Anforderungsanalyse für asynchrone Kommunikation beim Entwurf von objektorientierten, verteilten Systemen unter Echtzeitbedingungen

Marc Schanne

13. Oktober 2005

FZI Forschungszentrum Informatik
Software Engineering
schanne@fzi.de

Aus dem wachsenden Einsatz von verteilten Anwendungen mit Echtzeitanforderungen in sicherheitskritischen Systemen resultiert die Notwendigkeit der Klärung von Anforderungen, Begriffen und Techniken für den Anwendungsentwurf in diesem Einsatzszenario. Die hier vorgestellte Methodik verwendet asynchrone Kommunikation bei der einfachen Entwicklung verteilter sicherheits- oder geschäftskritischer Anwendungen und definiert Fachbegriffe und Systemanforderungen an eine objektorientierte Umgebung, die nebenläufige Programmierung mit Kontrollfäden erlaubt und bei der Kommunikation Ende-zu-Ende-Vorhersagbarkeit, Rechtzeitigkeit, Schnelligkeit und Echtzeitgarantien verlangt. Mit Einsatz einer Java-VM, die die "Real-Time Specification for Java" (RTSJ) unterstützt, mit prioritätsbasierter Ablaufkoordinierung, einem Software-Entwurfsmuster für asynchronen Nachrichtenempfang und -abfertigung und der Nutzung von COTS Hardware- und Software-Komponenten ist es möglich, Echtzeitanforderungen zu garantieren und dies bereits durch statische Analyse zu verifizieren.

Keywords: Real-Time, Event Service, Networking, Embedded Systems

1 Einführung

Diese Analyse präsentiert die aktuelle Forschungsarbeit zu EDV- und Netzwerk-Modellen für sicherheits- und geschäftskritische Java Anwendungen ("High-Integrity Java Applications", HIJA[1]) [1, 20]. Für den Entwurf verteilter Systeme mit eingebetteter Software, die Echtzeitbedingungen erfüllen, wird ein asynchroner und direkter Publiziere/Abonniere-Nachrichtendienst mit beschränkten Zeitgrenzen bei der Ausführung definiert. Dieser ist für moderne Echtzeitbusse und -netzwerke optimiert. Abschnitt 4 führt eine Methodik und das notwendige Architekturmuster ein, um den Entwurf für verteilte Rechtzeitsysteme zu demonstrieren. Eine weitere Beschreibung der wesentlichen Komponenten dieses Nachrichtendienstes ist in [23] zu finden.

Im Folgenden wird zuerst eine Übersicht der notwendigen Laufzeitumgebung gegeben und die Probleme in der Domäne werden eingeführt, bevor der Lösungsvorschlag näher diskutiert wird. Abschnitt 2 definiert erforderliche Fachbegriffe und klärt über eine beispielhafte Darstellung verwandter Arbeiten (vgl. Abschnitt 2.4) die Grundlagen bei der Entwicklung für sicherheits- oder geschäftskritische Systeme. Die in Abschnitt 3 folgende Beschreibung der Anforderungen bei der Entwicklung für verteilte, eingebettete Systeme identifiziert die Mängel bei bestehenden Umgebungs- und Anwendungsdefinitionen. Die vorgestellte objektorientierte Lösungsinfrastruktur

1 IST-511718, Forschungsprojekt im 6. Rahmenprogramm der Europäischen Kommission

für den ein-gesetzten Nachrichtendienst wird abschließend bewertet und die Zusammenfassung der Analyse endet mit dem Ausblick auf mögliche zukünftige Erweiterungen.

Mit dem Einsatz verteilter Systeme [30] auch bei eingebetteten sicherheits- oder geschäftskritischen Systemen [27] sollen deren Vorteile auf Echtzeitsysteme [16] übertragen werden.

- Moderne, intelligente Sensoren und Aktoren erlauben die Verarbeitung von Daten näher an der Quelle bzw. Senke.
- Durch Fehlerbehandlung innerhalb einzelner Komponenten kann das Gesamtsystem eine höhere Zuverlässigkeit garantieren.
- Komponenten erlauben den Entwurf von komplexeren Systemen durch Wiederverwendung im Einsatz getesteter Bausteine.
- Die Skalierbarkeit des Systems kann durch einfache Erweiterbarkeit mit Knoten und replizierter Funktionalität besser gewährleistet werden.
- Der modulare Entwurf von Systemen verbessert die Wartbarkeit durch einfachen Austausch defekter Teile und Vereinfachung der Verkabelung.

Objektorientierte, eingebettete Systeme für sicherheitskritische Anwendungen können durch ihren Entwurf diese Vorteile nutzen und so die Entwicklung sicherheits- oder geschäftskritischer Systeme für Regelungstechnik, Automation, Avionik und Automobiltechnik der Zukunft bestimmen.

