Forschungszentrum Informatik

an der

Universität Karlsruhe

Forschungsbereich Softwaretechnik

Dissertationsvorschlag

Methodik für den Einsatz asynchroner Kommunikation beim Entwurf von verteilten, objektorientierten Systemen mit Echtzeitanforderungen

Dezember 2005

Marc Schanne

Abteilung Software Engineering (SE)
FZI Forschungszentrum Informatik

schanne@fzi.de

Contents

Kapitel 1

Motivation und Problemstellung

Verteilte, betriebssichere Applikationen für komplexe, sicherheits- und geschäftskritische Systeme erfordern ein hohes Maß an Zuverlässigkeit. Dies soll durch den Einsatz wohldefinierter Methoden für objektorientierte Analyse und Design [5] bei Entwurf und Implementierung in diesem Einsatzgebiet unterstützt werden. Der in diesem Dissertationsvorschlag vorgestellte asynchrone Kommunikationsdienst für objektorientierte Systeme mit Echtzeitanforderungen richtet sich zwar an eingebettete Systeme, bleibt aber nicht auf solche Systeme beschränkt. Das Nachrichtenkanal-Netzwerk [53] definiert eine Methodik für die einfache Entwicklung verteilter, objektorientierter Systeme mit Echtzeitanforderungen allgemein.

Da der Anteil von Mikroprozessoren in eingebetteten Systemen die Nutzung in Arbeitsplatzrechnern und Dienstgebern weit übersteigt [13], sollen Erkenntnisse der objektorientierten Softwareentwicklung für Arbeitsplatzsysteme auf die Softwareentwicklung bei verteilten, eingebetteten Systemen übertragen werden [4, 62]. Einsatzszenarien hierfür finden sich in Automationssteuerung, Automobiltechnik und Avionik, aber auch bei Multimedia- oder Telekommunikationssystemen. Obwohl der Einsatz objektorientierter Techniken bei eingebetteten Systemen und Applikationen mit Echtzeitanforderungen nicht unumstritten ist [78, 30, 33], kann die Entwicklung und Wiederverwendung von getesteten Komponenten auch bei eingebetteten Systemen mit Echtzeitanforderungen durch Nutzung von Techniken der objektorientierte Analyse und Designs profitieren. Auf Basis von objektorientierten Rahmensystemen kann durch einen entsprechenden Entwicklungsprozess die erforderliche Zuverlässigkeit und Fehlerfreiheit dieser Systeme leichter garantiert werden.

Forschung und Analysen der letzten Jahre prognostizierten im Umfeld eingebetteter Systeme verschiedene Trends [21, 71] und die Europäische Union verfolgt deshalb eine strategische Ausrichtung in diesem Themenbereich [31]. Die Beteiligung an verschiedenen europäischen Forschungsprojekten zu objektorientierten, eingebetteten Systemen mit Echtzeitanforderungen [30, 32] erlaubt es die vorgestellte Methodik sowie Softwarearchitektur in realistischen Testszenarien zu prüfen und zu bewerten.

1.1 Einsatzgebiet

Das adressierte Einsatzgebiet von sicherheits- und geschäftskritischen Systemen ist weit und das Forschungsprojekt HIJA [32, 47] (vgl. Abschnitt 4.3.2) entwickelt EDV- und Netzwerkmodelle sowohl für sicherheitskritische Anwendungen in Automationssteuerung, Automobiltechnik und Avionik, als auch für geschäftskritische Anwendungsszenarien in Multimedia und Telekommunikation.

Die mit dieser Arbeit vorgestellte Methodik für den Einsatz von asynchroner Kommunikation bei der Vernetzung verteilter, objektorientierter Anwendungen ist Teil des Netzwerkmodells und trägt beiden vorgesehenen Profilen, für harte und flexible[1] Echtzeitanforderungen mit zwei Implementierungen für harte bzw. weiche Echtzeitanforderungen Rechnung. Das Rahmenwerk folgt in beiden Versionen dem gleichen Entwurfsmuster [54] und vereinheitlicht so die Entwicklung von Anwendungen für die gesamte Systempalette.

[1] weiche, bzw. gemischte (weiche und harte)

1.2 Thesen und Folgerungen

Die asynchrone Form der Kommunikation widerspricht im ersten Moment der Anforderung nach Echtzeitgarantien. Das Wort “asynchron” ist das gegenteilige Wort zu “synchron”, welches sich aus den altgriechische 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 Echtzeitanforderungen, die, wie in Abschnitt 2.3 näher diskutiert wird, auf zeitlicher Abstimmung basieren. Der Abschnitt 2.1 löst diesen Widersprich auf und der Begriff “asynchrone Kommunikation” wird im Folgenden für Formen von Kommunikation verwendet, bei denen das Senden und Empfangen von Daten zeitlich versetzt und ohne Blockieren der beteiligten Prozesse durch Warten auf Antwort des Empfängers erfolgt.

Basis für die weiteren Überlegungen und die Definition einer Methodik beim Einsatz asynchroner Kommunikation mit Echtzeitanforderung bilden zwei Thesen zu asynchroner Kommunikation mit Nachrichtenkanälen bei der Applikationsentwicklung.

These 1 Asynchrone Kommunikation mit Nachrichtenkanälen ermöglicht die einfache Entwicklung verteilter eingebetteter Systeme auf Basis gängiger Rundrufnetz- und Feldbus-Infrastrukturen.

Das vorgestellte Rahmenwerk ermöglicht die einfache Nutzung von Nachrichtenkommunikation nach dem Publiziere/Abonniere-Muster zur Vernetzung von verteilten, objektorientierten Komponenten in eingebetteten Systemen. Bei der Implementierung können Eigenschaften verbreiteter Netzwerktechniken (vgl. Abschnitt 2.4) bei eingebetteten Systeme (vgl. Abschnitt 2.2) ausgenutzt werden.

These 2 Bei statisch vorhersagbaren harten Echtzeitanforderungen ist es möglich, die Entwicklung von verteilten Systemen mit Nutzung eines Nachrichtenkanal-Nachrichtendienstes durch Einsatz deskriptiver Programmiermethoden zu vereinfachen.

Mit der für harte Echtzeitanforderungen notwendigen statischen, vorherigen Beschreibung von Systemen wird es möglich die Kommunikationsaspekte von der restlichen Geschäftslogik zu trennen und deskriptiv zu beschreiben. Der notwendige Programmcode für das Rahmenwerk wird dann generiert.

Obwohl beide Thesen die Möglichkeiten und Einschränkungen eingebetteter Echtzeitsysteme ausnutzen, bleibt der in Abschnitt 3 skizzierte Lösungsvorschlag und vorgestellte Prototyp allgemein einsetzbar. Mit der Vorstellung der Echtzeiterweiterung für Java (“Real-Time Specification for Java”, RTSJ) in Abschnitt 3.1, den daraus abgeleiteten Änderungsanforderungen und der Beschreibung von weiteren notwendigen Parametern der Ausführungsumgebung, wie zum Beispiel die Ablaufkoordinierung in Abschnitt 3.2, wird die Übertragung der vorgestellten Methodik auch auf andere Programmierplattformen, Sprachen und Bibliotheken vorgegeben. Die in dieser Dissertation entwickelte Version eines Rahmenwerkes basiert zwar auf Java-Technologie, aber es werden immer allgemeine Lösungen vorgestellt.

1.3 Inhaltsübersicht

Dieser Dissertationsvorschlag ist in 5 Kapitel gegliedert. Ausgehend von der Motivation und Beschreibung der Problemstellung in dieser Einführung werden in Kapitel 2 grundlegende Begriffe, existierende Ansätze und notwendige Vorbetrachtungen zum Einsatzgebiet eingeführt. Eine Lösungsskizze der oben vorgestellten Thesen bietet das Kapitel 3. Kapitel 4 vertieft die möglichen Einsatzszenarien und erläutert die weitere zeitliche Planung. In Kapitel 5 wird dieser Vorschlag mit einer Zusammenfassung der wichtigsten Ziele und einem Ausblick auf weitere mögliche wissenschaftliche Aktivitäten geschlossen.

Die Kapitel im einzelnen:

Kapitel 1 erklärt die Motivation dieser Arbeit mit der Beschreibung des Problemfeldes und daraus folgenden Inhalte. Im Abschnitt 1.2 wurden dazu zwei Thesen definiert, mit denen Vorteile für das Einsatzgebiet aus Abschnitt 1.1 gefolgert wurden.

In Kapitel 2 werden zunächst Begriffe der Kommunikation und Synchronisation sowie entsprechender Kommunikationsdienste eingeführt (vgl. Abschnitt 2.1). Diese werden in Zusammenhang mit eingebet-

teten Systemen, Echtzeitanforderungen und möglichen Netzwerksystemen eingeordnet und eine Bewertungsmetrik aus diesen Vorbetrachtungen ermittelt. Abschnitt 2.6 bietet einen exemplarischen Überblick verwandter Arbeiten aus verschiedenen Bereichen und bewertet mögliche Alternativen.

Mit Kapitel 3 wird die Lösungsskizze für das vorgestellte Einsatzgebiet näher beschrieben. Hier werden Anforderungen an die objektorientierte Ablaufumgebung, wie zum Beispiel in Abschnitt 3.2 eine prioritätsbasierte Ablaufkoordinierung, definiert und ein Entwurfsmuster (vgl. Abschnitt 3.3) integriert. Obwohl der Ansatz asynchroner Kommunikation plattformunabhängig ist, werden hier auch Anforderungen an Echtzeit-Java (RTSJ, vgl. Abschnitt 3.1) formuliert, die für die prototypische Implementierung des asynchronen Nachrichtendienstes notwendig sind. Abschnitt 3.4 beschriebt diese Implementierung und das Konzept der deskriptiven Programmbeschreibung mit Hilfe von XML wird im Abschnitt 3.5 grob skizziert.

Kapitel 4 vertieft ein paar Testszenarien für die beiden geplanten Versionen mit harten und flexiblen Echtzeitanforderungen und neben einer Beschreibung für den praktischen Einsatz und mögliche Bewertung des Nachrichtenkanal-Netzwerkes im Rahmen der europäischen Forschungsprojekte wird ein zeitlicher Plan der Dissertation selbst gegeben.

Kapitel 5 gibt eine Zusammenfassung der geplanten Methodik für den Einsatz asynchroner Kommunikation mit verteilten, objektorientierten Systemen mit Echtzeitanforderungen und gibt einen Ausblick auf mögliche weiterführende Arbeiten über diese Dissertation hinaus.

Kapitel 2

Grundlagen, Vorbetrachtungen und verwandte Arbeiten

Realistische Anwendungen operieren unter Realzeitbedingungen, sind verteilt und dynamisch [27]. Diese Anforderungen an moderne eingebettete Systeme werden in diesem Kapitel mit der Vorstellung technischer Grundlage, einer daraus abgeleiteten Diskussion erforderlicher Vorbetrachtungen und technischer Rahmenbedingungen sowie einem Überblick und einer Bewertung verwandter Arbeiten näher beschrieben. Die Anforderungsanalyse und Begriffsdefinition basiert dabei auf Überlegungen in [51].

Die erwartete Steigerung bei Heterogenität und Vernetzung von eingebetteten Systemen verlangt ein sicheres und fehlerfreies Netzwerkmodell für verteilte, sicherheits- und geschäftskritische Anwendungen. Im Gegensatz zur in Java etablierten synchronen Form der Kommunikation über entfernte Methodenaufrufe ("Remote Method Invocation", RMI) [65] und Forschungen zur Garantie von Echtzeitanforderungen mit RMI [76], soll in der vorgestellten Dissertation die Möglichkeit von asynchroner Nachrichtenkommunikation in Echtzeit untersucht und bewertet werden. Im Gegensatz zu entfernten Methodenaufrufen mit vorhersagbaren Zeitgrenzen [19, 11], die auf das strikte Dienstgeber/Dienstnehmer-Interaktionsmuster ausgerichtet sind, basiert der adressierte echtzeitfähige Nachrichtendienst auf einem Nachrichten-Benachrichtigungs-Muster [49].

Basis für einen Kommunikationsdienst mit Nachrichten für eingebettete Systeme bildet im Folgenden die Klärung von Begriffen und Techniken im Rahmen von Kommunikation allgemein. Durch Betrachtung der besonderen Rahmenbedingungen im Umfeld eingebetteter Systeme zur Erfüllung von Echtzeitanforderungen wird die Notwendigkeit an zugrundeliegende Kommunikationsnetze und Ablaufumgebungen definiert und die so herausgearbeiteten Anforderungen werden zur Klassifizierung und Bewertung verwandter Arbeiten verwendet.

