Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Verteilte Systeme

Diplomarbeit

Gateway zwischen zeit- und ereignisgesteuerten Kommunikationsmodellen

Verfasser:

Stefan Metzlaff

8. November 2006

Betreuer:

Prof. Dr. rer. nat. Jörg Kaiser

Dipl. Ing. Sebastian Zug

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg
Germany

Metzlaff, Stefan:
Gateway zwischen zeit- und ereignisgesteuerten Kommunikationsmodellen
Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 2006.

Abstract

For the communication of embedded systems two paradigms can be distinguished: the event-triggered and the time-triggered communication. Both models use distinct approaches for media access. Thereby the models have different characteristics. The event-triggered way provides a flexible communication, which is hardly predictable. For the time-triggered approach an a priori statically defined communication plan is created. The scope of this diploma thesis is to combine the advantages of both paradigms in one heterogeneous system. Therefore a gateway is designed, which allows the communication between both domains. An appropriate middleware can hide this heterogeneity of the distributed application and ensure the communication needs. Thus the Publish/Subscribe middleware COSMIC is used to enable transparent communication and to enforce the quality constrains for the traffic between the different systems.

The designed concept enables the coupling of CAN and TTP/C networks. The use of COSMIC allows an event channel based communication with a global addressing scheme. Therefore the COSMIC mechanisms are integrated into the TTP/C domain. The gateway between CAN and TTP/C controls the information flow and minimizes passing traffic by forwarding subscribed event channels only. The presented concept enables the use of the COSMIC real-time classes. The inspection of communication delays and jitters for each real-time class allows the planning of inter domain event channels. Large communication jitters will occur because of the mapping of event-triggered asynchronous messages to the periodic time slots in the TDMA scheme of the time-triggered protocol. These jitters cannot be eliminated, if no time synchronization is established between both domains. If a global time base is used synchronous hard real-time event channels minimize the jitter nearly to zero.

The implemented prototype of the gateway is able to serve non real-time event channels only. An optimized gateway can be created for each application by various configuration possibilities. The event channel concept of COSMIC is implemented in the TTP/C domain. It provides arbitrary slots and sporadic messages. The measured end-to-end latency for non real-time messages from CAN to TTP/C has a jitter caused by the mapping to the cyclic time slots. The period of the processing tasks for incoming CAN messages defines the jitter. Thus it is larger than the latency caused by the gateway. The measured latency in reverse direction is not affected by jitter, if the CAN bus is not blocked by other messages. Thus the jitter in this direction is caused by the CAN bus access only and not by the conversion of messages.

An example based on a mobile robot describes the capabilities of systems which couples event- and time-triggered communication. Two subsystems are created, one for time-critical and the other for non-critical communication. The time-critical subsystem performs a distributed motor control and a collision avoidance of the robot. Due to the characteristics of this subsystem, it is located in the time-triggered TTP/C. The navigation of the robot by external control is defined as non-critical. Thereby the navigation uses the CAN protocol. The example shows the usability of the gateway prototype. The separation of navigation from the control application delivers an extendable mobile robot system.

Danksagung

An dieser Stelle bedanke ich mich bei Prof. Dr. Kaiser für die Bereitstellung des Themas meiner Diplomarbeit sowie für viele hilfreiche Hinweise und bei Prof. Dr. Paul für die Übernahme des Zweitgutachtens. Mein Dank geht auch an die Arbeitsgruppe EOS, welche mir bei fachlichen Fragen weiterhalf und mir mit vielen Tipps und Anregungen bei der Diplomarbeit geholfen hat. Besonders hervorheben möchte ich Sebastian Zug, Michael Schulze und Jürgen Lehmann. Weiterhin danke ich allen Personen, die mir fachliche oder moralische Unterstützung gegeben haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Abkürzungen

CAN	Controller Area Network
CMS	Control Message Slot
CNI	Communication Network Interface
COI	Controlled Object Interface
CORBA	Common Object Request Broker Architecture
COSMIC	COoperating SMart devICes
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
ECB	Event Channel Broker
ECH	Event Channel Handler
EDF	Earliest Deadline First
ETB	Event Toggle Bit
FTU	Fault-Tolerant Unit
HRT	Hard Real-Time
IP	Internet Protocol
LIN	Local Interconnect Network
MTC	Message Transport Channel
NRT	Non Real-Time
ORB	Object Request Broker
RMI	Remote Method Invocation
RODL	Round Description List
RTE	Real-Time Entity
RTI	Real-Time Image
RTO	Real-Time Object
SOC	Sphere Of Control
SRT	Soft Real-Time
TCP	Transmission Control Protocol
TDMA	Time Division Multiple Access
TTA	Time Triggered Architecture
TTP	Time Triggered Protocol
UDP	User Datagram Protocol
WCET	Worst Case Execution Time

Kapitel 1

Einleitung

Im Laufe der Entwicklung von eingebetteten Systemen zur Steuerung und Regelung von Prozessen und Anlagen veränderte sich ihre Systemarchitektur. Anfangs gab es große Einzelsysteme, welche die Aufgabe zentral lösten und die zum Teil entfernte Peripherie befehligten. Dabei erfolgte die Kommunikation über digitale oder analoge Ein- und Ausgaben. Es konnten einfache Sensoren und Aktoren über eine direkte physische Verbindung vom zentralen System aus abgefragt bzw. angesprochen werden. Die hohen Kosten dieser Anlagen und die Entwicklung preisgünstiger und leistungsfähiger Hardware führte zu einer zunehmenden Dezentralisierung dieser Systeme [Rei02]. Es entwickelte sich eine Architektur von verteilten kooperierenden Komponenten, welche einzelne Teilfunktionen der Gesamtaufgabe ausführen. Um eine erfolgreiche Kopplung der Teilkomponenten zu einem Gesamtsystem zu erreichen, muss die Kommunikation zwischen den Komponenten verschiedenen Anforderungen z.B. an die Bandbreite, Echtzeitfähigkeit, Vorhersagbarkeit, Fehlererkennung und -toleranz Rechnung tragen.

Über die Zeit entwickelten sich unterschiedliche Kommunikationsverfahren im verteilten Gesamtsystem. Für die Kommunikation in dezentralen Systemen wurden sogenannte Feldbusse eingesetzt. Dabei entstanden verschiedenste Protokolle, welche unterschiedliche Kommunikationsmodelle zum Medienzugriff verwenden. Es werden zwei Modelle im Bereich der Kommunikation eingebetteter verteilter Systeme angewendet: die ereignis- und die zeitgesteuerte Kommunikation.