2 Fachbegriffe und Grundlagen

Zum besseren Verständnis des adressierten Anwendungsfeldes bietet dieser Abschnitt eine technische Diskussion von Begriffen und Paradigmen verteilter, echtzeitfähiger und sicherheits- oder geschäftskritischer eingebetteter Systeme.

2.1 Kommunikation und Synchronisation

Basis für verteilte Systeme ist die Verfügbarkeit einer Interprozesskommunikation [30]. Nebenläufige und vernetzte Dienste erlauben Systeme mit gesteigerter Performanz, mehr Zuverlässigkeit, besserer Skalierbarkeit und Kosteneffizienz [25]. Diese Vorteile im Bereich der Standard- und Unternehmenssoftware-Anwendungen sind auch für eingebettete Systeme nutzbar. Netzwerke von Sensoren, Aktoren und Prozessoren werden eingesetzt um sicherheits- oder geschäftskritische Systeme zu unterstützen. Für ihre Entwicklung sind unterschiedlich Formen der Vernetzung sowie Kommunikations- und Synchronisationsmechanismen denkbar. Diskutiert werden mögliche Prinzipien und Paradigmen bei der Kommunikation und Synchronisation zwischen unterschiedlichen vernetzten Komponenten und allgemein übliche Fachbegriffe in diesem Kontext werden eingeführt.

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 2.2) muss diese Kapazität ausreichend sein um eine rechtzeitige Auslieferung zu garantieren. Protokolle, strukturiert in Schichten[2], erlauben die Kommunikation auf Anwendungsschicht. Der Anwendungsnutzen des Kommunikationsprotokolls unterscheidet zwei grundlegende Formen: **verbindungsorientierte** und **verbindungslose** Kommunikation.

2 z.B. vergleichbar mit dem Open Systems Interconnection(ISO/OSI) Referenzmodell

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

Tabelle 1: Kommunikation und Synchronisation

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

Verbindungslose Kommunikation erfordert die Möglichkeit für 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"** [11] garantieren.

In einer objektorientierten Anwendungsumgebung, wie z.B. Java, gibt es für beide Kommunikationsformen konkrete Programmbibliotheken: Entfernter Methodenaufruf ("Remote Method Invocation", RMI [28]) und Java Nachrichtendienst ("Java Message Service", JMS [29]). Diese Standardimplementierungen erfüllen keinerlei Echtzeitanforderungen und sind nicht für eingebettete Systeme mit begrenzten Ressourcen geeignet.

Neben einem entfernten Methodenaufruf mit vorhersagbaren Zeitgrenzen [9, 5], der auf das strikte Dienstgeber/Dienstnehmer-Interaktionsmuster ausgerichtet ist, wird in HIJA ein echtzeitfähiger Nachrichtendienst mit Nachrichten-Benachrichtigungs-Muster [22] entwickelt. Zwei Versionen für harte und flexible (harte und weiche) Echtzeitsysteme basierend auf 100% purem Java mit der Echtzeiterweiterung RTSJ und mit einem geringen Prokolloverhead sind geplant.

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 Kommunikationsmodelle, die in Tabelle 1 aufgeführt sind. Das in HIJA entwickelte Modell unterstützt synchrone entfernte Aufrufe und eine asynchrone Nachrichtenkommunikation.

Diese Anforderungsanalyse für Kommunikationsmechanismen fokussiert sich auf asynchrone, nachrichtenbasierte Kommunikation. In den folgenden Paragraphen werden weitere Bewertungsmerkmale zur Charakterisierung dieses Modells eingeführt.

Persistenz vs. Transienz Persistente und transiente KommunikationSelbst asynchrone