2.1 Kommunikation

Basis für verteilte Systeme ist die Verfügbarkeit einer Interprozesskommunikation [70]. Nebenläufige und vernetzte Dienste erlauben Systeme mit gesteigerter Performanz, mehr Zuverlässigkeit, besserer Skalierbarkeit und Kosteneffizienz [59]. Diese Vorteile im Bereich der Standard- und Unternehmenssoftware-Anwendungen sind auch für eingebettete Systeme nutzbar. Netzwerke von modernen Sensoren, Aktoren und Prozessoren werden eingesetzt um sicherheits- und geschäftskritische Systeme zu unterstützen. Für ihre Entwicklung sind unterschiedliche Formen der Vernetzung sowie Kommunikations- und Synchronisationsmechanismen denkbar. Diskutiert werden in diesem Abschnitt mögliche Prinzipien und Paradigmen bei der Kommunikation und Synchronisation zwischen unterschiedlichen vernetzten Komponenten und allgemein übliche Fachbegriffe in diesem Kontext werden eingeführt.

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

Tabelle 2.1: Kommunikation und Synchronisation

2.1.1 Kommunikationsformen

Eine verteilte Umgebung ist auf ein Netzwerk, das seinen Komponenten und Prozessoren eine Bandbreite zur Kommunikation anbietet, angewiesen. Diese Kapazität wird auf niederer Ebene für den Austausch von einfachen Nachrichten genutzt. In Systemen mit harten Echtzeitanforderungen (vgl. dazu Abschnitt 2.3) muss diese Kapazität ausreichend sein um eine rechtzeitige Auslieferung zu garantieren. Protokolle, strukturiert in Schichten[1], erlauben die Kommunikation auf Anwendungsschicht. Der Anwendungsnutzen des Kommunikationsprotokolls unterscheidet zwei grundlegende Formen: **verbindungsorientierte** und **verbindungslose** Kommunikation.

Verbindungsorientiert beschreibt eine Form von Kommunikation, die auf vorlaufenden Protokollen aufbaut, die eine Ende-zu-Ende-Verbindung etabliert, bevor irgendwelche Daten gesendet werden.

Verbindungslose Kommunikation erfordert die Möglichkeit für den Versand einer Nachricht zwischen zwei Kommunikationsendpunkten ohne vorherige Maßnahmen. Der Begriff **nachrichtenorientiert** wird oft in äquivalenter Form verwendet.

Spezielle Formen von Kommunikation, wie z.B. **transaktionsorientierte** Kommunikation, ergänzen zusätzliche Protokolllogik zum einfachen Nachrichenaustausch und fassen mehrere Nachrichten in einer Transaktion zusammen.

Verbindungsorientierte Kommunikationsdienste werden auch als **zuverlässige** Netzwerkdienste bezeichnet, aber eine **verlust-** und **verfälschungsfreie** Kommunikation kann mit entsprechendem Prüfaufwand auch von nachrichtenorientierten Protokollen erbracht werden. Nachrichtenorientierte Kommunikation kann so unterschiedliche Zuverlässigkeitsanforderungen, mit Semantik von **“best effort”**, über **“at-most-once”** und **“at-least-once”** bis hin zu **“exactly-once”** [22] garantieren.

In einer objektorientierten Anwendungsumgebung, wie z.B. Java, gibt es für beide Kommunikationsformen konkrete Programmbibliotheken: Entfernter Methodenaufruf (“Remote Method Invocation”, RMI, vgl. Abschnitt 2.6.1) [65] und Java Nachrichtendienst (“Java Message Service”, JMS, vgl. Abschnitt 2.6.2) [66]. Diese Standard-Implementierungen erfüllen keinerlei Echtzeitanforderungen und sind nicht für eingebettete Systeme mit begrenzten Ressourcen geeignet.

2.1.2 Synchronisationsmechanismen

Synchronisation ist eine orthogonale Charaktereigenschaft, die direkt mit Kommunikation verbunden ist. Mit ihr wird beschrieben, wie entfernt ablaufende Prozesse sich untereinander verhalten. Synchronisation generell ist ein bekanntes Konzept bei der gemeinsamen Benutzung von Ressourcen oder Einheiten. Die Benutzung von Algorithmen für gegenseitigen Ausschluss charakterisieren so Kommunikation bezüglich Zeit und Datenzugriff.

Synchrone Kommunikation verlangt, dass Prozesse aufeinander “warten” und deshalb in ihrer sonstigen Ausführung blockiert sind. Der Einsatz von asynchroner Kommunikation erlaubt eine nebenläufige Ausführung unabhängig von der Zeit und dem Zustand anderer Kommunikationspartner. Dies erfordert aber beim Austausch der Nachrichten Pufferkapazität.

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

[1] z.B. vergleichbar mit dem Open Systems Interconnections (ISO/OSI) Referenzmodell

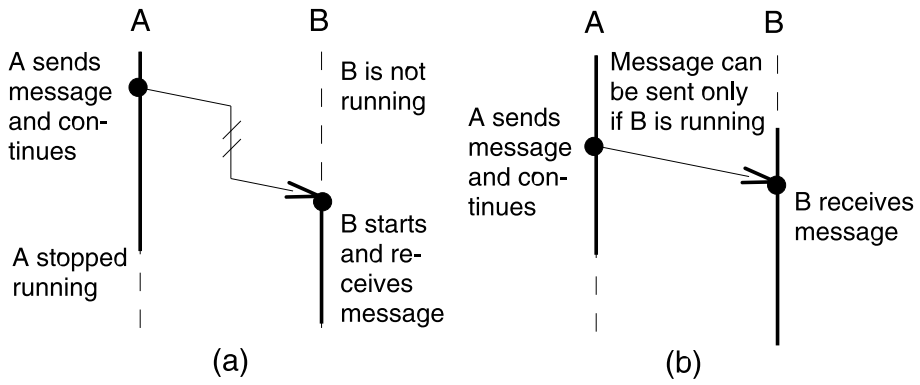

Abbildung 2.1: Persistente und transiente Kommunikation

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

Tabelle 2.2: Kommunikationsmodelle [60]

2.1.3 Persistenz vs. Transienz

Selbst asynchrone Kommunikation mit Nachrichten erfordert Synchronisation beim Nachrichtenaustausch. Zusammenhängend damit kann die Persistenzstrategie beim Umgang mit Nachrichten unterschieden werden. Abbildung 2.1 illustriert die verschiedenen Möglichkeiten.

Persistente Kommunikation erfordert, dass gesendete Nachrichten durch eine Kommunikationsinfrastruktur bis zu ihrer Auslieferung an den Empfänger gespeichert werden (a). So ist es möglich, dass der Empfänger gerade nicht aktiv empfängt, sondern andere Aufgaben bearbeitet. Hier muss die Nachricht so lange zwischengespeichert werden. Im Gegensatz dazu bietet **transiente** Kommunikation keinerlei Zwischenspeicherung (b).

Beide Varianten sind geeignet, um eine asynchrone nachrichtenbasierte Kommunikation mit dem Publiziere/Abonniere-Muster (vgl. den folgenden Abschnitt 2.1.4) bereitzustellen.

Zur besseren Beschreibung der Kommunikation von verteilten Systemen auf Basis von Nachrichtenaustausch ohne gemeinsam benutzten Speicher lassen sich so die in Tabelle 2.2 vorgestellten Kriterien zur Beschreibung unterscheiden.

Bei direkter Adressierung spricht der Sender den Empfäger direkt und im allgemeinen hartcodiert an. Im Unterschied dazu wird bei indirekter Adressierung über eine Empfangsstelle (Mailbox) kommuniziert.

Blockierende Kommunikation führt automatisch zu einer Synchronisation der Kommunikationspartner, ein Sender wartet, bis die Nachricht über das Netz losgeschickt worden ist und evtl. eine Empfangsbestätigung vom Empfänger zurückgekommen ist. Der korrespondierende Empfänger blockiert, bis eine Nachricht angekommen ist.

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, so muss blockiert werden, was dem asynchronen Konzept widerspricht, oder Daten gehen verloren.

Als Kommunikationsformen lassen sich meldungsorientierte Einwegnachrichten[2] mit höchstens einer

[2] Idealbetrachtung!

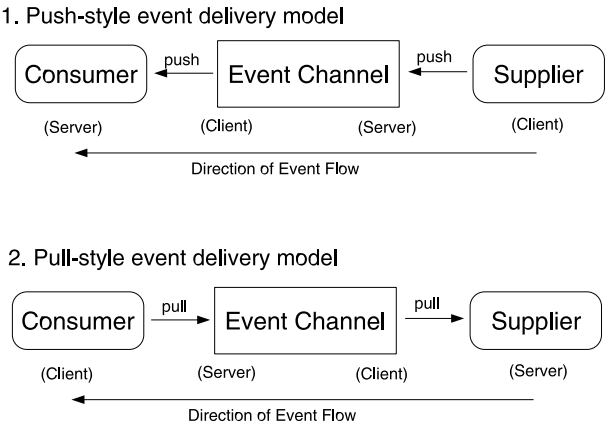

Abbildung 2.2: Push- und Pull-Nachrichtenübertragungsmodelle

Empfangsbestätigung und auftragsorientierte Hin- und damit verbundene Rücknachrichten unterscheiden. Diese Formen lassen sich mit den vorgestellten anderen Kriterien kombinieren.

Im folgenden wird insbesondere die asynchrone Option bei der Blockierung der einzelnen Prozesse mit indirekter Adressierung und gepuffertem Nachrichtenaustausch betrachtet. Asynchrone Kommunikation erlaubt die (zeitliche) Entkopplung von Sender und Empfänger, Parallelarbeit wird unterstützt.

2.1.4 Publiziere/Abonniere-Kommunikation

Die Eine-zu-viele-Kommunikation des Publiziere/Abonniere-Nachrichtendienstes lässt sich in zwei Schritte unterteilen. Bevor der eigentliche Nachrichtenaustausch stattfindet, müssen sich die Komponenten im Netzwerk untereinander oder bei einem Dienst registrieren. Ein Publizist erzeugt Nachrichten, die für den oder die Abonnenten interessant sind. Oft unterstützt er durch eine öffentliche Ankündigung die Registrierung der Abonnenten. Das Publiziere/Abonniere-Kommunikationsmuster erlaubt so die Entkopplung von Anbietern und Konsumenten in **Raum**, **Zeit** und **Synchronisation** [23].

- Die Entkopplung im Raum wird durch Einführung eines Dienstes für die Zustellung einer Nachricht ermöglicht. Der Anbieter publiziert seine Nachrichten über diesen Nachrichtendienst und der Konsument 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.

- Publiziere/Abonniere-Kommunikation erlaubt die Entkopplung bezüglich Synchronisation und Gleichlauf der Kommunikationspartner, sie können nebenläufig, unabhängig von der Kommunikation und ohne Blockierung andere Aktionen ausführen. Über die Verfügbarkeit einer Nachricht werden sie z.B. asynchron über eine Rückrufmethode informiert.

Die OMG (“Object Management Group”) führt in ihrer Spezifikation zum CORBA Nachrichtendienst [45] (vgl. auch Abschnitt 2.6.6) zwei Basismodelle für die Nachrichtenübertragung bei Publiziere/Abonniere-Systemen ein: **“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.

Das “Push”-Modell erfordert die vorherige Registrierung des Empfängers beim Erzeuger von interessanten Nachrichten. Wenn eine Nachricht gesendet werden soll, wird sie in die Empfangsschnittstelle des

Abonnenten “geschoben”. Das “Pull”-Modell agiert umgekehrt, nach der Registrierung bei interessanten Publizisten “bezieht” der Abonnent die Nachrichten aktiv.

Eine weitere Unterscheidung wird in [23] identifiziert, hier werden drei Konzepte für die Abonnierung unterschieden: **themenbasierte**, **inhaltsbasierte** oder **typbasierte** Publiziere/Abonniere-Systeme sind denkbar.

1. Themenbasierte Publiziere/Abonniere-Kommunikation stellt die früheste Form von Publiziere/Abonniere-Systemen dar und basiert auf der Notation eines Themas oder eines Titels. Themen sind analog zur Bildung von Gruppen und mit der Registrierung des Empfängers für ein Thema erhält er alle Nachrichten dieser Gruppe.

2. Publiziere/Abonniere-Kommunikation, basierend auf Inhalten, selektiert die zu empfangenden Nachrichten abhängig vom tatsächlichen Inhalt und nicht nur von externen Eigenschaften.

3. Bei typbasierter Auswahl wird die Auswahl nicht nur abhängig vom Inhalt, sondern auch von der Struktur bestimmt. Dieses Schema ersetzt oft die themenbasierte Auswahl mit einem Filter abhängig vom Typ der Nachrichten.

Für den Einsatz in verteilten eingebetteten Systemen eignet sich die Verwendung eines themenbasierten, direkten und asynchronen Publiziere/Abonniere-Nachrichtendienstes. Die Form sicherheits- und geschäftskritischer Systeme verlangt geringen Kommunikationsaufwand bei Zustellung und Selektion von Nachrichten und asynchrone Interaktion erlaubt die Nutzung aller verfügbarer Aktoren und Sensoren, die nicht durch übermäßigen zusätzlichen Protokollaufwand belastet werden können.