Die ereignisgesteuerte Kommunikation folgt der Prämisse nur dann mit anderen Teilsystemen in Kontakt zu treten, wenn ein zu verbreitendes Ereignis eintritt. Die zeitgesteuerte Kommunikation hingegen sendet periodisch Nachrichten, unabhängig davon, ob neue Informationen vorliegen. Die zeitgesteuerte Kommunikation ist daher vorhersagbar und für sicherheitskritische Anwendungen besonders geeignet. Im Gegensatz dazu ist die ereignisgesteuerte Kommunikation einfach und flexibel, aber nur unter strengen Bedingungen und mit großen Einschränkungen vorhersehbar. Durch die Anforderungen der zeitgesteuerten Kommunikation an die Kommunikationsteilnehmer, wie z.B. einen genauen Zeitgeber, ist sie im Vergleich zu ereignisgesteuerten Komponenten, welche kaum Ansprüche stellen, kostenintensiv. Zeitgesteuerte Kommunikation wird daher hauptsächlich in sicherheitskritischen Systemen wie z.B. im Flugzeug (fly-by-wire) eingesetzt, während sich die ereignisbasierte Kommunikation in weniger kritischen Systemen z.B. im automotiven Bereich (Karosserie- und Komfortelektronik) verbreitet hat. Aufgrund der unterschiedlichen Eigenschaften haben beide Modelle ihre Vor- und Nachteile.

Verteilte Systeme erschließen immer neue Anwendungen. Sie werden dabei zunehmend komplexer und vereinen unterschiedliche Teilanwendungen, so dass es sinnvoll und wirtschaftlich ist, Teilsysteme beider Kommunikationsmodelle zu kombinieren. Ein Beispiel dafür ist die Steuerung eines Fahrzeuges (*x-by-wire*) ohne mechanische oder hydraulische Rückfallsysteme. Jene sicherheitskritischen Anwendungen erfordern den Einsatz von verteilten Systemen in neuen Umgebungen, dabei übernimmt aber nur ein Teil des Gesamtsystems die sicherheitskritischen Aufgaben. Ein Einsatz der zeitgesteuerten Kommunikation im gesamten System ist daher nicht nötig und zu kostenintensiv. Zudem gibt es z.B. im Automobilbereich viele fertige Lösungen für Teilprobleme, welche bereits ereignisgesteuert realisiert sind und deren Neuentwicklung zu kostenaufwändig ist [Obe05]. Es liegt daher nahe, die Vorteile beider Kommunikationsmodelle in einem Gesamtsystem zu vereinen. Zwei Möglichkeiten für die Kopplung der Kommunikationsmodelle können unterschieden werden: entweder eine integrierte Kommunikation, die beide Charakteristika unterstützt, oder heterogene Teilsysteme werden mit Gateways verbunden. Mit einer Kopplung der Kommunikationsmodelle durch ein Gateway ist es möglich bereits entwickelte Teilanwendungen weiterzunutzen und deren Ergebnisse in einem anderen Kommunikationsmodell zur Verfügung zu stellen. Dabei können extern nicht benötigte Informationen in den jeweiligen Teilsystemen gekapselt werden.

1.1 Zielsetzung

Das Ziel der Arbeit ist die Realisierung eines Gateways zwischen ereignis- und zeitgesteuerter Kommunikation. Dabei wird der Fokus auf folgende Punkte gelegt:

- **Globales Adressierungsschema**
- **Kontrolle des Informationsflusses**
- **Unterstützung von Echtzeitkommunikation**

Ein systemweit konsistentes Adressierungsschema für Nachrichten in allen Netzen erfordert die Nutzung einer geeigneten Middleware. Sie ermöglicht die Adressierung der Information über die Netzgrenzen hinweg, sichert die korrekte Übermittlung der Nachrichten zu und erlaubt so eine transparente Kommunikation in einem heterogenen föderierten System.

Ein Gateway zwischen einem ereignis- und einem zeitgesteuerten Netz kann den Informationsfluss zwischen den Netzen kontrollieren. Dadurch kann es die Kommunikation zwischen den Netzen minimieren und Bandbreite in beiden Netzen einsparen. Es ergibt sich damit die Forderung, dass nur die Nachrichten das Gateway passieren dürfen, welche im anderen Netz auch tatsächlich benötigt werden.

Eine weitere Anforderung an das Gateway ist die Unterstützung von Echtzeitkommunikation. Dazu muss die Überwachung und Einhaltung der Zeitparameter der Informationen sichergestellt werden. Gerade in zeitgesteuerten Systemen spielt die Gültigkeit einer Meldung eine enorme Rolle, denn diese Systeme werden zumeist dort eingesetzt, wo die zeitlichen Parameter sehr streng sind und deren Verletzung katastrophale Folgen haben kann. Das Gateway muss, soweit es das Kommunikationsmedium ermöglicht, in beiden Netzen sicherstellen, dass alle Informationen zeitgerecht ihr Ziel erreichen. Sollte dies aufgrund von Überlastsituationen oder Fehlern nicht realisierbar sein, darf es veraltete Informationen nicht weiterleiten, da

dies die Übermittlung noch gültiger Informationen blockieren kann. Eine Middleware kann das Gateway insoweit unterstützen, in dem sie die Informationen nach ihrer Dringlichkeit einteilt und Kommunikationskanäle für verschiedene Echtzeitklassen bereitstellt. So können wichtigere Informationen vom Gateway bevorzugt zugestellt werden.

1.2 Anwendungsszenario

Zur Evaluierung des Gateways wird im Rahmen der Arbeit ein Anwendungsszenario erstellt. Es zeigt eine Anwendungsmöglichkeit für ein verteiltes System, welche die Eigenschaften von zeit- und ereignisgesteuerter Kommunikation nutzt. Die Basis des Szenarios ist ein mobiler Roboter mit verteilter Motorsteuerung und einfachen Sensoren. Die Idee ist, die zeitkritische Kommunikation in einem zeitgesteuerten und die unkritische Kommunikation in einem ereignisgesteuerten Netz zu realisieren.