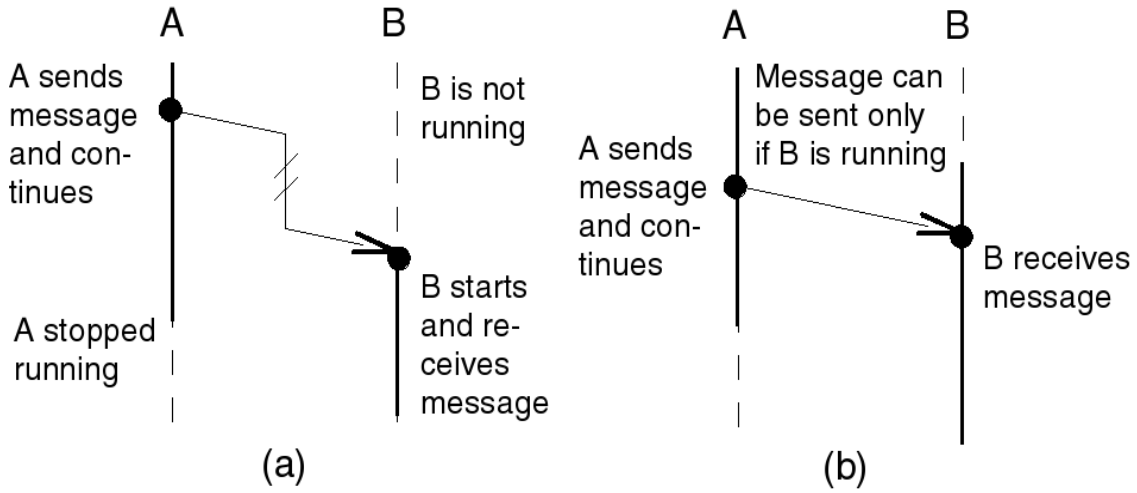

Abbildung 1: Persistente und transiente Kommunikation

Kommunikation mit Nachrichten erfordert Synchronisation beim Nachrichtenaustausch. Zusammenhängend damit kann die Persistenzstrategie beim Umgang mit Nachrichten unterschieden werden. Abbildung 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 zwischengespeichert werden. Im Gegensatz dazu bietet **transiente** Kommunikation keinerlei Zwischenspeicherung (b). Beide Varianten sind geeignet um eine asynchrone, nachrichtenbasierte Kommunikation mit Publiziere/Abonniere-Muster bereitzustellen.

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. DasPubliziere/AbonniereKommunikationsmuster erlaubt so die Entkopplung von Anbietern und Konsumenten in **Raum**, **Zeit** und **Synchronisation** [12].

- 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 so keine Referenzen aufeinander 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 so 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 führt in [19] zwei Basismodelle für die Nachrichtenübertragung bei Publiziere/Abonniere-Systemen ein: **"push"** und **"pull"**. Ohne Verlust der Allgemeingültigkeit sind beide Typen in Abbildung 2 ohne Entkopplung in Raum oder Zeit und mit jeweils nur einer Sender- und Empfängerinstanz dargestellt.

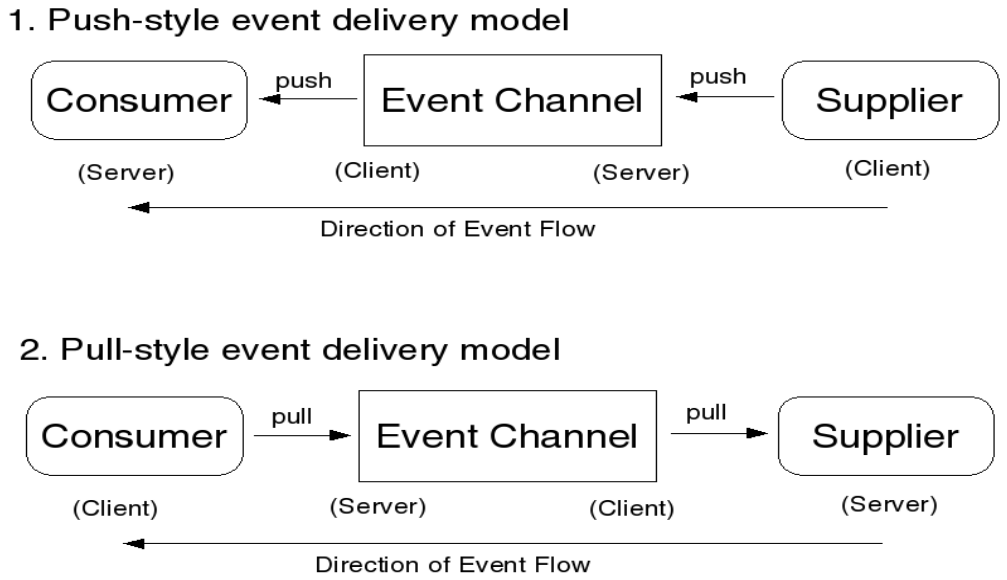

Abbildung 2: Push- und Pull-Nachrichtenübertragungsmodelle

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 [12] 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 Beschreibung von Gruppen und mit der Registrierung des Empfängers für ein Thema erhält er alle Nachrichten dieser Gruppe.

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

2. 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- oder 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.

Die weitere Bestimmung notwendiger Eigenschaften, um mit asynchroner Kommunikation das Feld der eingebetteten Systeme bei sicherheits- oder geschäftskritischen Diensten zu unterstützen, wird von der Garantie für Echtzeitigfähigkeit geleitet. Der folgende Abschnitt führt deshalb diesen Bereich und entsprechende Fachbegriffe und ihre Verbindung mit den Anforderungen für Abschnitt 3 näher ein.