2.2 Eingebettete Systeme

Eingebettete Systeme 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. Obwohl sie oft auf der gleichen Hardware wie Arbeitsplatzrechner basieren, unterliegen sie jedoch meist stark einschränkenden Randbedingungen: Minimale Kosten und damit geringem Platz-, Energie- und Speicherverbrauch [77].

Die Kommunikation im Bereich der eingebetteten Systeme erfordert eine besonders auf die Leistungsbegrenzung und eingeschränkte Kommunikationskapazität ausgerichtete Kommunikationsinfrastrukturen. 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 und [38] definiert Grundlagen, die bei der Bewertung von Softwareentwicklungsmethodiken für die im Folgenden vorgestellten Lösungen genutzt werden sollen.

2.3 Echtzeit

Echtzeitanforderungen werden im Rahmen von Rechen- bzw. Verteilungsmodellen für verteilte, eingebettete Systeme durch Einhaltung von Zeitgrenzen garantiert. Aus diesen Anforderungen an Rechtzeitigkeit und Erfüllung der Kommunikation in gegebenen Zeitgrenzen lässt sich nach [35] eine Klassifizierung von Echtzeit mit **harten**, **weichen** und **festen** Anforderungen ableiten. Diese Einteilung definiert sich über die schlechteste anzunehmende und die durchschnittliche Ausführungszeit.

Harte Echtzeit: Alle Antworten mit harten Zeitschranken müssen innerhalb des verfügbaren Zeitfensters ausgeführt sein. Wenn das System eine Zeitgrenze nicht einhält, ist das Gesamtsystem unvorschriftsmäßig. Bei Systemen mit sicherheitskritischen Anforderungen kann dies zu Gefahr für Leib und Leben, Schädigung der Umgebung oder zu signifikanten finanziellen Verlusten führen.

Weiche Echtzeit: Weiche Zeitanforderungen werden durch die Forderung nach einer durchschnittlichen Antwortzeit charakterisiert, verfehlte Zeitschranken äußern sich in Mängeln der allgemeinen Dienstequalität und nicht in einem vollständigen Systemausfall.

Feste Echtzeit: Die 3. Klasse der festen Zeitanforderungen beschreibt ebenfalls eine erforderte durchschnittliche Antwortzeit, die aber zusätzlich in harten Zeitfenstern zu erbringen ist.

Objektorientierte Softwareentwicklung mit Java wurde 2000 durch die "Real-Time Specification for Java" (RTSJ) für den Einsatz in Echtzeitanwendungen erweitert [8]. Der bei der Prototypentwicklung (vgl. Abschnitt 3.4) verwendete Lösungsansatz basiert auf RTSJ, sowie einiger im Rahmen des HIJA Forschungsprojektes (vgl. Abschnitt 4.3.2) untersuchten Einschränkungen dieser Spezifikation. Diese Ergebnisse sind Teil der vorgestellten Anforderungen an die objektorientierte Laufzeitumgebung, wie sie in Abschnitt 3.1 beschrieben wird und sind Grundlage für die beschriebenen Methodik bei der Nutzung eines echtzeitigen Nachrichtendienstes.

Kommunikation mit vorgegebenen Zeitgrenzen ermöglicht erst die Entwicklung verteilter, sicherheits- und geschäftskritischer Systeme. Mit Einsatz von Java als objektorientierter Programmiersprache kann man die Voraussetzungen für einen modularen, durch Kapselung fehlertoleranten und mit redundanten Komponenten skalierbaren Systementwurf ermöglichen. Der Einsatz eines Publiziere/Abonniere-Nachrichtendienstes, der asynchrone Interaktion in vorhersagbaren Zeitgrenzen bietet, unterstützt dieses Entwurfsverfahren eingebettete Systeme und die dort existierende Vernetzung.

2.4 Netzwerke

Das Netzwerk ist die Grundlage eines verteilten Systems. Es unterstützt die Kommunikation der einzelnen Knoten und ist deshalb eine kritische Ressource, da der Verlust von Kommunikation den Verlust globaler Dienste des Systems bewirkt [36]. Neben einer physikalischen Verbindung besteht das Netzwerk aus Netzwerkzugangspunkten, über die einzelne Knoten mit einem geschichteten Protokoll zuverlässig, sicher, effizienter und (bei Echtzeit-Netzwerken) mit Einhaltung von Zeitgrenzen kommunizieren können. Dieser Abschnitt stellt eine übliche Infrastruktur und Vernetzung bei eingebetteten Systemen vor.

Ein Beispiel für eine Protokollschichtung ist das ISO/OSI Referenzmodell, das mit 7 Schichten von physikalischer bis Anwendungsschicht arbeitet, aber im Umfeld der eingebetteten Systeme zu komplex ist und zu viel Overhead 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 durch 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 (Kabel).

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

Avionik: ARINC629 ARINC629 ist ein Datenbus-Standard (Netzwerkprotokoll), der von Boeing für die Verkabelung und Steuerung in den Boeing 777 entwickelt wurde.

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

Das Controller Area Network (CAN [28]) wurde von der Firma Bosch für die Kommunikation in Industrieanlagen und elektronischen Steurungsanlagen entwickelt und findet heute verbreitet im Automobil

Einsatz. Durch sein nachrichtenbasiertes Kommunikationsmodell unterstützt CAN aber von sich aus nur weiche Echtzeitanforderungen. 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 Echtzeitanforderungen genutzt werden [34].

Durch ein zeitbasiertes Multiplexverfahren und entsprechende Ressourcenzuteilung bietet eine zeitgesteuerte Architektur ("Time Triggered Architecture" [73]) weiterführende Garantien, die für die Bereitstellung von harten Echtzeitanforderungen notwendig sind.

Automation: ProfiBus, Ethernet/IP In der Automation werden schon lange speziell entwickelte Feldbusse eingesetzt [72]. Neben einfacher Verkabelung gibt es hier aber auch einen Trend hin zur Standardisierung auf Basis von Internettechnologie wie TCP/IP.

Netzwerksysteme wie z.B. das Avionics Full Duplex Switched Ethernet (AFDX/ARINC 664 [18]) Protokoll für deterministisches Ethernet oder andere Punkt-zu-Punkt-Verbindungsnetzwerke werden hier nicht weiter betrachtet, da der Mehraufwand in der Netzinfrastruktur komplexere Kommunikationssysteme wie z.B. Echtzeit-RMI erlaubt und den vorgestellten Ansatz asynchroner Nachrichtenkommunikation zwar nicht behindert, aber auch nicht fordert. Die in Kapitel 3 vorgestellte Lösungsskizze für die Methodik beim Einsatz von asynchroner Kommunikation unter Echtzeitanforderungen soll verteilte Applikationen für sicherheitskritische Systeme mit geringeren Investitionen ermöglichen.

2.5 Vorteile asynchroner Kommunikation im Echtzeiteinsatz

Obwohl eine asynchrone und zeitentkoppelte Kommunikation im ersten Moment den deterministischen und statisch verifizierbaren Anforderungen in sicherheitskritischen Echtzeitsystemen widerspricht, bietet die Publiziere/Abonniere-Kommunikation verschiedene Vorteile für verteilte Systeme und die Anpassung an Echtzeitsysteme versprechen lohnenswerte Ziele:

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

Entkopplung von Publizierer und Abonnent verspricht größere Flexibilität und eine einfachere Wiederverwendung von Programmcode durch die Entkopplung in Adressierung, Zeit und Synchronisation;

Viele-zu-viele-Kommunikation erlaubt die Ausnutzung von rundruforientierten Feldbussystemen im Umfeld von eingebetteten Systemen;

Skalibarkeit: Einfache erweiterbare asynchrone Kommunikation ohne blockierende Anfrage/Antwort Interaktionen skaliert gut von einfachen bis großen Systemen;

Einfache Programmierung von Steuerungsgeräten in der Automation: Verwandtes Kommunikationsmodell: Informationen müssen nicht gelesen werden, kein Wissen über interne Speicherabbildungen, Datenbankstrukturen oder unbekannte Steuerungsgeräte ist notwendig;

Effiziente Nutzung von Netzwerkbandbreite: Kein Netzwerkverkehr für Anforderungen, Nutzung direkter ereignisgesteuerter Kommunikation;

Robustes Applikationsdesign: Unterstützung für Ausfallsicherheit und Migration im Fehlerfall;

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

Ausrichtung auf Echtzeitanforderungen: Geeignet im Einsatz mit Signalströmen, Statusaktualisierungen, ereignisgesteuerten Befehlen in Echtzeitsystemen.

Um diese Vorteile asynchroner Kommunikation für das Anwendungsgebiet deterministischer, echtzeitfähiger Systeme zu erschließen, müssen vorhersagbare Grenzen auf Sender- und Empfängerseite in die Kommunikation eingebaut werden. Die Garantien, die die hier vorgestellte Nachrichtenkanalkommunikation bietet, werden im Abschnitt 3.3 theoretisch mit einem Entwurfsmuster und im Abschnitt 3.2 zu Ablaufkoordinierung mit ein paar praktischen Rahmenvorgaben vorgestellt und diskutiert. Hier wird auch die Voraussetzung für die Nutzung der vorgestellten Methodik mit anderen Ablaufumgebungen beschrieben.

Die in den folgenden Unterabschnitten vorgestellten Produkte, Forschungsprojekte und Programmierschnittstellen bieten einen Überblick über aktuelle konkurrierende Entwicklungen im Umfeld verteilter Kommunikation. Sie sind in erster Linie anhand ihrer Bereitstellung von asynchronen Kommunikationscharakteristika für unterschiedliche Anwendungsgebiete zusammengestellt. Insbesondere ihre Eignung für den Einsatz in echtzeitkritischen Systemen wird dabei verglichen und bewertet.

2.6 Verwandte Arbeiten

Im Umfeld verteilter Systeme sind recht unterschiedliche Kommunikationsinfrastrukturen entwickelt worden. Die folgenden Unterabschnitte stellen eine Auswahl vor. Ihre Eignung bezüglich der Echtzeitanforderungen und dem Einsatz in eingebetteten Systemen wird hierbei besonders bewertet. Die vorgestellten Arbeiten basieren meist auf Java Technologie, da der vorgestellte Ansatz selbst mit einem Prototypen basierend auf Echtzeit-Java (vgl. Abschnitt 3.1) dargestellt wird.

Verwandte Arbeiten für die Entwicklung verteilter, sicherheits- oder geschäftskritischer Systeme mit dem Fokus auf echtzeitfähige Kommunikation sind von Seiten verteilter und plattformübergreifender Programmiersysteme wie CORBA (vgl. dazu Abschnitt 2.6.6), aber auch beim Hardware-nahen Entwurf von Mikrokern-basierten Echtzeitbetriebssystemen zu identifizieren. Die Anforderungsanalyse [51] beschreibt und bewertet beide Ansätze unter Berücksichtigung der gegebenen Netzwerkinfrastruktur bei eingebetteten Systemen und einer objektorientierten Entwicklungsmethodik.

2.6.1 Synchrone Anfrage/Antwort Architektur

In Java wird bei synchroner Kommunikation mit entfernten Methodenaufrufen seit der Version 1.1 der Einsatz von RMI (“Remote Method Invocation”) [65] angeboten. RT-RMI (“Real-Time” RMI) für echtzeitfähige Systementwicklung ist derzeit Teil der Forschung [76]. RMI konkurriert dabei direkt mit dem plattformübergreifenden Kommunikationsmodell von CORBA (Common Object Request Broker Architecture), auf dessen Basis die Kommunikaton zwischen Anwendungen verschiedener Sprachen und Plattformen ermöglicht wird. Für die Echtzeitvariante von CORBA vergleiche Abschnitt 2.6.6.

2.6.2 Java Messaging Service (JMS)

Echte asynchrone Kommunikation im Java Umfeld wurde 2002 mit der Definition von JMS (“Java Messaging Service”) [66] eingeführt. Dieser skalierbare, asynchrone Nachrichtendienst bietet sowohl themenbasierte Publiziere/Abonniere-Kommunikation als auch inhaltsbezogene Filterung der übertragenen Nachrichten an. Die Architektur selbst ist zentralisiert.

2.6.3 Jini, JavaSpaces, JXTA

Für die Implementierung von dezentralen Ad-Hoc-Netzwerken zwischen unterschiedlichsten Internetfähigen Endgeräten hat Sun Microsystems ein auf RMI basierendes Protokoll für Java-Anwendungen: Jini [68]. Durch Einsatz von Zwischenspeichern (JavaSpaces [67]) 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 entwickeln.

Ausgehend von dienstorientierter Anwendungsentwicklung 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-Netzwerke ausgerichtete Ansatz zur Diensteinstallation basiert zwar auf einem Netzwerkansatz 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 weniger auf Java ausgerichtete Variante[3] für die Entwicklung von P2P Anwendungen ist seit 2001 die JXTA Plattform [14]. Basierend auf 6 zugrundliegenden Protokollen, wie z.B. 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.

2.6.4 Java InfoBus

Zur lockeren 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 [69] definiert einen Themen-bezogenen 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.