In einem mobilen Roboter kann ein schwerwiegendes Fehlverhalten wie z.B. Zusammenstöße oder unkorrekte Lageregelung eintreten, wenn Kollisionswarnungen oder Motorsteuerungsbefehle nicht rechtzeitig erhalten werden. Daher ist die Kollisionsvermeidung bzw. -erkennung und die Motorsteuerung als zeitkritisch anzusehen. Die Navigation mittels Kartenerstellung, Wegplanung oder externer Steuerung gilt im dargestellten Anwendungsszenario als nicht zeitkritisch. Verzögerungen in diesem Teilsystem führen dazu, dass der Roboter einen Fahrauftrag nicht rechtzeitig erhält und solange wartet bis ein Befehl von der Navigation eintrifft.

Zur Verbindung der Kommunikationsnetze wird ein Gateway benötigt, dass die Steuerbefehle der Navigation an die im zeitgesteuerten Netz angesiedelte verteilte Motorsteuerung übermittelt. Neben der Weiterleitung der Fahrbefehle, muss das Gateway relevante Informationen von der Motorsteuerung an die Navigation vermitteln. Solche Informationen können die aktuelle Geschwindigkeit, den bereits zurückgelegten Weg und Kollisionswarnungen sein. Die Trennung der Anwendung in zwei Teilsysteme ermöglicht eine robuste deterministische Motorsteuerung sowie eine dynamisch erweiter- und veränderbare Navigationskomponente.

1.3 Gliederung

Die Grundlagen sowie einführende Begriffe werden in Kapitel 2 dargestellt. Das Kapitel enthält eine Gegenüberstellung der Kommunikationsmodelle, die Vorstellung verschiedener Kommunikationsprotokolle und Middleware-Systeme sowie die Beschreibung der Funktionen eines Gateways. Das Kapitel 3 beschreibt die Kopplung von ereignis- und zeitgesteuerter Kommunikation. Es werden bestehende Lösungen vorgestellt und ein eigenes Konzept für das Gateway entwickelt. Die prototypische Umsetzung des Konzeptes wird in Kapitel 4 dargestellt. Dabei wird speziell auf die Konfigurierbarkeit und die Programmierschnittstelle für die Anwendung eingegangen. Die Evaluation der Implementierung wird in Kapitel 5 durchgeführt. Anhand von Messungen der Verzögerungszeit, welche durch die modellübergreifende Kommunikation entsteht, wird das zeitliche Verhalten des Gateways untersucht. Weiterhin erfolgt die funktionale Evaluation der Gateway-Implementierung durch das beschriebene Anwendungsbeispiel. Das Kapitel 6 beendet die Arbeit mit einer Zusammenfassung der Ergebnisse und einem Ausblick.

Kapitel 2

Grundlagen

Die Grundlagen, welche für die konzeptuellen Betrachtungen sowie für die Beschreibung der Umsetzung benötigt werden, werden in diesem Kapitel beschrieben. Zu Beginn werden in Abschnitt 2.1 einige in der Arbeit verwendete Begriffe dargestellt. Der zweite Abschnitt dieses Kapitels erläutert die Kommunikationsmodelle. Der sich anschließende Abschnitt 2.3 stellt verschiedene ereignis- und zeitgesteuerte Kommunikationsprotokolle dar. Hierbei wird speziell auf das ereignisgesteuerte CAN-Protokoll und auf das zeitgesteuerte TTP/C-Protokoll eingegangen. Die Definition sowie die Funktionen eines Gateways werden im Abschnitt 2.4 beschrieben. Der letzte Abschnitt 2.5 stellt einige Middlewaresysteme vor.

2.1 Begriffe

Bevor die Kommunikationsmodelle beschrieben werden, wird kurz eine Begriffsbasis aufgebaut, auf die im Folgenden zurückgegriffen werden kann.

Verteiltes System

Ein verteiltes System ist laut [Tan03] eine Ansammlung unabhängiger Systeme, welche so zusammenarbeiten, dass sie dem Nutzer als kohärentes Einzelsystem erscheinen. Das verteilte System besitzt ein Modell mit dem es sich dem Nutzer präsentiert. Dieses Modell wird dabei oft durch verschiedene Softwareschichten, die auf das Betriebssystem aufsetzen, realisiert. Diese Software wird auch als **Middleware** bezeichnet.

Verteiltes Eingebettetes System

Handelt es sich bei den unabhängigen Einzelsystemen um eingebettete Systeme, wird der Begriff **verteiltes eingebettetes System** genutzt. Dabei gehen die speziellen Eigenschaften der eingebetteten Systeme in das verteilte eingebettete System mit ein. [Kop97] beschreibt ein eingebettetes Echtzeitsystem als einen Teil eines wohlspezifizierten größeren Systems. Ein eingebettetes System interagiert meist mit einem mechanischen Teilsystem und ist daher auf einem **Mikrocontroller** (μController) implementiert. Eingebettete Systeme besitzen nach [Kop97] eine Reihe von Eigenschaften:

- hohe Produktionsstückzahl
- fest definierte Funktionen und statische Struktur
- Mensch-Maschine Schnittstelle
- minimales mechanisches Teilsystem
- Funktionalität auf Read-Only Memory
- wartungsfreies Gesamtprodukt
- Kommunikationsfähigkeit

Es gibt auch eingebettete Systeme, welche nicht alle diese Eigenschaften besitzen. So wird z.B. das Steuerungssystem zur Bahnkorrektur [LR05] der Internationalen Raumstation nicht in großen Stückzahlen produziert. Daher beschreibt [Hea03] ein eingebettetes System allgemeiner: Ein eingebettetes System ist demnach ein System, welches entwickelt wurde, um eine spezifische Aufgabe zu lösen. Die Funktion eines eingebetteten Systems kann vom Nutzer gesteuert und/oder konfiguriert werden. Es ist jedoch nicht möglich, dass der Nutzer die Funktionalität eines eingebetteten Systems so verändern kann, dass es eine andere Aufgabe übernimmt. Nach [Nau05] sind eingebettete Systeme Systeme, die spezifische Aufgaben für übergeordnete Anwendungen durchführen und als eigenständige Geräte oder Geräteteile konstruiert sind.

Unterliegt ein eingebettetes System zeitlichen Abhängigkeiten, d.h. Aufgaben, Berechnungen oder Nachrichtenübertragungen müssen bis zu einer gewissen Zeit beendet sein, bezeichnet man es als eingebettetes **Echtzeitsystem**.