2.2 Echtzeit

Echtzeitanforderungen werden im Rahmen von Rechen- bzw. Verteilungsmodellen für verteilte sicherheits- oder geschäftskritischen Systeme durch Einhaltung von Zeitgrenzen garantiert. Aus diesen Anforderungen an Rechtzeitigkeit und Erfüllung der Kommunikation in gegebenen Zeitgrenzen lässt sich nach [2] 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. Im Bereich der eingebetteten Systeme mit sicherheits- oder geschäftskritischen 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 im Jahr 2000 durch die Erweiterung in der "Real-Time Specification for Java" (RTSJ) für Echtzeitanwendungen geöffnet [4]. Der in diesem Artikel vorgestellte Lösungsansatz basiert auf RTSJ, sowie im Rahmen des HIJA Forschungsprojektes untersuchten Einschränkungen dieser Spezifikation. Diese Ergebnisse sind Teil der vorgestellten Laufzeitumgebung für den beschriebenen echtzeitigen Nachrichtendienst.

Kommunikation mit vorgegebenen Zeitgrenzen ermöglicht erst die Entwicklung verteilter sicherheits- oder geschäftskritischer eingebetteteter 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 eingebetteter Systeme und die dort existierende Vernetzung.

2.3 Netzwerk

Das in HIJA zu definierende Netzwerk-Modell versucht bei Verteilung in sicherheits- oder geschäftskritischen Applikationen dem Entwickler Transparenz zu gewährleisten. Dieser Abschnitt stellt eine übliche Infrastruktur und Vernetzung bei eingebetteten Systemen vor.

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 [14]. Neben einer physikalischen Verbindung besteht das Netzwerk aus Netzwerkzugangspunkten, über die einzelnen Knoten mit einem geschichteten Protokoll zuverlässig, sicher, effizienter und (bei Echtzeit-Netzwerken) mit Einhaltung von Zeitgrenzen kommunizieren können.

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.

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).

Automation: ProfiBus, Ethernet/IP In der Automation werden schon lange speziell entwickelte Feldbusse eingesetzt [31]. Neben einfacher Verkabelung gibt es hier aber auch einen Trend hin zur Standardisierung auf Basis von Internettechnologie wie TCP/IP.

Netzwerksysteme wie z.B. AFDX (ARINC664) [8] für deterministisches Ethernet oder andere Punkt-zu-Punkt-Verbindungsnetzwerke werden in dieser Analyse nicht weiter betrachtet, da der Mehraufwand in der Netzinfrastruktur komplexere Kommunikationssysteme wie z.B. Echtzeit-RMI erlaubt und den hier vorgestellten Ansatz asynchroner Nachrichtenkommunikation zwar nicht behindert, aber eben auch nicht fordert. Die in Abschnitt 4 vorgestellte Methodik ermöglicht verteilte eingebettete Applikationen für sicherheits- oder geschäftskritische Systeme mit geringeren Investitionen.

2.4 Verwandte Arbeiten

Verwandte Arbeit im Bereich plattformunabhängiger und architekturneutraler Kommunikation sind bei der Object Management Group (OMG) und ihrer Definition eines Objekte/Dienste-Informationsmodells für heterogene Applikationen in verschiedenen Sprachen und auf verschiedenen Plattformen zu finden. Mit der Echtzeiterweiterung der Common Object Request Broker Architektur (RT-CORBA) [18] werden Echtzeit- und sicherheits- oder geschäftskritische Systeme adressiert.

Die Entwicklung von RT-CORBA sieht ebenfalls einen auf Nachrichtenkommunikation basierenden Dienst als Erweiterung der synchronen Punkt-zu-Punkt-Kommunikation vor [17]. Dieser Dienst ist Teil der Object Request Broker (ORB) Funktionalität und erfordert so eine vollwertige CORBA-Infrastruktur. Aufgrund der TCP/IP-Orientierung von CORBA und der mächtigen plattformunabhängigen Konzepte muss der Ansatz für eingebettete Systeme kritisch betrachtet werden. Die eingeschränkten Ressourcen auf diesen Geräten und ein oft auf Nachrichtenversand an alle Komponenten ("broadcast") aufbauendes Kommunikationsnetzwerk, das dem Internet Inter-ORB Protokoll (IIOP) mit Punkt-zu-Punkt-Kommunikation nicht entspricht, passen nur mit zusätzlichem Synchronisationsaufwand zusammen.

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+) [6] wird auf einer Dienste-orientierten Mikrokern Architektur eine Auftragsbearbeitung eingeführt. Die zu komplexe Infrastruktur einer CORBA Middleware wird durch eine reduzierte Microkern-API ersetzt. Auf diesem Kern mit nur 6 Funktionen müsste eine objektorientierten Schnittstelle zur Nutzung in komplexeren sicherheits- oder geschäftskritischen Systemen erst noch implementiert werden.