2.6.5 JavaGroups

JavaGroups [6, 7] definieren ebenfalls ein gruppenbasiertes und anonymes Kommunikationsprotokoll. Dieses Open-Source Projekt wird z.B. bei der netzwerkweiten Kopplung von Enterprise-Java-Containern des JBoss Projektes [1] zur Bildung von Rechnerbündeln ("cluster") eingesetzt.

2.6.6 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 Echtzeitanforderungen anzupassen, ist deshalb die Echtzeiterweiterung (RT-CORBA) definiert worden. RT-CORBA beschreibt auf Basis der synchronen CORBA-Kommunikationsplattform auch asynchrone Nachrichteninteraktion. Neben dem relativ hohen Protokoll-Overhead bei dieser Kommunikation ist besonders die TCP/IP-Orientierung der CORBA-Plattform ein Hindernis für die Nutzung im Umfeld sicherheits- oder geschäftskritischer Systeme, die oft Netzwerke mit Viele-zu-Viele-Kommunikation verwenden [51]. Ein weiterer Ansatz der OMG, um mit beschränkten Ressourcen und typischen Problemen eingebetteter Systeme umzugehen, stellt die MinimumCORBA Spezifikation dar. Beide Spezifikationen sind zwar orthogonale Weiterentwicklungen der CORBA Rahmenarchitektur, aber die Integration der unterschiedlichen Anforderungen für eine CORBA-Implementierung auf eingebetteten Systemen mit beschränkten Ressourcen und Echtzeitanforderungen bleibt unklar.

Mit der Einführung von Echtzeitanforderungen im CORBA Softwareentwurf (RT-CORBA [44]) hat die OMG (Obejct Management Group) zwei interessante asynchrone Kommunikationsprotokolle auf Basis der etablierten synchronen Infrastruktur definiert. Auf Basis der synchronen Kommunikation kann mit RT-CORBA beziehungsweise mit unterschiedlichen ORB Implementierungen (Beispiel: TAO [58]) so auf einen Nachrichten- und Benachrichtungsdienst [43, 42] zugegriffen werden. Dieser Ansatz und die TCP/IP-Orientierung von CORBA müssen beim Einsatz mit eingebetteten Systemen kritisch betrachtet werden.

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

2.6.7 OSA+

Konzepte, die von Hardware-Seite die Erfordernisse von verteilten Echtzeitsystemen angehen, sind ebenfalls in der aktuellen Entwicklung. Mit der “Open System Architecture - Platform for Universal Services” (OSA+) [12] 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 6 Funktionen müsste eine objektorientierte Schnittstelle zur Nutzung in hochsprachlich entwickelten, sicherheits- oder geschäftskritschen Systemen aber erst noch implementiert werden.

Kapitel 3

Lösungsskizze: Nachrichtenkommunikation in Echtzeit

Die in [54] vorgeschlagene Architektur mit Empfänger-Kollektiv, einem Kontrollfaden zur Steuerung von Verarbeitungskontrollfäden und Warteschlangen ermöglicht asynchrone Kommunikation mit Nachrichten auf Basis von synchronen Ein-/Ausgabeschnittstellen auf Ein-Prozessorsystemen in harten Zeitgrenzen. Die notwendige Struktur mit den wesentlichen Komponenten sowie deren Interaktion wird in Abschnitt 3.3 nochmal grob dargestellt und neben Anforderungen an die erforderliche objektorientierte und echtzeitfähige Laufzeitumgebung wird auch die prototypische Implementierung in Abschnitt 3.4 eingeführt. Basierend auf einer Beschreibung von Echtzeit-Java[1] im Abschnitt 3.1 beschreiben die Unterabschnitte 3.3.2 bis 3.3.7 diese Struktur, sowie die Interaktion der jeweiligen Komponenten. In Abschnitt 3.2 wird die Interaktion in die Ablaufkoordinierung der Laufzeitumgebung integriert und eine Methodik mit Anforderungen an die verwendete Laufzeitumgebung bestimmt. Abschließend definiert Abschnitt 3.5 eine vereinfachte Softwareentwicklungsmethode, basierend auf einer Beschreibung mit XML.

3.1 Echtzeit-Java

Die Entwicklung der Java Plattform hin zur Unterstützung von eingebetteten Systemen ist durch Definition von unterschiedlichen Editionen in der Java 2 Plattform (vgl. Abbildung 3.1) weiter verbessert worden. Zusätzlich dazu erlauben weitergehende Bibliotheken und Definitionen mit Erweiterungen für die virtuelle Maschine der Java Laufzeitumgebung auch die Nutzung im Umfeld harter Echtzeitanforderungen [9].

Das gesamte Feld eingebetteter und mobiler Systeme wird durch die Java 2 Micro Edition [40] unterstützt. Die Palette dieser unterschiedlichen, anwendungsorientierten und reaktiven, kleinen Systeme wird durch die Partitionierung in zwei Konfigurationen und eine weitere Differenzierung mit Profilen, in denen Bibliotheken definiert werden, unterstützt.

Mit Definition einer Echtzeit Spezifikation für Java (“Real-Time Specification for Java”, RTSJ) [10] ist es gelungen, die Semantik der Java-Sprache um Anforderungen in harten und weichen Echtzeitumgebungen zu erweitern. Mit der RTSJ wird sowohl eine spezielle virtuelle Maschine als auch eine umfassende Bibliothek für die Bereitstellung von Echtzeit-Ablaufkoordinierung, Speichermanagement, Hardwarezugriff und Synchronisation definiert. Wesentliche Änderungen gegenüber der normalen virtuellen Maschine sind:

- Semantik für Echtzeit-Kontrollfäden (“Real-Time Threads”)
- Speicherverwaltung mit Nicht-Heap-Speicherbereichen
- Asynchrone Mechanismen (Nachrichten, Kontrolltransfer, ...)

[1] mit Erweiterungen und Restriktionen, die für harte und flexible Echtzeitanforderungen notwendig sind

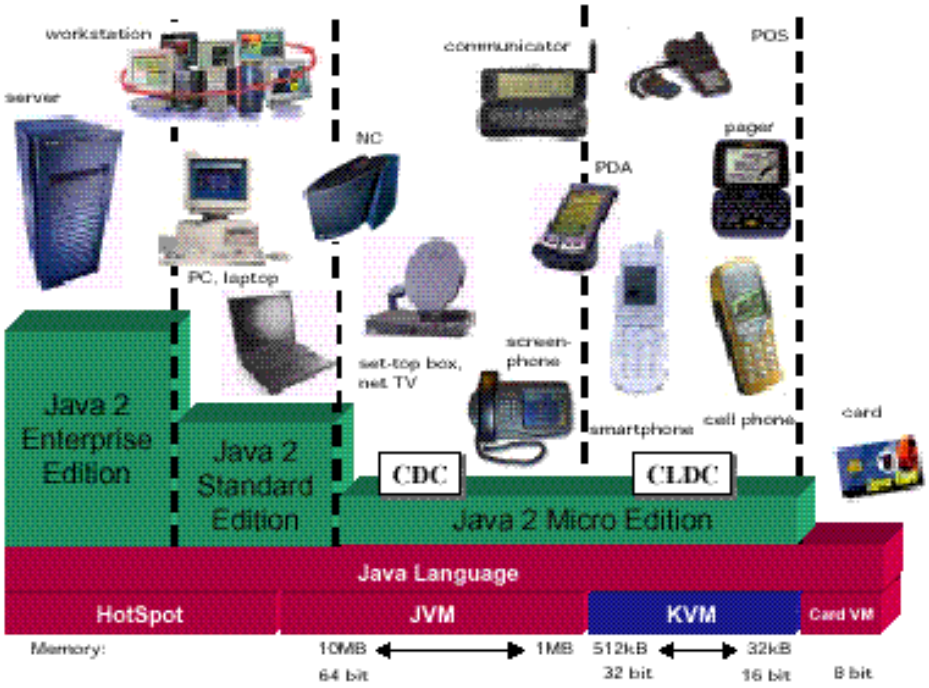

Abbildung 3.1: Java 2 Plattfom

- Wohldefinierte Synchronisation über Prioritäten
- Physikalischer Speicherzugriff (RawMemory)

Der in diesem Vorschlag vorgestellte Entwurf eines asynchronen Kommunikationsmusters [52] basiert auf diesen beiden Entwicklungen der Java-Plattform und zeigt die Notwendigkeit für Weiterentwicklungen in diesem Feld.

Die im Rahmen des europäischen Forschungsprojektes HIJA (vgl. Abschnitt 4.3.2) vorgeschlagenen Änderungen sollen durch unterschiedliche Profile für harte und flexible Echtzeitanforderungen definiert und mit Erweiterungen bzw. Einschränkungen der Standard-RTSJ für unterschiedliche EDV-Modelle erreicht werden. Nähere Informationen dazu bietet der Abschnitt 4.3.2. Im “Hard Real-Time Java” (HRTJ) Profil werden hier zum Beispiel nur Echtzeitkontrollfäden mit periodischer oder sporadischer[2] Ausführung erlaubt, da für aperiodische Aktivitäten ohne die Vorhersagbarkeit einer Mindestzwischenzeit die Analyse erst zur Laufzeit und nicht statisch im voraus möglich ist. Beim Einsatz von Nicht-Heap-Speicherbereichen ohne “Garbage Collection” schränkt HIJA für HRTJ RTSJ weiter ein. Es werden nur der dauerhafte (“immortal”) Speicherbereich und kurzlebige “scoped” Speicherbereiche mit linearer Allokationszeit, die nur von jeweils einem Kontrollfaden betreten und nicht gestapelt werden, erlaubt.

Durch Einsatz einer festen, prioritätsgebundenen Ablaufkoordinierung mit einem Laufzeitsystem, das ein Zugriffsprotokoll auf Ressourcen mit “Priority Ceiling Emulation” implementiert, lässt sich mit Techniken der “Rate Monotonic Analysis” (RMA) [35] die Abwesenheit von Verklemmungen (“deadlocks”) statisch analysieren und nachweisen. Nähere Überlegungen zur Ablaufkoordinierung und der Integration des vorgestellten Entwurfsmusters bieten die Abschnitte 3.2 und 3.3.7.

Eine weitere Überlegung, mit der die deterministische Ausführung für harte und flexible Echtzeitanforderungen erleichtert werden soll ist die Definition getrennte Phasen für Initialisierung und Aufgabenausführung (“mission”) [48]. Die Initialisierungsphase unterliegt noch nicht den Echtzeitanforderun-

[2] mit garantierten Empfangszwischenzeiten

gen und hier können Aktivitäten, wie die Erzeugung von Objekten und die Konfiguration dieser Objekte erfolgen.

3.2 Ablaufkoordinierung

Beim Entwurf objektorientierter, verteilter Systeme muss der Entwickler derzeit über ein umfassendes Verständnis von komplexen Techniken, wie z.B. Ablaufkoordinierung und Speichermanagement, verfügen. Die Entwicklung im HIJA Forschungsprojekt (vgl. Abschnitt 4.3.2) versucht ihn davon zu entlasten und mit der Definition von Anwendungsprofilen und EDV-Modellen einen standardisierten Rahmen zu bieten, innerhalb dessen der Entwickler sich auf den eigentlichen Teil seiner Anwendung konzentrieren kann.

Die nebenläufige Ausführung von Kontrollfäden ist fest in die Java Sprache und Laufzeitumgebung integriert [29]. Der Einsatz von Java mit eingebetteten Systemen soll diese Eigenschaft für sicherheitskritische Applikationen mit Echtzeitanforderungen erfüllen. In der RTSJ werden hierfür spezielle Echtzeitkontrollfäden definiert und ihre Ausführung mit einem Ablaufplan koordiniert. Bei nebenläufiger Ausführung von Kontrollfäden mit unterschiedlichen Prioritäten erlaubt die Java-Laufzeitumgebung die Unterbrechung (“preemption”) von Kontrollfäden, um die Ausführung solcher mit höherer Priorität zu beschleunigen. Der konkurrierende Zugriff von Kontrollfäden auf gemeinsam benutzte Ressourcen ist in Java (und RTSJ) mit gegenseitigem Ausschluss über Semaphore möglich. Um zu verhindern, dass Kontrollfäden mit höherer Priorität durch niederprioritäre, die ein Semaphor besitzen, in ihrer Ausführung aufgehalten werden (“priority inversion”), implementiert die RTSJ bekannte Mechanismen, die dies verhindern können [61]. 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 das Semaphor ohne Unterbrechung durch andere Kontrollfäden (mit einer mittleen Priorität) möglichst bald abgeben kann. Zur Vermeidung von möglichen Verklemmungen (“deadlocks”) wird in [16] ein Protokoll mit einem Prioritätshöchtmaß (“priority ceiling”) für jedes Semaphor definiert.

Prioritäten Prioritäten ermöglichen es, Kontrollfäden unterschiedlich zu gewichten und durch den Einsatz von Ablaufkoordinierung ihre Ausführung zu steuern. Der Einsatz von Prioritäten ist auch bei der Klassifizierung von Nachrichten innerhalb der Kommunikation hilfreich, durch Definition einer Gewichtung beim Nachrichtenaustausch ist es so möglich, die Priorität auf Sender- und Empfängerseite für die Verarbeitung zu bestimmen.