Echtzeitsystem

Echtzeitsysteme sind Systeme, welche ihre Ergebnisse innerhalb einer gesetzten Zeit ermitteln müssen. Nach DIN 44300 ist der Begriff Echtzeit wie folgt definiert:

> *Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen.*

Es gibt in der Literatur [Sch05, Kop97, KS97] zwei Klassen von Echtzeitsystemen: harte und weiche Echtzeitsysteme. In [Coo03] wird jede Klasse noch in zwei Unterklassen aufgeteilt. Kriterium dafür ist die Dauer der Gültigkeit bis zum Eintreten der zeitlichen Bedingungen (*slow and fast systems*). Dieser Unterteilung wird hier nicht nachgegangen, da dies prinzipiell keinen Unterschied macht, wie schnell die zeitlichen Beschränkungen erreicht sind. Harte Echtzeitsysteme (*HRT System, hard real-time system, saefty-critical real-time system*) bezeichnen Systeme bei denen es unbedingt erforderlich ist, dass eine Berechnung oder eine Aktion vor der zeitlichen Begrenzung (Frist, *Deadline*) abgeschlossen ist. Ein Verletzen der Deadline kann erhebliche Folgen nach sich ziehen. Beispiele für harte Echtzeitsysteme sind z.B. x-by-wire Systeme, bei denen das Verpassen der Deadline sicherheitskritische Folgen hat. Das Überschreiten

der Deadline ist bei weichen Echtzeitsystemen (*SRT System, soft real-time system*) nicht sicherheitskritisch, dennoch sinkt der Wert des Ergebnisses nach dem Verletzen der Deadline abhängig von der Zeit. Die Deadlines sollten in solchen Systemen daher nur gelegentlich Überschritten werden. Ein Beispiel für ein weiches Echtzeitsystem ist ein Programm zum Abspielen von Videos. Als zeitliche Anforderung hat dieses System die Einhaltung einer gewissen Framerate (in Bilder pro Sekunde). Wird sie nur gelegentlich um einen kleinen Wert verringert fällt dies dem Nutzer zwar auf, das Multimediasystem erfüllt aber dennoch korrekt seine Aufgabe. Sollte es jedoch zu häufigen starken Überschreitungen der Deadline kommen, kann die Framerate über einen längeren Zeitraum nicht erreicht werden oder deren Wert sinkt zeitweise rapide ab. In beiden Fällen ist das Multimediasystem für den Anwender unbrauchbar.

Systeme die keinen strengen zeitlichen Anforderungen unterliegen, im Allgemeinen Systeme mit Nutzerinteraktion, können als Nicht-Echtzeitsysteme (*NRT System, non real-time system*) bezeichnet werden.

Knoten

Ein Knoten eines verteilten Systems ist ein eigenständiges System mit eigener Hardware und Software, welches über das Kommunikationsmedium mit dem Rest des verteilten Systems verbunden ist. Damit ist der Knoten, laut [Kop97], eine wichtige Abstraktion in verteilten Systemen, weil er Software- und Hardwarebestandteile direkt an eine funktionsfähige Einheit bindet. Ein Knoten hat ein beobachtbares Verhalten im Zeit- und Wertebereich. Ein Knoten besteht nach [Kop98] aus einem Applikationsrechner (*host computer*), einem Kommunikationssystem (Kommunikationscontroller, *communication controller*) und einem Ein-/Ausgabesystem (*process I/O subsystem*).

Der Applikationsrechner besteht aus einem Prozessor, eigenem Speicher, einer Zeitquelle und einem Betriebssystem. Auf diesem Teil des Knotens wird die Anwendungssoftware abgelegt. Der Applikationsrechner bildet somit das funktionale Zentrum des Knotens. Er ist über Schnittstellen an die anderen beiden Bestandteile angebunden. Das Kommunikationssystem wird über die Kommunikationsschnittstelle **CNI** (*communication network interface*) erreicht. Dieses Teilsystem ist für die Kommunikation des Knotens mit anderen Knoten im verteilten System zuständig. Diese Kommunikation kann z.B. zeit- oder ereignisgesteuert (siehe Abschnitt 2.2) ablaufen. Das Ein-/Ausgabesystem verbindet über die **COI** (*controlled object interface*) den Applikationsrechner mit der direkt angeschlossenen Peripherie (z.B. Sensoren oder Aktoren) und ermöglicht so die Anbindung an ein technisches System.

Echtzeitvariablen

Eine Echtzeitvariable (**RTE**, *real-time entity*) ist eine Statusvariable in einem verteilten Echtzeitsystem. Sie markiert einen Zustand in einem Knoten oder in einem Gegenstand der Peripherie (z.B. Sensor). Zu jeder Echtzeitvariablen gibt es einen Bereich, in dem es erlaubt ist den Wert zu setzen. In der Literatur wird dies als *sphere of control* (**SOC**) bezeichnet. So kann z.B. der Temperaturwert eines Temperatursensors nur vom Sensor selbst gesetzt werden. Wird der Wert einer Echtzeitvariablen außerhalb der SOC benötigt, wird zu einem Zeitpunkt eine Abbildung (*real-time image*, **RTI**) dieser Variable erzeugt. Sie kann dann an ein anderes Teilsystem des verteilten Systems übermittelt werden. Die Abbildung gibt den Wert zu einem definierten Zeitpunkt an und hat daher eine begrenzte zeitliche Gültigkeit. Diese Gültigkeit richtet sich

z.B. nach der Änderungsgeschwindigkeit des Wertes. Ein RTI kann in einem Gegenstand der Peripherie (z.B. Aktor) oder in einem Knoten gespeichert werden. Neben den RTE und RTI gibt es Echtzeitobjekte (*real-time objects*, **RTO**). Diese verstehen sich als ein Container in einem Knoten für eine Echtzeitvariable oder deren Abbildung. Diese Container beinhalten zusätzlich noch Funktionalitäten für die Variable oder dessen Abbildung. In verteilten Systemen kann ein RTO in verschiedenen Knoten vorliegen, dabei hat jeder Knoten seine eigene Version des RTO mit einer eigenen Funktionalität. Das verteilte System muss sicherstellen, dass alle Echtzeitobjekte des Systems konsistent sind. Für detaillierte Informationen bezüglich Echtzeitvariablen, wie z.B. zeitliche Gültigkeit, sei an dieser Stelle auf [Kop97] verwiesen.