3 Anforderungsdefinition

Beim Entwurf objektorientierter, verteilter sicherheits- oder geschäftskritischen Systeme muss der Entwickler derzeit über ein umfassendes Verständnis von komplexen Techniken, wie z.B. Ablaufkoordinierung und Speichermanagement, verfügen. Die Entwicklung im HIJA Forschungsprojekt 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. Im Bereich der Kommunikation ist der Einsatz einer "Middleware" für die Nutzung der Kommunikationsnetzwerke wünschenswert. Hier soll durch Bereitstellung einer Infrastruktur eine effiziente, flexible und skalierbare Nutzung erreicht werden. Dieser Abschnitt führt in die Voraussetzungen der Java Sprache als Laufzeitumgebung für eingebettete Systeme ein und kann so am Ende Anforderungen für die Entwicklung von objektorientierten, verteilten eingebetteten Systemen auf Basis von Echtzeit-Java beschreiben.

3.1 Nebenläufige Kontrollfäden und Ablaufkoordinierung

Die nebenläufige Ausführung von Kontrollfäden ist fest in die Java Sprache und Laufzeitumgebung integriert [13]. Der Einsatz von Java mit eingebetteten Systemen soll diese Eigenschaft für sicherheits- oder geschäftskritische Anwendungen mit Echtzeitbedingungen 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 [26]. 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 möglichst bald abgeben kann. Zur Vermeidung von möglichen Verklemmungen ("deadlocks") wird in [7] 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 sindAblaufkoordinierunsanalysen erforderlich, die dies garantieren können. Grundsätzlich besteht ein solches Verfahren aus drei Komponenten [33]:

- 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 [10].

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" [15]. Bei R-M-Ablaufkoordinierung wird die Priorität für Kontrollfäden abhängig von ihrer Periodenlänge berechnet[3]. Dieser Algorithmus ist optimal, dass falls ein möglicher Ablaufplan mit statischen Prioritäten existiert, wird einer auch durch den R-M-Algorithmus ermittelt. D-M-Ablaufkoordinierung bestimmt die Priorität abhängig von der erlaubten Zeitgrenze und für Zeitgrenze gleich Periodenlänge entspricht dieser Ansatz der Lösung mit R-M.

Eine detailliertere Einführung in den Einsatz der RTSJ für nebenläufige Programmierung bietet z.B. [32].

3.2 Anforderungen

Für die Entwicklung verteilter sicherheits- oder geschäftskritischen Systeme mit Garantien in Echtzeit ist insbesondere Zuverlässigkeit und Planbarkeit der Systeme ausschlaggebend. Das Forschungsprojekt HIJA definiert hier EDV- und Netzwerk-Modelle für den Entwurf architekturneutraler, echtzeitfähiger Systeme ("Architecturally Neutral high-integrity Real-Time Systems", ANRTS). Java mit der Echtzeiterweiterung RTSJ ist Basis für diese Entwicklung.

Der in diesem Artikel vorgestellte themenbasierte, asynchrone und direkte Publiziere/Abonniere-Nachrichtendienst bietet aufbauend darauf Möglichkeiten für Echtzeitkommunikation, angepasst an verteilte eingebettete Systeme. Dies erfordert neben garantierter Rechtzeitigkeit des Nachrichtenaustauschs[4] insbesondere die Erbringung einer vorhersagbaren Dienstqualität. Dienstqualität ("Quality of Service", QoS) ist hier eine multidimensionale Bewertungsfunktion die sich über Parameter wie Verzögerung, Verlust, Aufrufblockierungswahrscheinlichkeit, Jitter, Ende-zu-Ende-Kommunikationszeit und verfügbarem Durchsatz bestimmt.

3 Je kürzer die Periode, desto höher die Priorität.
4 für harte Echtzeitanforderungen

Die Echtzeitgarantien eines unterliegenden Kommunikationsnetzwerkes oder -feldbussystems sollen ohne Prokolloverhead genutzt werden. Die Beschränkung der Ausführungszeit für den Nachrichtenversand und die entfernte Interpretation soll durch eine flexible Beschreibung von Echtzeitparametern des aufrufenden und aufgerufenen Prozesses in einer netzwerkunabhängigen Implementierung des Nachrichtendienstes ermöglicht werden.

4 Methodik und Kommunikationsinfrastruktur

Für die Entwicklung von ANRTS Applikationen wird die Java Sprachspezifikation [13] 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. Dieser Abschnitt beschreibt eine asynchrone Kommunikationsinfrastruktur und führt eine grundsätzliche Methodik beim Entwurf verteilter, sicherheits- oder geschäftskritischen Systeme ein.