Ablaufkoordinierung Ablaufkoordinierung (“scheduling”) von nebenläufigen Aktivitäten in Echtzeitsystemen unterliegt der Bedingung, dass die Ausführung aller beteiligten Aktivitäten eine pünktliche Ausführung mit vorhersagbaren Zeitgrenzen erlauben muss. Harte Echtzeit in einem verteilten System verlangt, dass ein Kontrollfaden für die Kommunikation Zeit hat und aktiv ist und eingehende Nachrichten verarbeiten bzw. innerhalb einer harten Zeitgrenze aus einem evtl. Zwischenpuffer auslesen und verarbeiten kann. Um dies statisch vorherberechenbar zu machen, sind Ablaufkoordinierunsanalysen erforderlich, die dies garantieren können. Grundsätzlich besteht ein solches Verfahren aus drei Komponenten [75]:

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

Sobald das Verhalten unter schlechtesten Voraussetzungen berechnet ist, kann es mit den Zeitanforderungen des Systems verglichen werden, um sicherzustellen, dass alle Zeitschranken erfüllt werden.

Dieses Verfahren kann dynamisch zur Laufzeit oder wie für Systeme mit harten Echtzeitanforderungen notwendig im voraus und statisch eingesetzt werden.

Bei der RTSJ wird für die dynamische Ablaufkoordinierung der Algorithmus mit einem Fabrikmuster in der Klasse `javax.realtime.Scheduler` und ihren Unterklassen definiert. Die Spezifikation sieht standardmäßig nur eine Unterklasse `PriorityScheduler` mit einem Algorithmus für Ablaufkoordinierung mit festen Prioritäten ("Fixed Prioriy scheduling") vor. Mögliche andere Ablaufkoordinierungsalgorithmen wie z.B. die Ausführung von Aktivitäten mit frühen Zeitgrenzen zuerst ("Earliest Deadline First") sind denkbar [20].

Earliest Deadline First (EDF) EDF ist einer der gebräuchlichsten Ablaufkoordinierungsalgorithmen für Echtzeitsysteme. Als zeitbasierter Algorithmus garantiert er die Einhaltung aller Zeitgrenzen. Bei EDF wird den Kontrollfäden abhängig von der nächsten zu erreichenden Zeitgrenze der Prozessor zugeteilt. Wenn solch ein Ablaufplan existiert, der alle Zeitgrenzen erfüllt, ist dieses Verfahren deshalb 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.

Fixed Priority Scheduling (FPS) Ablaufkoordinierung über feste Prioritäten kann schon statisch vor der Programmausführung eine Planung auf Einhaltung von harten Zeitgrenzen ermöglichen. Zwei Algorithmen sind hier z.B. "Deadline-Monotonic (D-M) Scheduling" [3] oder "Rate-Monotonic (R-M) Scheduling" [39]. 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, durch den R-M-Algorithmus auch einer ermittelt wird. D-M-Ablaufkoordinierung bestimmt die Priorität abhängig von der erlaubten Zeitgrenze und für Zeitgrenze gleich Periodenlänge entspricht dieser Ansatz der Lösung mit R-M. Bei der tatsächlichen Auslastung (U) eines Prozessors durch mehrere Kontrollfäden erreichen diese Verfahren zwar eher schlechte Werte aber es wird garantiert, dass die Leistung immer zur Verfügung steht und alle Zeitanforderungen eingehalten werden. Die folgende Ungleichung zeigt diese Anforderung und für den mathematischen Beweis sei auf die Literatur verwiesen. $C_i$ist die schlechteste (längste) Ausführungszeit für die Aktivität 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 Aktivitäten definiert ist wurde versucht, diesen Ansatz auf aperiodische Aktivitäten zu erweitern [64, 2]. Dies ist aber für den Einsatz mit harten Echtzeitanforderungen nur für sporadische aperiodische Aktivitäten mathematisch beweisbar. Die Auswirkungen dieser Ergebnisse auf die vorgestellte Methodik wird in der Zusammenfassung (vgl. Abschnitt 3.3.7) nach der Vorstellung im nächsten Abschnitt näher beschrieben. Eine detailliertere Einführung in den Einsatz der RTSJ für nebenläufige Programmierung bietet z.B. [74].

3.3 Struktur und Interaktion - Entwurfsmuster des Rahmenwerkes

Für das bessere Verständnis der Struktur und Interaktion im Nachrichtenkanal-Netzwerk werden in diesem Abschnitt sämtliche Komponenten und ihr Zusammenspiel zur Erbringung der asynchronen und deterministischen Kommunikation beschrieben. Ausgehend vom Nachrichtenkanal selbst werden die auf ihm interagierenden aktiven Objekte sowie notwendige Kommunikationswarteschlangen definiert [54]. Abschnitt 3.3.7 illustriert diese Struktur in einer Zusammenfassung und es wird die Verbindung zu existierenden Entwurfsmustern gezogen und diskutiert.

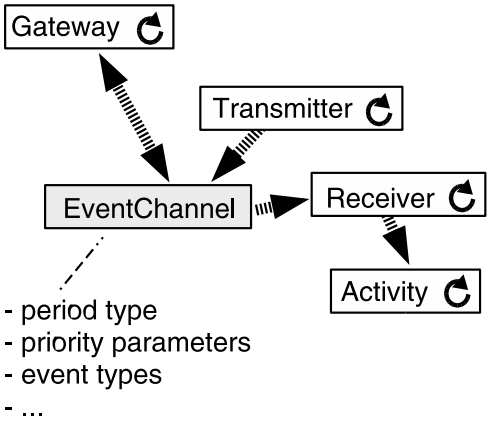

Abbildung 3.2: Nachrichtenkanal und zugeordnete verarbeitende Einheiten

3.3.1 Nachrichtenkanal

Der Nachrichtenkanal (“EventChannel”, vgl. Abbildung 3.2) ist die zentrale Komponente im vorgestellten Nachrichtendienst. Der Nachrichtenkanal repräsentiert das Thema der Publiziere/Abonnenten-Kommunikation und hier werden neben den möglichen Nachrichtentypen auf diesem Kanal Anforderungen an die aktiven Komponenten, die Nachrichten senden und empfangen definiert. Ihre Aufgaben werden im Folgenden beschrieben.

3.3.2 Empfänger-Kollektiv

Das Empfänger-Kollektiv umfasst die aktiven Komponenten eines Knotens, die für das Empfangen von Nachrichten zuständig sind. Um den fehlerfreien Empfang aller Nachrichten zu gewährleisten, müssen diese Kontrollfäden mit höchster Priorität[4] arbeiten und nach dem Empfang einer Nachricht wird diese zur weiteren Verarbeitung in eine nach Prioritäten sortierte Warteschlange (vgl. Abschnitt 3.3.6) eingereiht. Eine genauere Betrachtung der Prioritäten aller aktiven Komponenten im Nachrichtenkanal-Netzwerk wird in Abschnitt 3.2 bereitgestellt.

3.3.3 Zugriffssockel

Der Zugriff auf das dem Nachrichtenkanal-Netzwerk zugrunde liegende, physikalische Netzwerk wird über einfache Byte-orientierte Zugriffssockel garantiert. Diese Sockel bieten den Lese/Schreibe-Zugriff auf das Kommunikationsnetzwerk, das im wesentlichen Funktionalität, vergleichbar der ISO/OSI Schicht 5 (Transportschicht), anbieten muss. Hier wird kein zuverlässiges Protokoll als Basis vorausgesetzt. Der Zugriffssockel ermöglicht ohne weiteren Protokolloverhead die Ausnutzung von z.B. Echtzeitgarantien. Für einen Vergleich möglicher Kommunikationsnetzwerke und -protokolle vergleiche Abschnitt 2.4. Es ist denkbar, dass der Zugriffssockel neben der direkten Interaktion mit dem zugrundeliegenden Netzwerk weitere Funktionalität für die Aufteilung und Verschmelzung von Nachrichtenpaketen in Netzwerktransporteinheiten sowie die in Abschnitt 3.3.2 angeregte Bandierung und Oberbegrenzung von Prioritäten der empfangenden Kontrollfäden enthalten kann. Voraussetzung für den Einsatz mit Echtzeitnetzwerken muss dieser zusätzliche Protokollaufwand begrenzt und nach oben abschätzbar bleiben.

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

[4] Um zu verhindern, dass niederprioritäre Nachrichten allein durch ihren Empfang die Bearbeitung wichtiger Nachrichten behindern wird hier ein Verfahren mit Prioritätsbändern [44] für die verwendeten Zugriffssockel eingesetzt und so die Partitionierung der Rechenkapazität abhängig von Prioritäten ermöglicht.

3.3.4 Weitere aktive Komponenten für die Kommunikation

Neben einem Empfangsthread wird auf den Zugriffssockel durch eine Komponente für die Versendung ("Transmitter") und evtl. die Netzwerkeüberbrückung ("Gateway") zugegriffen. Analog zu den Empfangskontrollfäden werden die Parameter dieser aktiven Komponenten durch den Nachrichtenkanal vorgegeben. Diese Komponente ist für den asynchronen Versand der Nachrichten und die Interaktion mit dem zugrundeliegenden Kommunikationsnetzwerk notwendig. Der Versand der Nachrichten kann gepuffert sein um unterschiediche Zeitfenster eines z.B. zeitfensterbasierten Protokolls zu synchronisieren. Die erforderliche Priorität und Parameter für die Periode sowie verfügbare Warteschlangenkapazitäten sind hier erforderlich.

Eine denkbare Netzwerkbrücke zwischen unterschiedlichen zugrunde liegenden Kommunikationsnetzwerken vereinigt sowohl Empfänger- als auch Versenderfunktionalität und unterliegt dabei den Prioritätsanforderungen des Nachrichtenkanals.

3.3.5 Aktivitätsmanager und Pool von Aktivitäten

Der Aktivitätsmanager ist als aktive Komponente für die Abarbeitung der Warteschlangen des Empfänger-Kollektivs verantwortlich und garantiert die Zuordung wartender, empfangener Nachrichten zu freien Aktivitätskontrollfäden aus einem Pool. Wartende Kontrollfäden mit ausreichender Priortät werden aktiviert und die Abarbeitung mit registrierter Bearbeitungslogik ("PushEventHandler") für die Nachrichten mit ihnen gestartet.

3.3.6 FIFO-Warteschlangen

Die Interaktion aller aktiven Komponenten wird durch eine Kopplung mit FIFO-Warteschlangen unterstützt. Sowohl zwischen Empfänger-Kollektiv und Aktivitätsmanager als auch bei der Zuteilung von Nachrichten an Aktivitäten wird dieses Verfahren eingesetzt.

3.3.7 Zusammenfassung

Die soweit eingeführten Komponenten können in einem neuen Architekturentwurfsmuster zusammengefasst werden. Durch die Interaktion eines Kollektivs von Empfängern, der asynchronen Interaktion mit einer Managerkomponente, die empfangene Nachrichten anhand ihrer Priorität durch einen Pool verfügbarer Aktivitäten ausführen lässt, wird auf Basis einfacher synchroner Kommunikationsnetzwerke und -hardware ein asynchrones Kommunikationsmuster mit deterministischen Zeitbegrenzungen erlaubt.

Entwurfsmuster: Empfänger-Kollektiv, mit Aktivitätsmanager und Warteschlagen [54] Die Abbildung 3.3 illustriert die Struktur der bisher vorgestellten Komponenten im Nachrichtenkanal-Netzwerk. Das eingeführte Entwurfsmuster erfordert keine speziellen asynchronen Netzwerkzugriffs- oder Betriebssystemunterstützung. Der Entwurf ist ähnlich des Reaktormusters [57] und basiert ebenfalls auf einem synchronen Nachrichten-Demultiplexer, erweitert diesen aber um den Aktivitätsmanager und eine asynchrone Warteschlangeninteraktion.