Taskmanagement

Innerhalb des Knotens (im Applikationsrechner) laufen in der Regel mehrere Aufgaben (*task*) parallel ab. Damit jeder Task rechtzeitig fertig gestellt werden kann und wartende Tasks nur eine maximale Zeitspanne auf die Ausführung warten, gibt es verschiedene Planungsstrategien. Zwei Kategorien von Planungen werden in [Kop97] unterschieden: die zeit- und ereignisgesteuerte Planung. Das ereignisgesteuerte Taskmanagement aktiviert die Tasks über Unterbrechungen (*Interrupt*), wenn ein Ereignis eintritt. Im zeitgesteuerten Taskmanagement wird ein statischer Ablaufplan erzeugt, der angibt welcher Task wann und wie lange aktiv ist. Beide Planungsarten haben Eigenschaften, welche den Eigenschaften der beiden in Abschnitt 2.2 beschriebenen Kommunikationsmodelle entsprechen. Tatsächlich werden das Taskmanagement und das Kommunikationsmodell deshalb meist komplett ereignis- oder zeitgesteuert realisiert. Diese Kombination eignet sich aufgrund der ähnlichen Kontroll- und Zeiteigenschaften der Taskplanung und der Kommunikation. Im weiteren Verlauf dieser Arbeit, wird davon ausgegangen, dass ein System komplett ereignis- oder zeitgesteuert ist. Wenn es nicht anders erwähnt wird, verwendet ein System mit zeitgesteuerter Kommunikation ein zeitgesteuertes Taskmanagement und ein System mit ereignisgesteuerter Kommunikation eine ereignisgesteuertes Taskmanagement.

Cluster

Im Bereich der verteilten eingebetteten Systeme können mehrere Knoten, welche eine gemeinsame Aufgabe lösen, als **Cluster** bezeichnet werden. Ein Cluster beschreibt dabei einen funktionalen Teil eines verteilten Gesamtsystems. Ein Cluster kann intern verschiedene Mechanismen zur Fehlertoleranz (z.B. gegen Knotenausfall) bereitstellen.

2.2 Kommunikationsmodelle

Es gibt verschiedene Verfahren auf ein Kommunikationsmedium zuzugreifen: z.B. Master-Slave, asynchroner (ereignisgesteuerter) Zugriff, Token-Passing und synchroner (zeitgesteuerter) Zugriff [ZS06]. Das **Master-Slave** Verfahren ist aufgrund der Abhängigkeit vom Master gegen Ausfälle der Master-Komponente sehr empfindlich (*single point of failure*). Die Kommunikationsmöglichkeiten dieses Verfahrens sind begrenzt, da Slaves im Allgemeinen nur mit dem Master und nicht untereinander kommunizieren können. Dieses Zugriffsverfahren wird in der Regel bei Low-Cost Bussystemen wie LIN eingesetzt. Das **Token-Passing** Verfahren verwendet eine umlaufende Sendeberechtigung (*Token*) zum Zugriff auf das Medium. Nur der Knoten,

welcher das Token besitzt, kann auf das Medium zugreifen. Dieses Verfahren wird aufgrund der notwendigen Überwachung der korrekten Weitergabe des Tokens nur noch selten eingesetzt.

Das asynchrone und das synchrone Verfahren sind grundlegend verschieden. Der **asynchrone** Zugriff kann jederzeit auf das Medium zugreifen, es muss aber vom Zugriffsprotokoll zugesichert werden, dass Kollisionen erkannt (*CSMA/CD*) oder vermieden (*CSMA/CA*) werden. Diese Methode des Medienzugriffes ermöglicht eine ereignisgesteuerte Anwendung, welche beim Auftreten von Ereignissen asynchron kommuniziert. Es wird bei diesem Medienzugriff von der **ereignisgesteuerten Kommunikation** gesprochen.

Im **synchronen** Verfahren kann der Zugriff auf das Medium nur innerhalb von fest vorgegebenen Zeitfenstern erfolgen (*TDMA*). Diese Art des Medienzugriffes begünstigt eine zeitgesteuerte Anwendung, da der Buszugriff und die Nachrichtengenerierung synchron stattfinden können. Kommunikation, welche den synchronen Medienzugriff verwendet, wird **zeitgesteuerte Kommunikation** bezeichnet.

In den folgenden Unterabschnitten wird die ereignisgesteuerte und die zeitgesteuerte Kommunikation dargestellt und miteinander verglichen.

2.2.1 Ereignisgesteuerte Kommunikation

Bei der ereignisgesteuerten Kommunikation hat die Anwendung die Kontrolle, wann eine Nachricht gesendet werden soll. Tritt ein Ereignis ein, übergibt die Anwendung eine entsprechende Nachricht an das Kommunikationssystem. Die **Verzögerungszeit** der Nachricht ist nicht im Voraus bestimmbar, weil der Empfangszeitpunkt von der Belastung des Kommunikationsmediums und, sofern das Kommunikationsprotokoll dies unterstützt, der Priorität der Nachricht abhängig ist. Ist zu dem Zeitpunkt, an dem die Nachricht gesendet werden soll, das Kommunikationsmedium unbenutzt, kann die Nachricht sofort übermittelt werden. Die Verzögerung zwischen dem Bereitwerden der Nachricht und dem Empfang richtet sich dann nach den Eigenschaften des Kommunikationsprotokolls (z.B. Übertragungsgeschwindigkeit, Verarbeitungsaufwand, Anteil der Nutzlast in der Nachricht). Ist das Kommunikationsmedium beim Bereitwerden der Nachricht belegt oder wird zum selben Zeitpunkt von einem anderen Knoten versucht eine (wichtigere) Nachricht zu versenden, verzögert sich der Sendezeitpunkt. Diese Latenzzeit ist von Protokoll zu Protokoll unterschiedlich. Die Schwankungen der Latenzzeiten durch Belegung des Kommunikationsmediums oder anderer Verzögerungen werden als **Jitter** bezeichnet.

Der Zugriff auf das Medium ist protokollabhängig. **Ethernet** (*IEEE 802.3*) greift z.B. auf das Medium mit dem CSMA/CD Verfahren zu. Eine Nachricht wird nur dann gesendet, wenn das Medium zum Sendezeitpunkt frei ist. Ist das Medium durch eine andere Nachricht belegt, wird solange gewartet bis das Medium wieder frei gegeben wird. Stellt der Kommunikationscontroller bei der Übertragung fest, dass gleichzeitig noch ein anderer Knoten eine Nachricht sendet, wird die Übertragung abgebrochen und versucht die Nachricht später erneut zu senden. Die Neuübertragung der Nachricht wird dann durch eine zufällige Zeit verzögert durchgeführt. Damit wird die Wahrscheinlichkeit für eine erneute Kollision verringert. Die Kollisionsvermeidung bei der wiederholten Übertragung verwendet den so genannte *Binary Exponential Backoff*-Algorithmus [Tan03]. Das Medienzugriffsverfahren von Ethernet unterstützt keine Prioritäten. Dadurch kann nicht garantiert werden, dass wichtige Nachrichten weniger wichtigen

vorgezogen werden. Es ist daher weitestgehend unbestimmbar, wie groß die Latenzzeit für die Nachrichtenübermittlung ist.