4.1 Laufzeitumgebung

Der Einsatz von Java mit verteilten, sicherheits- oder geschäftskritischen Systemen erfordert neben Echtzeitnetzwerk, dem Prozessor zur Ausführung der Applikationslogik und einem direkten Zugriff auf Sensoren und Aktoren eine Java-VM mit Unterstützung von Echtzeitgarantien und den beschriebenden Algorithmen der Festprioritätsablaufkooridierung.

RTSJ Einschränkungen Die RTSJ Erweiterung für die Java Plattform ermöglicht die Ausführung von Java-Programmen mit Echtzeitanforderungen. Für diesen Zweck wurde das Zusammenspiel mit der automatischen Speicherverwaltung[5] mit Einführung neuer Speicherbereiche reduziert, um auch ohne Speicherverwaltung Speicherlecks bei der Programmierung zu verhindern. HIJA schränkt die Möglichkeiten der RTSJ für harte Echtzeitanforderungen weiter ein. Um eine statische Analyse der Ablaufkoordinierung für Kontrollfäden in Systemen unter harten Echtzeitanforderungen berechnen zu können, werden in HIJA nur der dauerhafte ("immortal") Speicherbereich und kurzlebige "scoped" Speicherbereiche mit linearer Allokationszeit, die nur von einem Kontrollfaden betreten und nicht gestapelt werden, erlaubt. Kontrollfäden selbst sind nur mit periodischer und sporadischer[6] Ausführungform vorgesehen, da für aperiodische Aktivitäten ohne die Vorhersagbarkeit einer Mindestzwischenzeit die Analyse erst zur Laufzeit und nicht statisch im voraus möglich ist.

Für die Garantie harter Echtzeitanforderungen werden getrennte Phasen für Initialisierung und Aufgabenausführung ("mission") [21] definiert, nach der Initialisierungsphase sind alle Kontrollfäden für die Durchführung der folgenden Aktivitäten erzeugt.

Beim Einsatz von Netzwerkkommunikation wird auf die Standard-Objektserialisierung zugunsten einer auf Speicherplatz und Rechenaufwand optimierten Serialisierung der Inhalte von Objekten verzichtet, auch beim Austausch von Informationen zwischen Kontrollfäden innerhalb einer VM werden nur primitive Typen und wiederverwendete Objektstrukturen eingesetzt.

4.2 Software Entwurfsmuster

Die so definierten Rahmenbedingungen für die asynchrone, themenbasierte Nachrichtenkommunikationsinfrastruktur erlauben den objektorientierten Entwurf verteilter Applikationen für Regel- und Automationssysteme.

Um dem Gegensatz zwischen "asynchron" und garantierten Zeitgrenzen an beiden Enden der

5 Stichwort "Garbage Collection"
6 aperiodisch bei einer garantierten Mindestzwischenzeit

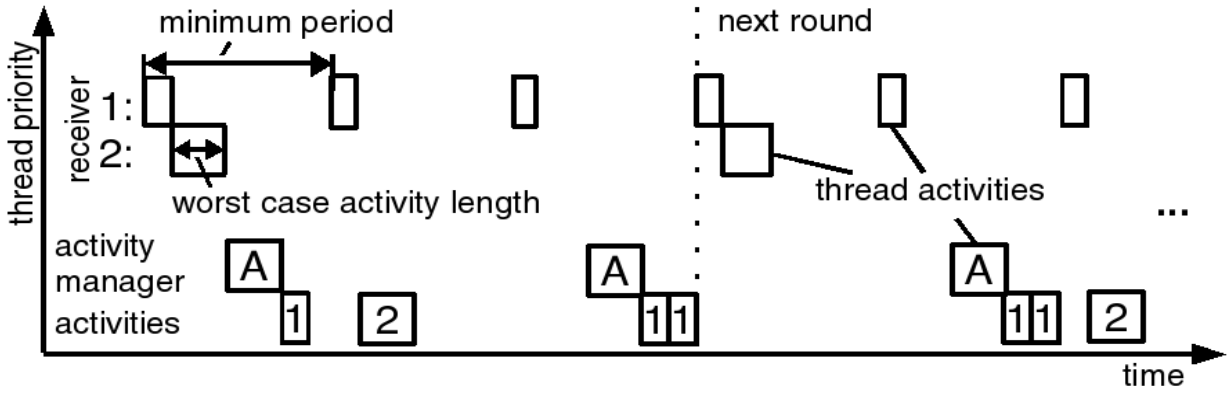

Abbildung 3: Ablaufplan

Kommunikation zu begegnen hat [23] ein Entwurfsmuster mit Empfängerkollektiv, einem Manager und Warteschlangen definiert. Zur Erfüllung von harten Echtzeitanforderungen wird das dort beschriebene Entwurfsmuster mit den RTSJ Einschränkungen implementiert. Da sich dieser Artikel hierauf konzentriert, ist für den Einsatz unter weicher Echtzeit und die Behandlung möglicher Fehlerzustände auf [24] verwiesen.