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

Reaktor [59, 79]: Das Reaktor-Muster ist mit der hier vorgestellten Architektur verwandt und adressiert ebenfalls die 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

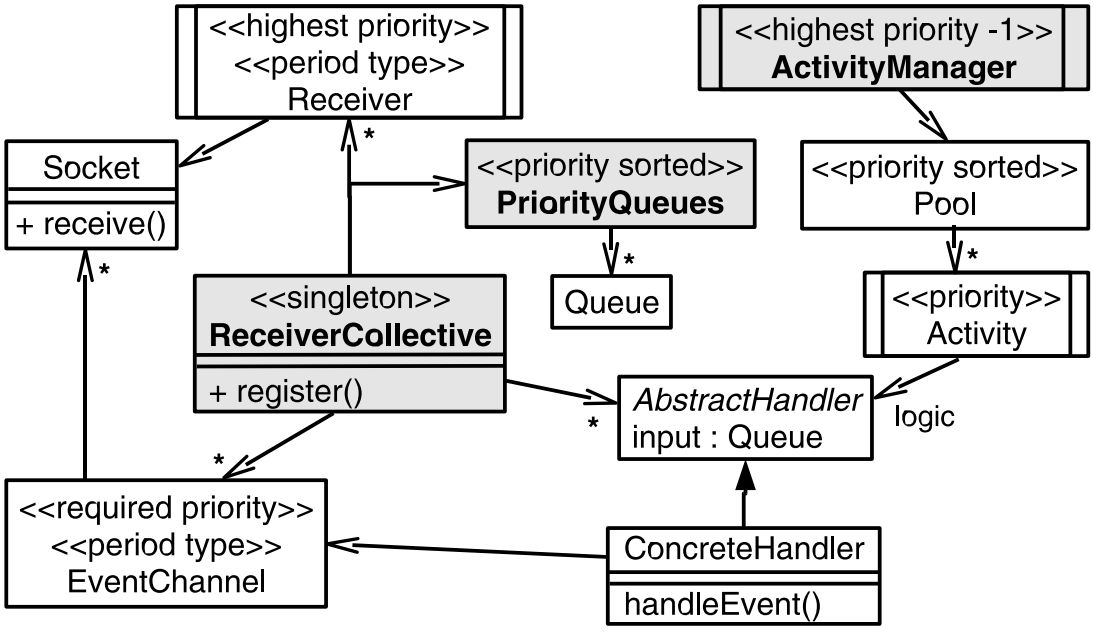

Abbildung 3.3: Struktur Nachrichtenkanal-Netzwerk

hier vorgestellte Ansatz erweitert das Muster durch den Einsatz von FIFO-Warteschlangen, um unterschiedliche Prioritäten in der Nachrichtenkommunikation zu unterstützen.

Beobachter [25]: Der Beobachter basiert ebenfalls auf dem Rückrufprinzip aber das Muster ist weniger auf Nachrichtenkommunikation, als auf direkte Methodenaufrufe ausgerichtet.

Aktivator [41]: Das zum Beispiel in der Java 2 Enterprise Edition oft verwendete Muster eines Aktivators[5] beschreibt den in dieser Architektur vorgesehenen Aktivitätsmanager, der bei asynchron eingetroffenen Nachrichten ihre Verarbeitung durch einen Pool von Aktivitätskontrollfäden steuert.

Der vorgestellte Nachrichtendienst bietet eine asynchrone, themenbasierte Publiziere/Abonniere-Kommunikation und unterstützt ein direktes "Push"-Nachrichtenübertragungsmodell ohne zentrale Diensteinheiten. Weitere Anforderungen und Informationen zu Kommunikation und Synchronisation bietet [51].

Die mit einem Nachrichtenkanal verbundenen Nachrichten können in periodischer oder sporadischer[6] Häufigkeit eintreffen und mit dieser Einschränkung wird eine Abbildung auf die in RTSJ (bzw. dem in HIJA vorgesehenen EDV-Modell für harte Echtzeitanforderungen HRTJ) verfügbare periodischen und sporadischen Kontrollfäden ermöglicht. Die zugeordneten Empfangskontrollfäden sind für den direkten Empfang oder die zeitnahe Leerung des Empfangspuffers der Netzwerkzugangsschnittstelle und die Einreihung der Nachrichten in ein nach Prioritäten sortiertes Warteschlangensystem verantwortlich.

Durch die Nutzung einer prioritätsbasierten Ablaufkoordinierung (vgl. Abschnitt 3.2) kann ausgehend von diesen Anforderungen auch für Systeme mit harten Echtzeitanforderungen statisch die Erfüllung von Zeitanforderungen garantiert werden. Die für jedes physikalische Netzwerk existierenden periodisch und kurzzeitig aktiven Empfängerkontrollfäden nutzen die Attribute der Nachrichtenkanäle für die Festlegung ihrer Ausführungsparameter und ermöglichen die Verarbeitung periodischer Nachrichten. Bei sporadischem Auftreten der Nachrichten wird das Verfahren von "sporadic servers" eingesetzt, das zusammen mit den periodischen Kontrollfäden eine statische Analyse für die gemeinsame Ablaufkoordinierung ermöglicht [64]. Innerhalb einer virtuellen Maschine kann die Ablaufkoordinierung für die Kommunikation eines Knotens im Nachrichtkanal-Netzwerk statisch berechnet und zu verifiziert werden.

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

[6] mit garantierten Empfangszwischenzeiten

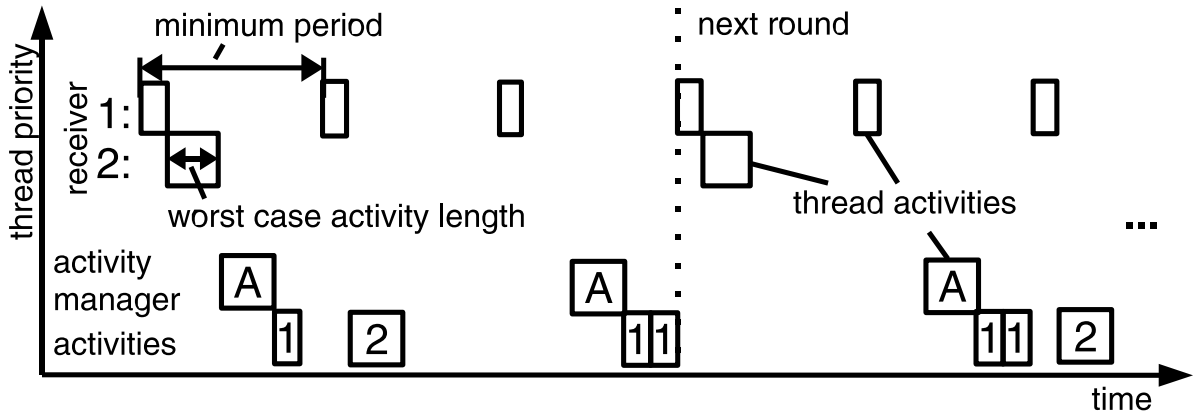

Abbildung 3.4: Beispiel der Ablaufkoordinierung mit zwei Empfängerkontrollfäden

Das in Abbildung 3.4 dargestellte UML-Zeitdiagramm beschreibt die eigentliche Kommunikation in der Ausführungsphase und lässt sich auf seine Ausführbarkeit hin prüfen. Wie in [55] näher beschrieben lässt sich für periodisch und sporadisch eintreffende Nachrichten und mit RMA vergebenen Prioritäten der Bearbeitungskontrollfäden, für die statisch Attribute wie maximale Nachrichten- und Puffergröße sowie Anforderungen bezüglich Antwortzeit und Verarbeitungsgeschwindigkeit bekannt sind, ein Gesamtablaufmodell finden, das alle Echtzeitanforderungen erfüllen wird.

Für den Einsatz mit RTSJ-Systemen bietet die Implementierung des Nachrichtenkanal-Netzwerkes nicht diese Garantien. Mit dem leichtgewichtigen und dezentralen Kommunikatonsprotokoll wird aber ein möglichst hoher Grad an Dienstqualität (“Quality of Service”, QoS) angestrebt.

3.4 Implementierung des Prototypen

Im Rahmen des HIJA Forschungsprojektes (vgl. Abschnitt 4.3.2) wird für die Entwicklung von “Application Neutral Real-Time System” (ANRTS) Applikationen die Java Sprachspezifikation [29] mit der Echtzeiterweiterung RTSJ als Grundlage definiert. Neben sprachlichen Mitteln wie Erweiterungen des Programmcodes durch Annotationen zur Unterstützung der Ausführungs- und Entwicklungsumgebung werden Nebenläufigkeit und die Möglichkeiten zur festprioritätsgebunden Ablaufkoordinierung für die Entwicklung zukünftiger, hart echtzeitfähiger und verteilter Applikationen proklamiert. Für die asynchrone Nachrichtenkommunikation sind zwei Versionen für harte und flexible Echtzeitsysteme basierend auf 100% purem Java mit der Echtzeiterweiterung RTSJ und mit einem geringen Prokolloverhead vorgesehen.

Dieser Abschnitt beschreibt die Implementierung der asynchronen Kommunikationsinfrastruktur und führt eine grundsätzliche Methodik beim Entwurf verteilter, eingebetteter Systeme mit asynchroner Kommunikation unter Echtzeitanforderungen ein.

Zur Implementierung der Kommunikationsinfrastruktur nutzt der Nachrichtendienst die Möglichkeiten der (eingeschränkten) RTSJ (vgl. Abschnitt 3.1). Das Nachrichtenaufkommen innerhalb der einzelnen Kanäle wird auf periodische und sporadische Kontrollfäden abgebildet und die Kapazitätsbegrenzung gleichzeitig unverarbeiteter Nachrichten erlaubt die Nutzung einer festen Datenstruktur im unveränderlichen Speicherbereich. Es wird eine sortierte Warteschlange der erforderlichen Kapazität während der Initalisierungsphase angelegt. Grundlage für diese Berechnung sind die Informationen über die Perioden aller Empfangskontrollfäden, die mittels des kleinsten gemeinsamen Vielfachen “Runden” definieren (vgl. Abbildung 3.4), nach denen jeweils die festen Pufferstrukturen geleert sein müssen. Nur für die flexible Version ist eine nachträgliche Änderung hier möglich.

In der Abbildung 3.5 werden die dauerhaften Datenobjekte der Version für harte Echtzeitanforderungen mit ihren Beziehungen zu den notwendigen Kontrollfäden dargestellt. Neben dem dauerhaften Spe-

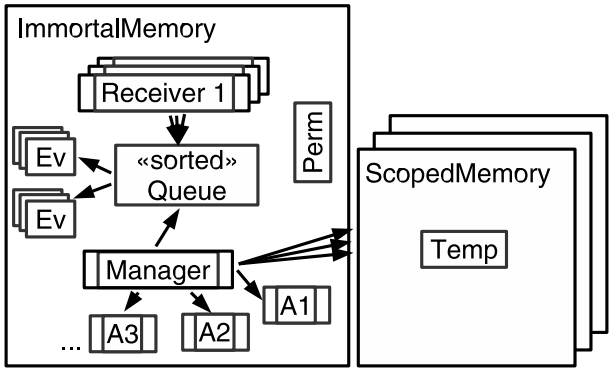

Abbildung 3.5: Speicher und Kontrollfäden

icherbereich werden in der Initialisierungsphase auch weitere kurzlebige Speicherbereiche für die Ausführung der Behandlungsaktivitäten definiert. Diese Speicherbereiche werden wiederverwendet und die dort erzeugten Objekte sind nur temporär verfügbar. Für Datenaustausch über mehrere Perioden und Runden hinweg müssen im dauerhaften Speicher Containerobjekte zur Verfügung stehen. Bei der statischen Version des Nachrichtendienstes für harte Echtzeitanforderungen kann dies mittels einer Konfiguratonsdatei mit allen notwendigen Beschreibungen erfolgen [55]. Diese Beschreibung kann mit Systemvoraussetzungen und Anwendungszielen verbunden werden und eine architekturneutrale Beschreibung des verteilten Systems bieten. Die flexible Version bei weichen oder gemischten Echtzeitanforderungen definiert einen gesonderten allen Partnern bekannten Nachrichtenkanal ("admin channel"), über den notwendige Verhandlungen und Benachrichtigungen stattfinden können [56].

Jeder Knoten verfügt pro Zugangssockel über einen oder mehrere Kontrollfäden mit höchster Priorität, die periodisch die asynchron eingetroffenen und im Netzwerkzugangspunkt gepufferten Nachrichten in Empfang nehmen und in ein Warteschlangensystem zur Weiterverarbeitung einreihen. Systeme mit harten Echtzeitanforderungen müssen garantieren, dass für alle eintreffenden Nachrichten ein Empfangskontrollfaden die Puffer der Eingangsschnittstelle lesen und Nachrichten an das Verarbeitungssystem weiterreichen kann. Dies muss zeitnah, bevor die Daten durch neue überschrieben würden, geschehen und kann ebenfalls durch Hardwarecharakeristika in der Systembeschreibung definiert sein.

Nachrichten in den Warteschlangen werden von einem Managerkontrollfaden an wartende Kontrollfäden mit passenden Prioritäten und Systemattributen weitergereicht und so die Behandlung in vorgegebenen Zeitgrenzen ermöglicht.

Die Abbildung 3.6 zeigt die hier beschriebene Interaktion beim Empfang von Nachrichten über mehrere physikalische Kommunikationsnetzwerke. Der Versand von Nachrichten wird nicht weiter betrachtet, weil neben der Definition von Veröffentlichungsattributen bei Nachrichtenkanälen lediglich eine entsprechende Bereitstellung über die Netzzugangsschnittstelle notwendig ist.

Beiden Formen werden auf Nachrichtenfrequenzen der definierten Nachrichtenkanäle abgebildet. Für Systeme mit harten Echtzeitanforderungen wird in HIJA die Segmentierung der Applikation in zwei Phasen (Initalisierungs- und Missionsphase) propagiert und das Nachrichtenkanal-Netzwerk nutzt diese Aufteilung und verlagert die nicht zeitkritischen Aktionen der Initalisierung in eine gesonderte Phase vor der eigentlichen echtzeitigen Ausführung.