Die prioritätsbasierte Arbitrierung beim **CAN-Protokoll** sichert zu, dass Nachrichten mit höherer Priorität vorgezogen werden. Trotzdem lässt sich keine obere Schranke für die Verzögerung der Nachrichtenübermittlung benennen, da der Bus von Nachrichten höherer Priorität blockiert werden kann. Die Ausnahme sind Nachrichten der höchsten Priorität. Sie müssen maximal die Sendezeit einer Nachricht auf dem Bus warten, da sendende Knoten nicht unterbrochen werden dürfen. Für Nachrichten geringerer Priorität können lediglich statistische Betrachtungen des Kommunikationsverhaltens im schlimmsten Fall der jeweiligen Anwendung obere Schranken für die Latenzzeit liefern.

Mit Hilfe der ereignisgesteuerten Kommunikation lassen sich sowohl Ereignisnachrichten als auch Statusnachrichten übermitteln. **Ereignisnachrichten** sind Nachrichten, welche über das Auftreten eines Ereignisses informieren. Dabei enthält die Nachricht nur Statusänderungen (z.B. der Wert hat sich um 2 Punkte erhöht). Damit ein Empfänger den Statuswert rekonstruieren kann, muss er jede Ereignisnachricht genau einmal empfangen (*exactly-once*). Um keine Nachricht zu verpassen, muss der Empfänger eine Warteschlange für eingehende Ereignisnachrichten bereitstellen. So kann jede Nachricht nacheinander verarbeitet und der Statuswert der Echtzeitvariablen zum aktuellen Zeitpunkt ermittelt werden. Der Verlauf des Statuswertes ist durch Speicherung der Ereignisnachrichten rekonstruierbar. **Statusnachrichten** hingegen enthalten den kompletten Statuswert der Variable (z.B. der Wert ist 78 Punkte) und müssen mindestens einmal empfangen werden (*at-least-once*). Da im Allgemeinen nur der aktuelle Statuswert interessant ist, genügt es dem Empfänger beim Eintreffen einer Statusnachricht den alten Wert zu überschreiben. Um den Verlauf des Statuswertes aufzuzeichnen, müssen die alten ungültigen Statusnachrichten erhalten bleiben.

Systeme mit ereignisgesteuerter Kommunikation sind für sporadische und aperiodische Nachrichten optimal [CES03]. **Sporadische Nachrichten** sind wiederkehrend auftretende Nachrichten mit einem definierten minimalen Zeitabstand zwischen zwei Nachrichten. Sie treten daher mit einer bestimmten maximalen Häufigkeit auf. Bei **aperiodischen Nachrichten** ist dieser Abstand nicht bestimmbar. Es kann daher keine Aussage bezüglich der Häufigkeit der Nachrichten getroffen werden. Die ereignisgesteuerte Kommunikation ist für solche Nachrichten sinnvoll, da der Zugriff auf das Kommunikationsmedium nicht von einer Periodizität abhängt, sondern mit dem Auftreten der Ereignisse erfolgt. Natürlich können ereignisgesteuerte Systeme auch periodische Nachrichten verarbeiten. Es kann jedoch aufgrund der Auslastung des Mediums nicht garantiert werden, dass die Nachrichten mit konstanter Verzögerung übermittelt werden. Damit ist es möglich, dass eine Nachricht solange verzögert wird, dass sie erst nach dem Bereitwerden der Nachricht des nächsten Zyklus gesendet wird. Das Verwerfen einer veralteten Nachricht oder das Sicherstellen der zeitgerechten Übermittlung kann nicht durch das ereignisgesteuerte Kommunikationssystem selbst erfolgen. Dazu sind Mechanismen in der verteilten Anwendung oder eine entsprechende Middleware (siehe Abschnitt 2.5) notwendig.

Der Vorteil der ereignisgesteuerten Kommunikation ist die Flexibilität. Ein Knoten kann jederzeit auf das Medium zugreifen und versuchen eine Nachricht zu senden. Dadurch ist die Erweiterung bestehender verteilter Systeme ohne Anpassungen an den davon nicht betroffenen Komponenten möglich. Es können jederzeit neue Knoten oder Nachrichten integriert werden. Bei der Planung des Kommunikationsverhaltens eines Systems sowie bei der Erweiterung muss darauf geachtet werden, dass das Kommunikationsmedium nicht überlastet wird. Die Betrach-

tung der Mediumauslastung eines ereignisgesteuerten Systems ist nicht trivial, da die Kontrolle der Kommunikation der Anwendung obliegt. So kann ein verteiltes System im Normalzustand ein absehbar geringes Nachrichtenaufkommen produzieren. Befindet sich die Anwendung hingegen in einer kritischen Phase, z.B. Start bzw. Landung eines Flugzeuges oder Unfallsituation im Fahrzeug, kann die Nachrichtenzahl stark ansteigen. Gerade in diesen Phasen müssen die entscheidenden Nachrichten z.B. Messwerte, Alarme und Steuerbefehle über das Medium gesendet und vor Ablauf der Fristen korrekt empfangen werden. Deshalb muss die Kommunikation in einem ereignisgesteuerten System für den kritischen Fall[1] geplant werden. Ist dies nicht möglich, muss die Kommunikation so eingeschränkt werden, dass sie planbar wird. In der Literatur gibt es verschiedene Verfahren die ereignisgesteuerte Kommunikation zu planen, z.B. in [TB94, TBW95] werden die Nachrichten mit festen Prioritäten geplant, [LK98, LKJ98, RSH01] beschreiben eine *Earliest Deadline First*-Planung im ereignisgesteuerten CAN-Protokoll und in [ZS97] wird ein Ansatz dargestellt, der *Deadline Monotonic*- und *Earliest Deadline First*-Planung in einem CAN-Netz für unterschiedliche Nachrichtenklassen nutzt.