Infrastrukturkomponenten Hier werden die zugrundeliegenden Konzepte aus [23] nocheinmal zusammengefasst und in den Zusammenhang mit Methodik bei Systementwurf und Einschränkungen der Laufzeitumgebung speziell für harte Echtzeitanforderungen gestellt.

Im Publiziere/Abonniere-Nachrichtendienst wird das Objekt eines Kanals (EventChannel) als Thema für die Gruppierung von Nachrichten verwendet. Die Gruppierung schließt neben Informatonen über die Periode[7] der zu sendenden Nachrichten auch zeitliche und Performanzanforderungen an die Abonnentenseite ein. Die Kommunikation erfolgt über einen Byte-orientierten BroadcastSocket, der den direkten Zugangspunkt zum darunterliegenden Echtzeitnetzwerk darstellt. Zugriff darauf hat eine Transmitterkomponente, sowie je Netzwerkzugangspunkt ein PushEventReceiver Kontrollfaden, der mit höchster Priorität die empfangenen Daten des PrimtiveRemoteEvent direkt vom Netz oder aus einem Hardwarepuffer des Controllers liest. Die empfangenen Nachrichtenobjekte werden zur weiteren Behandlung durch registrierte PushEventHandler in ein nach Prioritäten sortiertes Warteschlangensystem eingereiht. In der vorgesehenen Initalisierungsphase werden die Empfänger sowie ein Manager (ActivityManager) zur Verwaltung eingegangender Nachrichten und die in einem Pool auf die Ausführung wartenden Aktivitätskontrollfäden definiert und gestartet.

Integration in Analyse für Ablaufkoordinierung Für die Ausführungsphase müssen die Kontrollfäden in einen Ablaufplan, der alle Zeitgrenzen einhält, integriert werden. Für den Nachrichtendienst muss die Ablaufkoordinierungsanalyse um eine Untersuchung der beteiligten Kanäle und Netzwerzugangspunkte erweitert werden. Abbildung 3 illustriert einen möglichen Ablaufplan mit 2 Kontrollfäden für den Empfang, einem Manager und 2 Kontrollfäden zur Ausführung der registrierten Aktivitäten.

Die Berechnung des Ablaufplans erfordert die Abstimmung von Kontrollfäden, Warteschlangen und Empfangspuffern der Netzwerkzugangspunkte. Aus den statisch verfügbaren Informationen über die minimale Periode der Kontrollfäden (Empfänger, Manager), Kapazität für Warteschlangen, sowie der längsten Ausführungszeit für alle periodischen und sporadischen Aktivitäten wird eine Runde bestimmt. Innerhalb einer Runde müssen sämtliche empfangene Nachrichten abgearbeitet und die Warteschlangen in der Kommunikationsinfrastruktur und die Puffer der Netzwerkzugangspunkt wieder auf einem initialen Zustand sein.

Diese Berechnung ergibt zusammen mit der Ablaufkoordinierungsanalyse für Kontrollfäden mit festen Prioritäten einen Ablaufplan, der harte Zeitanforderungen an die verteilte Applikation

7 oder auch sporadischen Charakter

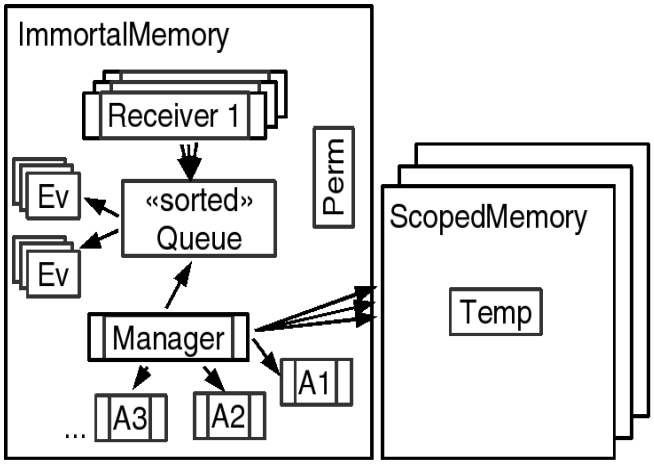

Abbildung 4: Speicher und Kontrollfäden

garantieren kann.

Implementierung mit RTSJ Speicher und KontrollfädenZur Implementierung der Kommunikationsinfrastruktur nutzt der Nachrichtendienst die Möglichkeiten der eingeschränkten RTSJ. 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.