3.5 Deskriptive Softwareentwicklungsmethode

Der so definierte asynchrone, direkte, themenbasierte Publiziere/Abonniere-Kommunikationsdienst greift direkt auf Datensicherungs- oder Transaktionsschicht[7] des unterliegenden Netzwerkes zu und verwendet für den Versand von Informationen einzelne Datenpakete oder möglichst wenige Datenzellen des

[7] Schicht 4 in ISO/OSI

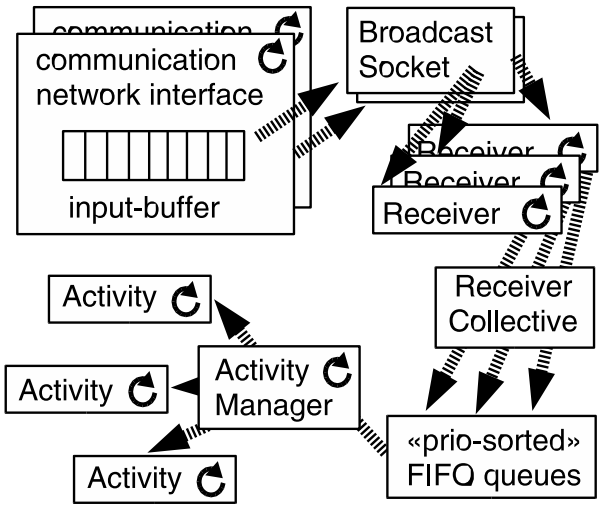

Abbildung 3.6: Struktur und Interaktion beim Nachrichtenempfang

Netzwerkprotokolls.

Die angebotene Eins-zu-Viele-Kommunikation lässt sich leicht in objektorientierten, verteilten Applikationen für eingebettete Systeme nutzen. Das Erfordernis für statische Analyse und Verifikation erlaubt eine deklarative Beschreibung von Knoten und Kommunikationsmechanismen der verteilten Applikation. Exemplarisch werden dazu im Folgenden einige Abschnitte in XML-Notation beschrieben. Für jeden Knoten im Netzwerk muss der Netzwerkzugangspunkt mit seinen physikalischen Voraussetzungen möglichst genau definiert werden.

```
01:   <socket name="socket" code="BroadcastSocketImpl">
02:    <buffer>12288</buffer>
03:   </socket>
```

Zeile 02 beschreibt hier die Pufferkapazität innerhalb des Netzwerkzugangspunktes. Neben der physikalischen Anbindung und Unterstützung für asynchrone Kommunikation mit Paketpuffern durch den Controller wird für einen Knoten, der als Publizist für Nachrichten agiert, die Beschreibung der angebotenen Kanäle definiert.

```
11:   <server name="Server1">
12:    <channel name="Channel42" id="42">
13:     <socket-ref>socket</socket-ref>
14:     <priority>13</priority>
15:     <periodic>100ms</periodic>
16:     <event-size>worst case size in bytes</event-size>
17:     <event-types>...</event-types>
18:    </channel>
19:   </server>
```

Der Dienstgeber beschreibt die von ihm zur Verfügung gestellten Nachrichtenkanäle und Nachrichten. Neben Anforderungen an Kontrollfäden auf Empfängerseite (Priorität und Periode, Zeilen 14,15) werden die verwendeten Netzwerkzugangspunkte und die Anforderung an die verfügbare Bandbreite des Netzwerkes in Byte (Zeile 16) für die größten Nachrichten definiert.

Jeder Knoten kann neben Dienstgeberfunktionalität auch als Dienstnehmer auftreten und nimmt in der Beschreibung auf Empfängerseite Bezug auf die Kanaldeklarationen der Dienstgeberknoten (Zeile 22).

```
21:   <client>
```

```
22:     <channel ref-name="Server1.Channel42"/>
23:     <capacity>queue-size</capacity>
24:     <handler>event-type and handler mapping </handler>
25:    </client>
```

Für den Einsatz mit flexiblen Echtzeitanforderungen schlägt [56] eine auf XML und XSLT basierende Generierung des Programmcodes zur Nutzung der Nachrichtendienst-Infrastruktur vor.

Kapitel 4

Einsatz und zeitliche Planung

Zur Bewertung der vorgestellten Methodik und Untersuchung des eingeführten Prototyps werden in der Dissertation auf unterschiedlichen Ebenen Vergleichskriterien definiert. Da das vorgestellte Verfahren asynchroner Nachrichten-Kommunikation sowohl mit harten als auch flexiblen Echtzeitanforderungen Vorteile bietet, ist es Ziel, beide Versionen in realistischen Systemumgebungen auf ihre Einsatzmöglichkeit zu testen. Dieses Kapitel führt einerseits kurz die konzeptuellen Unterschiede zwischen den beiden geplanten Versionen ein und beschreibt im Abschnitt 4.3 die möglichen Testszenarien, die sich durch die Beteiligung an den dort vorgestellten Forschungsprojekten ergeben.

4.1 Statische Version mit harten Echtzeitanforderungen

Harte Echtzeitanforderungen verlangen statische Analyse, um die Ablaufkoordinierung vor der Ausführung zu verifizieren. Für die Kommunikation mit Nachrichten bedeutet dies eine Deklaration aller notwendigen Systemattribute im voraus und diese Anforderung erlaubt die Nutzung einer deskriptiven Entwicklungsmethode in XML (vgl. Abschnitt 3.5).

Für die Verifikation der Entwicklungsmethode und der Nachrichtenkommunikation soll neben einem Prototypen für ein sicherheits- oder geschäftskritisches Steuerungs- und Kontrollsystem insbesondere die statische Analyse der Ablaufkoordinierung für ein Gesamtsystem untersucht werden.

4.2 Dynamische Version mit flexiblen Echtzeitanforderungen

Bei der Nutzung mit flexiblen Echtzeitanforderungen wird eine deterministische Behandlung möglicher Fehlerquellen zur Laufzeit interessant. In [56] wird basierend auf unterschiedlichen möglichen physikalischen Kommunikationsnetzwerken ein Fehlermodell vorgestellt und zwei Verfahren der Fehlerbehandlung für periodische und sporadische Nachrichten vorgeschlagen. Der dynamische Charakter wird durch Nutzung eines Verwaltungsnachrichtenkanals, auf dem die Erzeugung, Suche und Bewerbung neuer Kommunikationskanäle erfolgt, möglich.

Für die Nutzung im HIJA Forschungsprojekt soll auf Basis des vorgestellten Publiziere/Abonniere-Nachrichtendienstes eine Gleiche-zu-Gleiche-Kommunikationsinfrastruktur (Peer-to-Peer, P2P) mit dem Programmierschnittstelle des Industriestandards Pastry [37] implementiert und genutzt werden. Die Nutzung mit “Controller Area Network“ (CAN) Netzwerken im Automobil wird untersucht [50]. Aus beiden Testszenarien sollen realistische Bewertungskriterien und Erfahrungen bei der Nutzung in die Dissertation integriert werden.

4.3 Projekte

Der erfolgreiche Einsatz des Nachrichtenkanal-Netzwerkes wurde und wird derzeit in zwei Forschungsprojekten, die von der Europäischen Kommission gefördert sind, belegt. Das von 2002 bis 2004 durchgeführte Forschungsprojekt zu "High Integrity Distributed Object-Oriented Realtime Systems" (HIDOORS) [30] fokusierte sich dabei im wesentlichen auf den Einsatz von Echtzeit-Java (RTSJ) für die objektorientierte Entwicklung von verteilten eingebetteten Systemen mit (weichen) Echtzeitanforderungen. Mit dem Nachfolgeprojekt von 2004 bis 2006 "High Integrity Java Applications" (HIJA) [32] wurde das Forschungsfeld auf Analysetechniken für funktionale und nicht-funktionale Anforderungen in sicherheits- und geschäftskritischen eingebetteten Systemen ausgedehnt und die Entwicklung des Nachrichtendienstes wird um harte Echtzeitanforderungen erweitert.

4.3.1 HIDOORS

Das Projekt HIDOORS untersuchte die Voraussetzungen für eine integrierte Entwicklungsumgebung für eingebettete Systeme basierend auf der Java 2 Plattform. Mit der Bibliothek für Nachrichtenkommunikation wurden verteilte Applikationen auf Basis des "EventChannelNetwork" in UML entworfen und durch automatisierte Quellcode-Generierung große Teile der notwendigen Kommunikationsinfrastruktur ohne Mehraufwand erzeugt. Die Eignung des Nachrichtenkanal-Netzwerkes für verteilte, eingebettete Systeme wurde mit einem Demonstrator aus dem Avionik-Umfeld belegt. Weitere Informationen zur Testumgebung und -ergebnissen bietet der Abschnitt 4.4.1.

4.3.2 HIJA

Mit HIJA wird in einem weiteren Projekt die Eignung von Java bei der Applikationsentwicklung im Umfeld eingebetteter Systeme mit Echtzeitanforderungen untersucht und in verschiedenen Testszenarien die Eignung des Nachrichtenkanal-Netzwerkes mit den vorgesehenen EDV-Modelle überprüft. Die EDV-Modelle werden als Profile definiert und sind hier kurz beschrieben:

Hard Real-Time Java (HRTJ): Dieses Profil adressiert sicherheitskritische Applikationen, wie zum Beispiel solche, die dem Standard DO-178B (Level A und B) für Softwareentwicklung in der Flugzeugtechnik der "Radio Technical Commission for Aeronautics" (RTCA) entsprechen. Dies erfordert unter anderem statische off-line Analysetechniken, die die Funktionstüchtigkeit des Systems schon vor der Inbetriebnahme verifizieren können. Um das nebenläufige EDV-Modell von Java hier zu ermöglichen wird, die RTSJ analog zum Ada Ravenscar Profil [15] weiter eingeschränkt und die Laufzeitumgebung muss eine pre-emptive prioritätsbasierte Ablaufkoordinierung (vgl. Abschnitt 3.2) für periodische und sporadische Kontrollfäden garantieren. Das Priority Ceiling Protokoll ist notwendig um die statische Analyse der Ablaufkoordinierung mit festen Prioritäten zu ermöglichen. Durch Einsatz unterschiedlicher Phasen für Initialisierung und Ausführung wird bei diesem Profil garantiert, dass nach der Initialisierungsphase alle Kontrollfäden für die Durchführung der folgenden Aktivitäten erzeugt sind und entsprechend harte Echtzeitanforderungen einhalten können. Der Informationsaustausch zwischen Kontrollfäden erfolgt über Puffer-Datenstrukturen im dauerhaften Speicherbereich und "Garbage Collection" wird nicht eingesetzt.

Flexible Soft Real-Time Java (FSRTJ): Das Ziel bei der Entwicklung dieses Profils ist der Einsatz bei geschäftskritischen Applikationen. Viele dieser Applikationen haben weiche Echtzeitanforderungen, die durch den Einsatz echtzeitfähiger "Garbage Collection" Techniken [63] auch ohne die speziellen Nicht-Heap-Speicherbereiche und anderer Einschränkungen die Verwendung von Java Technologie erlauben sollen.

Flexible Mixed Real-Time Java (FMRTJ): Das Ziel dieses Profils ist der gleichzeitige Einsatz von Komponenten, die harten bzw. weichen Echtzeitanforderungen unterliegen. Für dieses Szenario wird der volle Funktions- und Leistungsumfang der Standard-RTSJ eingesetzt.

4.4 Test-Szenarien

Die Entwicklung für das Forschungsprojekt HIDOORS war eine Vorläuferimplementierung des in Abschnitt 3.3 vorgestellten Entwurfsmusters. Die grundlegenden Komponenten dieses Kommunikationsrahmenwerkes waren zwar schon mit der endgültigen Implementierung im Projekt HIJA vergleichbar, aber die Aufteilung in zwei Profile für weiche und harte Echtzeitanforderungen war noch nicht vorgesehen.

Im Projekt HIJA gibt es sowohl Beispielszenarien mit harten (vgl. Abschnitt 4.4.5), als auch mit weichen oder gemischten Echtzeitanforderungen (vgl. Abschnitt 4.4.3). Um den direkten Zugriff auf Testergebnisse zu garantieren, werden die Test-Szenarien bei Projektpartnern jeweils um eigene Versuche mit eingeschränkter Hardware- und Software-Infrastruktur ergänzt. Durch Studien- und Diplomarbeiten ist es möglich, Gleiche-zu-Gleiche-Netzwerke auf Basis des Rahmenwerkes sowie seine grundsätzliche Eignung für zeitgesteuerte Echtzeitkommunikationsnetze zu überprüfen.