2.2.2 Zeitgesteuerte Kommunikation

Im Gegensatz zur ereignisgesteuerten Kommunikation liegt die Kontrolle, wann eine Nachricht gesendet wird, bei der zeitgesteuerten Kommunikation nicht in der Anwendung sondern im Kommunikationssystem selbst. Das Kommunikationssystem greift anhand eines Plans (*Schedule*) auf das Medium zu. Die Anwendung bestimmt nur was gesendet wird und hat keinen Einfluss auf den Sendezeitpunkt. Das Kommunikationssystem kennt als einziges Kontrollsignal die periodischen Unterbrechungen durch den Zeitgeber. Dadurch wird die Zeit in gleichgroße Zeitschlitze eingeteilt. Diese Begrenzung des Kontrollflusses garantiert, dass sich ein zeitliches Fehlverhalten der Anwendung nicht auf das Kommunikationssystem und dadurch auf das Verhalten anderer Knoten ausbreitet. In [KB03] wird die zeitliche Trennung von Anwendung und Kommunikationssystem als *Temporal Firewall* bezeichnet.

Alle Knoten des verteilten Systems kennen den für sie wichtigen Ausschnitt des globalen Kommunikationsplans oder den gesamten Plan. Der globale Plan ordnet jedem Knoten die benötigten Sendezeitschlitze zu und sichert einen kollisionsfreien Zugriff aller Knoten auf das Medium. Die Planung erfolgt a priori und der Plan ist für eine Periode definiert. Der Kommunikationsplan wird zur Laufzeit des Systems zyklisch wiederholt. Dieses Medienzugriffsverfahren wird als TDMA bezeichnet. Innerhalb der Periode sind z.B. im TTP/C-Protokoll [TTA03] mehrere Runden definierbar, wodurch für unterschiedliche Nachrichten unterschiedliche Periodenzeiten ermöglicht werden.

Damit der Zugriff jedes Knotens auf das Medium auch genau in dem Zeitschlitz liegt, müssen die Uhren der Kommunikationssysteme hinreichend synchron sein. Neben der Synchronisation der Kommunikationssysteme sollten auch die Betriebssysteme und somit das Taskmanagement der Knoten auf diese globale Zeit synchronisiert werden. Dies verringert zwar nicht die Latenz der Nachrichtenübertragung, lässt sie aber einen konstanten Wert annehmen [Alb04]. Damit ist der Kommunikationsjitter nahezu null und die Kommunikation wird direkt planbar. Die Nachrichten erreichen den Empfänger genau zu dem a priori geplanten Zeitpunkt. In [Alb04] wird dies anhand eines verteilten Regelungssystems dargestellt.

[1] Der kritische Fall beschreibt das höchst mögliche Nachrichtenaufkommen in der jeweiligen Anwendung.

Aufgrund der Periodizität des Plans ist die zeitgesteuerte Kommunikation besonders für periodische Nachrichten geeignet. Mit dem gesicherten Zugriff des Kommunikationssystems auf des Medium zu periodischen Zeitpunkten erreicht die Nachricht zur definierten Zeit den Empfänger. Durch diese Vorhersagbarkeit, kann die Verzögerung periodischer Nachrichten als gleich null angesehen werden [CES03]. Das Modell der zeitgesteuerten Kommunikation ist speziell für periodische **Statusnachrichten** ausgelegt. Statusnachrichten haben die Eigenschaft, dass sie, auch wenn sich der Statuswert nicht ändert, wiederholt gesendet werden. Eine fehlerhafte oder fehlende Statusnachricht wird dabei automatisch in der nächsten Periode erneut gesendet. Dadurch ist eine explizite wiederholte Übermittlung im Fehlerfall bei periodischen Statusnachrichten unnötig.

Sporadische oder aperiodische Nachrichten werden vom zeitgesteuerten Kommunikationsmodell nicht unterstützt. Sie können aber als periodisch interpretiert oder über eine Erweiterung des Kommunikationsprotokolls im zeitgesteuerten Modell übermittelt werden. Nicht periodische Nachrichten, die als periodische Nachrichten interpretiert werden, erhalten einen eigenen Zeitschlitz. Dabei gibt die Gültigkeit der Nachricht die Periode der Nachricht an. Aufgrund der nicht periodischen Eigenschaften der Nachrichten wird der zugeordnete Zeitschlitz nicht immer ausgelastet. Eine andere Möglichkeit sporadische Nachrichten in zeitgesteuerten Systemen zu Versenden beschreibt [CES03], dort werden sporadische Nachrichten immer dann versendet, wenn der Sender einen freien Platz in seinem Kommunikationsplan hat. Es muss dazu ein Zeitschlitz für nicht periodischen Nachrichten in jedem Knoten reserviert werden. In diese Zeitschlitze reihen sich die Nachrichten z.B. nach ihrer Priorität ein und werden gesendet. Ist der Zeitschlitz zu voll, werden die restlichen Nachrichten in den nächsten Perioden übermittelt. Dieses Verfahren wird vom zeitgesteuerten Kommunikationsmodell nicht vorgesehen und bildet damit eine Erweiterung des zeitgesteuerten Kommunikationsprotokolls.

Für hochpriore nicht periodische Nachrichten mit sehr kurzer Deadline wie Notaus- oder Alarmmeldungen, reicht es nicht aus die Nachrichten zu senden, wenn im Kommunikationsplan einen freier Platz ist. Es muss ein eigener Zeitschlitz mit einer von der Deadline abhängenden Periode reserviert werden, um zu garantieren, dass jede Nachricht rechtzeitig ankommt. Dadurch nimmt eine Nachricht, welche vielleicht nie oder nur sehr selten im System auftreten wird, einen festen Teil der Bandbreite des Mediums ein. Anhand dieses Beispieles lässt sich einer der wichtigsten Vorteile des zeitgesteuerten Kommunikationsmodells erkennen. Durch die Zusicherung der Bandbreite für eine Nachricht wird sichergestellt, dass sie auch zum festgelegten Zeitpunkt übermittelt wird. Selbst für nicht periodische Nachrichten wird damit eine maximale Latenz garantiert. Damit ist das Verhalten des zeitgesteuerten Ansatzes vorhersehbar. Die Vorhersehbarkeit der Kommunikation spielt bei verteilten Echtzeitsystemen eine entscheidende Rolle.