In der Abbildung 4 werden diese dauerhaften Datenobjekte mit ihren Beziehungen zu den notwendigen Kontrollfäden dargestellt. Neben dem dauerhaften Speicherbereich 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.

Applikationsentwurf Der so definierte asynchrone, direkte, themenbasierte Publiziere/Abonniere-Kommunikationsdienst greift direkt auf Datensicherungs- oder Transaktionsschicht[8] des unterliegenden Netzwerkes zu und verwendet für den Versand von Informationen einzelne Datenpakete oder möglichst wenige Datenzellen des Netzwerkprotokolls.

Die angebotene Eins-zu-viele-Kommunikation lässt sich leicht in objektorientierten, verteilten Applikationen für eingebettete Systeme nutzen. Das Erfordernis für statischer Analyse und Verifikation erlaubt auch eine deklarative Beschreibung von Knoten und Kommunikationsmechanismen der verteilten, sicherheits- oder geschäftskritischen 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="socket1" 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.

8 Schicht 4 in ISO/OSI

```
11: <server name="Server1">
12:  <channel name="Channel42" id="42">
13:   <socket-ref>socket1</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 Netzwerks 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>
```

5 Zusammenfassung und Ausblick

Durch das in diesem Artikel beschriebene Kommunikatonsmodell auf Basis eines asynchronen, direkten, themenbasierten Publiziere/Abonniere-Nachrichtendienstes wird die objektorientierte Softwareentwicklung für eingebettete verteilte sicherheits- oder geschäftskritischen Echtzeitsysteme unterstützt. Die Darstellung notwendiger Fachbegriffe und Paradigmen bei der Entwicklung solcher Systeme unter der Java-Echtzeiterweiterung mit Nutzung von vorhandenen COTS-Netzwerk- und Bussystemen im Bereich eingebetteter Systeme wird erfolgreich adressiert und ein Softwareentwurfsverfahren für Anwendungen mit harten Zeitgrenzen und statischer Analyse und Verifikation ermöglicht.

Die vorgestellte Methodik und das definierte Softwareentwurfsmuster werden derzeit auf Basis des HIJA Forschungsprojektes umgesetzt. Für flexible (weiche und harte) Echtzeitanforderungen wird eine dynamische Version entwickelt. Wesentliche Änderungen des bisher vorgestellten Softwareentwurfsmuster gibt es dabei nicht, es werden nur die Möglichkeiten der vollen RTSJ-Implementierung genutzt und der Dienst wird um dynamische Aspekte zur Ankündigung, Aushandlung und Entdeckung von Nachrichtenkanälen und Ereignissen erweitert. Eine mögliche Fehlerbehandlung von Zielverletzungen wird mit einem Java-Ausnahmenmodell geboten.

Literaturverzeichnis

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

[2] M. H. Klein, T. Ralya, B. Pollak, R.Obenza, and M. G. Harbour. *A Practitioner's Handbook for Real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, 1993.

[3] N. C. Audsley, A. Burns, M. F. Richardson, and A. Wellings. Hard Real-Time Scheduling:

The Deadline-Monotonic Approach. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems*, 1991.

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

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

[6] U. Brinkschulte, A. Bechina, F. Picioroaga, and E. 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.

[7] A. Burns and A. Wellings. R*eal-Time Systems and Programming Languages. Ada 95, Real-Time Java and Real-Time POSIX*. Addison-Wesley, third edition, 2001.

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

[9] M. 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.

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

[11] W. 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.

[12] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114-131, June 2003.

[13] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification* . Addison-Wesley, 3rd edition, 2005.

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

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

[16] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

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

[18] Object Management Group, Inc. *Real-Time CORBA Specification*, version 1.1, formal/02-08-02 edition, August 2002.

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

[20] Project Partners. *D8.1 – HIJA White Paper*. Technical report, The Open Group, June 2005.

[21] P. Puschner and A. 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.

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

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

[24] M. Schanne and D. J. J. Hunt. *Remote Event Service Design*. Technical report, FZI Forschungszentrum Informatik, 2004. Deliverable D4.2 describing the HIDOORS event channel network.

[25] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-oriented Software Architecture. Patterns for Concurrent and Networked Objects*, volume 2. John Wiley & Sons Ltd., April 2001.

[26] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, 39(9):1175-1185, September 1990.

[27] N. Storey. *Safety-critical Computer Systems*. Addison-Wesley, 1996.

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

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

[30] A. S. Tanenbaum and M. van Steen. *Distributed Systems. Principles and Paradigms*. Prentice Hall, Inc, 2002.

[31] J. P. Thomesse. A review of the fieldbuses. *Annual Reviews in Control*, 22:35-45, 1998.

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

[33] A.Wellings, G. Bollella, P. Dibble, and D. 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.