article/31351, April 2006.

[114] TTA Group. *TTP. Time-Trigered Protocol TTP/C. High-Level Specification Document*. TTA Group, July 2002.

[115] Joao Ventura, Fridtjof Siebert, Andy Walter, and James J. Hunt. HIDOORS - A high integrity distributed deterministic Java environment. In *Proceedings of the The Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002)*, 2002.

[116] Michael Weber. *Verteilte Systeme*. Spektrum Akad. Verlag, 1998.

[117] Andy Wellings. *Concurrent and Real-Time Programming in Java*. John Wiley & Sons, Ltd, 2004.

[118] Andy Wellings, Greg Bollella, Peter Dibble, and David Holmes. Cost Enforcement and deadline Monitoring in the Real-Time Specification for Java. In *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04)*, May 2004.

[119] Andy J. Wellings, Ray Clark, Doug Jensen, and Doug Wells. A Framework for Integrating the Real-Time Specification for Java and Java's Remote Method Invocation. In *Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 13–22, 2002.

[120] Janusz Zalewski. Object orientation vs. real-time systems – response to Alan C. Shaw's contribution. *International Journal of Time-Critical Comuting Systems*, 18:75–77, 2000.

[121] Janusz Zalewski. Real-Time Software Design Patterns. In *Proceedings of the 9th Conf. on Real-Time Systems, Ulstron, Poland*, 2002.