4.4.1 HIDOORS - Avionics

Beim Einsatz der Vorversion des Nachrichtenkanal-Netzwerkes im Projekt HIDOORS wurde Wert auf die durchgängige Entwicklungsmethodik von UML bis hin zur automatisierten Codeerzeugung gelegt. Ergebnisse aus der Verwendung mit dem UML-Entwurfswerkzeug von Aonix sowie die Ergebnisse der portugisischen Firma Skysoft sollen hier integriert werden.

Für die Demonstration der Funktionaltät des Rahmenwerkes wurden unterschiedliche Beispiele auf Basis von UDP/IP und “Controller Area Network” (CAN) Netzwerken entwickelt. Diese Beispiele werden an die aktuelle API angepasst und mit Beispielen für harte Echtzeitanforderungen ergänzt.

4.4.2 Gleiche-zu-Gleiche-Netzwerk mit Pastry

Der Einsatz von asynchroner Kommunikation als Grundlage für Gleiche-zu-Gleiche-Protokolle wird in einer Studienarbeit im Rahmen der Dissertation untersucht [37]. Die flexible Version des Nachrichtenkanal-Netzwerkes wird dabei um die dynamische Erweiterung der Kommunikation mit neuen Nachrichtenkanälen ergänzt.

4.4.3 HIJA - AI

Die Projektpartner Telecom Italia und Bellstream, Polen, setzen auf diese Version des Rahmenwerkes auf und implementieren eine API bzw. einen “Ambient Intelligence Demonstrator” um die weichen Echtzeitanforderungen mit asynchroner Kommunikation garantieren zu können.

4.4.4 TTP/C-Simulator

Für harte Echtzeitanforderungen erfordert die vorgestellte Methodik den Einsatz von kostenintensiven, physikalischen Netzwerken, die diese garantieren können. Um die prinzipielle Funktionalität demonstrieren zu können, wird der Prototyp deshalb erst einmal auf einem neu entwickelten Simulator für TTP/C Kommunikationsnetze [17] getestet. Der TTP/C-Simulator setzt auf normaler Internettechnologie, TCP/IP, auf. Das Testszenario soll jedoch die Eignung asynchroner Kommunikaton auch ohne zugrundeliegendes nachrichtenorientiertes Kommunikationsnetz demonstrieren.

Mit statischer Analyse der Ablaufkoordinierung für einen Kommunikationsknoten soll ein weiteres Test-Szenario für den asynchronen Nachrichtendienst aufgebaut werden. Die Simulation von Nachrichteneingang und -ausgang soll dabei auf einfacher PC-Architektur mit einem Echtzeitbetriebssystem (VxWorks) praktisch verifiziert werden.

4.4.5 HIJA - Avionics

Für den Einsatz der vorgestellten Methodik auf einem System unter sicherheitskritischen harten Echtzeitanforderungen empfiehlt sich ein Test-Szenario des HIJA Projektpartners Thales Avonics in Toulouse. In Kombination mit statischen Analysetechniken für die Kommunikation soll hier ein Einsatz von asynchroner Kommunikation untersucht werden.

4.5 Vorläufiger Zeitplan

ab Januar 2006: Erster Prototyp für beide Versionen des Nachrichtenkanal-Netzwerkes. Übergabe des Prototypen an verschiedene HIJA Projektpartner, um die praxisnahe Erprobung zu starten. Beim Übergabetreffen wird auf weiterentwickelte Beispiele des HIDOORS Forschungsprojektes aufgebaut und besonders mit dem deklarativen Applikationsentwurf unter harten Echtzeitanforderungen der Einsatzbereich erweitert.
Für den Einsatz mit weichen Echtzeitanforderungen wird das auf dem asynchronen Nachrichtendienst basierende Gleiche-zu-Gleichen-Netzwerk mit Pastry-API vorgestellt.

bis März 2006: Anpassung des Prototypen an die Testumgebungen der Projektpartner. Aufbau eines eigenen Test-Szenarios für harte Echtzeitanforderungen auf Basis des neu entwickelten TTP/C-Simulators.

April 2006: 2. Review durch die Europäische Kommisson, Vorstellung der asynchronen Kommunikation in Applikationen mit harten und weichen Echtzeitanforderungen.

Sommer 2006: Erste Erfahrungen aus den Test-Szenarien der Projektpartner und eigenen Versuchen, vorläufge Version der Dissertation.

Herbst 2006: Abschluss des europäischen Forschungsprojektes mit Praxisergebnissen über die Nutzung von asynchroner Kommunikation in sicherheits- und geschäftskritischen Applikationen mit Echtzeitanforderungen. Integration von Softwareentwicklungsrichtlinien der Dissertation in das Methodologie-Handbuch von HIJA.

November 2006: Endgültige Version der Ausarbeitung.

4.6 Veröffentlichungen

- 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.
- Marc Schanne. **Real-Time Communication with a Receiver Collective, Activity Manager, and Queues.** In Proceedings of IADIS Internatonal Conference Applied Computing 2005, Algarve, 2005.
- Marc Schanne. **Anforderungsanalyse für asynchrone Kommunikation beim Entwurf von objektorientierten, verteilten Systemen unter Echtzeitbedingungen.** Workshop zu Zuverlässigkeit in eingebetteten Systemen. Ada Deutschland Tagung 2005.
- Marc Schanne. **Real-Time Communication with Direct Publish/Subscribe Event Service.** 3rd Workshop on Java Technologies for Real-time and Embedded Systms (JTRES) OOPSLA 2005.
- Marc Schanne. **Asynchrone Kommunikation bei objektorientierten verteilten Systemen mit Echtzeitbedingungen.** In GI Softwaretechnik-Trends, Band 25, Heft 4, November 2005. PDF

Kapitel 5

Zusammenfassung und Ausblick

Durch die Integration in europäische Forschungsprojekte versucht der vorgestellte Dissertationsvorschlag realistische Einsatzszenarien als Testumgebung zu nutzen. Die Verifikation der Vorteile einer asynchronen nachrichtenbasierten Publiziere/Abonniere-Kommunikation für aktuelle und zukünftige eingebettete Systeme im Umfeld sicherheits- und geschäftskritischer verteilter Applikationen ist Ziel der präsentierten Methodik. Hiermit und mit Integration einer Softwareentwicklungsmethode, die Programmiersprachenkonzepte der 5. Generation ausnutzt, soll die Entwicklung dieser Systeme vereinfacht und verbessert werden.

Die Verwendung asynchroner Kommunikation für die einfachen Entwicklung von verteilten Systemen mit Echtzeitanforderungen wird durch ein klares Konzept von Nachrichtenkanälen vereinfacht. Die Anforderungen an den Entwickler solcher nebenläufgen Applikationen wird durch die Bereitstellung eines Rahmenwerks reduziert und der Entwickler kann sich auf die eigentliche Logik konzentrieren. Die in Systemen mit harten Echtzeitanforderungen notwendige statische Verifikation erfordert eine vollständige Systembeschreibung. Für den Aspekt der Kommunikation wird diese Beschreibung von Zeitanforderungen und Systeminformationen genutzt um große Teile der Applikationsentwicklung zu automatisieren. Da für harte und flexible Echtzeitanforderungen die gleiche Methodik zum Einsatz kommt, vereinheitlicht das Nachrichtenkanal-Netzwerk darüber hinaus die Entwicklung von neuen und komplexen sicherheits- und geschäftskritischen Systemen.

Über die in den beiden Forschungsprojekten genutzten Einsatzszenarien für asynchrone Kommunikation mit Publiziere/Abonniere-Kommunikation sind noch weitere Verwendungen denkbar. Der mit Nutzung von Pastry (vgl. Abschnitt 4.4.2) schon geplante Einsatz des Nachrichtendienstes als Basis für höherwertige Protokollabstraktionen würde sich zum Beispiel auch für eine bessere Integration in den aktuellen Sprachstandard Java 5 eignen. Die dort eingeführten asynchronen lokalen Methodenaufrufe mit Future-Objekten versucht [26], ähnlich anderer Versuche mit RMI [24], für die verteilte Nutzung zu erweitern. Die Garantie von Echtzeitanforderungen für asynchrone Kommunikation mit dem Publiziere/Abonniere-Muster legt außerdem den Einsatz in einer von der OMG definierten Spezifikation für “Data Distribution Service” [46] nahe.

Bibliography

[1] Jboss application server. website. http://www.jboss.org, 2005.

[2] 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.

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

[4] 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.

[5] Helmut Balzert. *Lehrbuch der Software-Technik. Software-Entwicklung*. Spektrum Akad. Verlag, 1996.

[6] 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.

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

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

[9] Greg Bollella, Ben Brosgol, Peter Dibble, Steve Furr, James Gosling, David Hardin, Marc Turnbull, and Rudy Belliardi. *The Real-Time Specification for Java*. Addison-Wesley, 2000.

[10] 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.

[11] Andrew Borg and Andy Wellings. A real-Time RMI Framework for the RTSJ. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)*, 2003.

[12] 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.

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

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

[15] Alan Burns. The Ravenscar Profile. Technical report, Real-Time Systems Research Group, Department of Computer Science, University of York, UK, 1998.

[16] 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.

[17] Nawfel Chaieb. Systemarchitektur fÃ¼r einen TTP/C-Simulator mit Testumgebung auf Basis zuverlässiger Transportdienste, 2006. (to appear).

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

[19] Miguel A. de Miguel. Solutions to make Java-RMI time predictable. In *Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 379–386, 2001.

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

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

[22] Wolfgang Emmerich. Software Engineering and Middleware: A Roadmap. In *Proceedings of the conference on The future of Software engineering*, pages 117–129, Limerick, Ireland, June 2000.

[23] 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.

[24] Katrina E. Kerry Falkner, Paul D. Coddington, and Michael J. Oudshoorn. Implementing Asynchronous Remote Method Invocation in Java. Technical Report DHPC-072, Distributed High Performance Computing Group. Department of Computer Science. University of Adelaide, July 1999.

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

[26] Eka Gelashvili. Prototypimplementierung fÃ¼r einen asynchronen entfernten Methodenaufruf in Java mit Future-Objekten.

[27] M. Gergeleit, J. Kaiser, and H. Streich. DIRECT: Towards a Distributed Object-Oriented Real-Time Control System. In *Proceedings of Workshop on Concurrent Object-based Systems*, 1994.

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

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

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

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

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

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

[34] 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.

[35] 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.

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

[37] Alexander Liebrich. Gleiche-zu-Gleiche-Protokolle auf Basis asynchroner Nachrichtenkommunikation mit prototypischer Implementierung eines Pastry-Netzwerkes, 2005. (to appear).

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

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

[40] Sun Microsystems. Datasheet Java 2 Platform, Micro Edition. The Java platform for consumer and embedded devices, howpublished = http://java.sun.com/j2me/docs/j2me-ds.pdf, year = 2002.

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

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

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

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

[45] Object Management Group, Inc. *CORBA Event Service Specification*, 1.2 edition, October 2004.

[46] Object Management Group, Inc. *Data Distribution Service for Real-time Systems Specification*, version 1.0, formal/04-12-02 edition, December 2004.

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

[48] 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.

[49] Dirk Riehle. The Event Notification Pattern – Integrating Implicit Invocation with Object-Orientation. *Theory and Practice of Object Systems*, 2(1):43–52, 1996.

[50] Alexander Roth. Prototypimplementierung der "Real-Time Data Access"(RTDA) Spezifikation 2.0 fÃ$\frac{1}{4}$r den Java-Hardwarezugriff am Beispiel einer "Controller Area Network"(CAN) Schnittstellenkarte, 2005. (to appear).

[51] Marc Schanne. Anforderungsanalyse für asynchrone Kommunikation beim Entwurf von objektorientierten, verteilten Systemen unter Echtzeitbedingungen. In *Proceedings of Workshop Zuverlässigkeit in eingebetteten Systemen. Ada Deutschland Tagung 2005*, pages 23–37, October 2005.

[52] Marc Schanne. Asynchrone Kommunikation bei objektorientierten verteilten Systemen mit Echtzeitbedingungen. *GI Softwaretechnik-Trends*, 25(4), November 2005.

[53] Marc Schanne. ECN - Event Channel Network. Project Website. http://www.eventchannelnetwork.org, 2005.

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

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

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

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

[62] 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.

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

[64] 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.

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

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

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

[68] Sun Microsystems. *Jini Specification*, 2.0 edition, June 2003.

[69] Sun Microsystems. *InfoBus 1.2 Specificaton*, 1.2 edition, February 2004.

[70] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems. Principles and Paradigms*. Prentice Hall, Inc, 2002.

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

[72] Jean Pierre Thomesse. A review of the fieldbuses. *Annual Reviews in Control*, 22:35–45, 1998.

[73] TTA Group. *TTP. Time-Trigered Protocol TTP/C. High-Level Specification Document*, July 2002.

[74] Andy Wellings. *Concurrent and real-Time Programming in Java*. John Wiley & Sons, Ltd, 2004.

[75] 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.

[76] 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.

[77] Wikipedia.org. Embedded system. http://de.wikipedia.org/wiki/Eingebettetes_System, December 2005.

[78] 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.

[79] Janusz Zalewski. Real-Time Software Design Patterns, 2002.