Der Nachteil des Kommunikationsmodells, der im obigen Beispiel erkennbar wird, ist Blockierung von unbenutzter Bandbreite bei der Abbildung von nicht periodischen Nachrichten auf periodische Zeitschlitze. Die Verwendung der periodischen Zeitschlitze durch das zeitgesteuerte Kommunikationsmodell erlaubt aber die einfache Planung des Kommunikationsverhaltens. Denn gerade das nicht absehbare Nachrichtenaufkommen der ereignisbasierten Kommunikation macht die Planung bzw. die Planbarkeitsanalyse komplex. Durch das deterministische Kommunikationsverhalten lässt sich eine weit höhere Auslastung der Bandbreite des Kommunikationsmediums erreichen [CES03].

2.2.3 Vergleich der Kommunikationsmodelle

Im diesem Unterabschnitt werden die beiden zuvor vorgestellten Kommunikationsmodelle verglichen. Die Tabelle 2.1 am Ende dieses Unterabschnittes gibt eine Übersicht über die Eigenschaften der beiden Modelle. Die Kriterien für den Vergleich sind in Anlehnung an [Obe05] gewählt.

Vorhersagbarkeit

In ereignisgesteuerten Kommunikationssystemen ist die Kommunikation nicht vorhersehbar und daher nur schwer planbar, d.h. es kann nicht bestimmt werden, zu welchem Zeitpunkt welche Nachricht gesendet wird. Dies liegt darin begründet, dass jeder Knoten im verteilten System zu jedem Zeitpunkt die Möglichkeit hat Nachrichten zu senden. Versuchen mehrere Knoten (oder auch Tasks auf einem Knoten) gleichzeitig eine Nachricht zu senden, kommt es zu Verzögerungen in der Übermittlung. Das Auftreten dieses Falles ist deshalb nicht vorhersehbar, weil die Kontrolle darüber, wann eine Nachricht gesendet wird, nicht dem Kommunikationssystem sondern der Anwendung (oder gar der Peripherie) unterstellt ist. Daher kann nicht bestimmt werden, wann oder ob es zu Hochlastsituationen (*peak load scenario*) kommt und welches Nachrichtenaufkommen sie erzeugen. Es sind nur Aussagen zum schlechtesten Fall des Nachrichtenaufkommens möglich, bei dem alle Nachrichten zum gleichen Zeitpunkt gesendet werden. Damit kann die obere Schranke für die Latenz der Nachrichten bestimmt werden. Diese maximale Verzögerung gibt aber keine Aussage darüber, wie sich die Verzögerungszeiten in allen anderen Situationen zusammensetzen. Es kommt somit zu Schwankungen der Kommunikationsverzögerung. Die Größe dieses Jitters beschreibt die Vorhersagbarkeit. Eine Kommunikation mit kleinem Jitter ist besser vorhersagbar als eine Kommunikation mit großem Jitter. Aufgrund der Latenzschwankungen durch die Überlagerung der Kommunikation von verschiedenen Knoten ist die Vorhersagbarkeit der ereignisgesteuerten Kommunikation stark eingeschränkt.

Werden in einem ereignisgesteuerten Kommunikationssystem Ereignisnachrichten genutzt, muss sichergestellt werden, dass der Empfänger die Nachricht genau einmal erhält. Dazu können Quittierungen verwendet werden. Wird eine Nachricht innerhalb einer gewissen Zeitspanne nicht quittiert, geht der Sender davon aus, dass die Nachricht nicht angekommen ist und sendet sie erneut. Alternativ kann ein Empfänger einer fehlerhaften Nachricht dem Sender den Fehler signalisieren. Durch die wiederholte Übermittlung von nicht empfangenen oder fehlerhaften Nachrichten entsteht eine weitere variable Schwankung der Kommunikationsverzögerung. Damit wird die Vorhersagbarkeit der Kommunikation weiter beeinträchtigt, da nicht vorausgesehen werden kann, ob der Empfänger eine Nachricht korrekt, mit Fehlern oder gar nicht erhält.

In der zeitgesteuerten Kommunikation steht der Plan, welcher Knoten welche Nachricht zu welchem Zeitpunk sendet, zur Entwurfszeit des Systems fest. Daher ist die Kommunikation vorhersehbar. Die Kommunikationsverzögerung ist von der Sendekontrolle im Kommunikationssystem und nicht vom Auftreten eines Ereignisses in der Anwendung abhängig. Die Latenz ist allein durch das Kommunikationsmedium und den Kommunikationscontroller gegeben. Schwankungen der Latenzzeiten in der zeitgesteuerten Kommunikation können nur durch Ungenauigkeiten der Zeitsynchronisation entstehen, aus diesem Grund muss in jedem Knoten eine möglichst genaue globale Zeit vorhanden sein. Das Problem des wiederholten Sendens falsch oder nicht angekommener Nachrichten gibt es in der zeitgesteuerten Kommunikation nicht,

da kein Fehlersignalisierungsmechanismus vorgesehen und ein fehlerabhängiges wiederholtes Senden im Kommunikationsplan auf Protokollebene nicht abbildbar ist. Wegen den fehlenden Nachrichtenneuübermittlung im Fehlerfall werden in zeitgesteuerten Protokollen im Allgemeinen nur Statusnachrichten versendet. Empfängt ein Knoten eine Statusnachricht nicht, so kann er eine Periode warten und erhält dann den Statuswert erneut. In sicherheitskritischen Systemen ist diese Verzögerung nicht akzeptabel. Sicherheitskritische Systeme übermitteln eine Nachricht daher immer redundant (entweder mehrmals und/oder auf verschiedenen Leitungen). Das mehrmalige Senden ist dabei unabhängig von Fehlern in Nachrichten. Durch die Verhinderung von erneutem Übertragen falscher oder nicht empfangener Nachrichten und der Nutzung festen Sendeplans ist der Jitter minimal und die Kommunikation in zeitgesteuerten Kommunikationssystemen vorhersagbar.

Zusammensetzbarkeit

Die Zusammensetzbarkeit von Teilsystemen zu einem Gesamtsystem ist für ein verteiltes System von großer Wichtigkeit. Es wird daher diese Eigenschaft für beide Kommunikationsmodelle untersucht. Wie bereits erläutert liegt bei ereignisgesteuerten Kommunikationssystemen die Kontrolle der Nachrichtenübermittlung außerhalb des Kommunikationssystems. Ereignisse im Applikationsrechner oder der angeschlossenen Peripherie (z.B. Sensoren) bestimmen das Kommunikationsverhalten. Nach [Obe05] wird dadurch die Trennung von Gesamtstruktur und Knotendesign behindert. Daraus folgt, dass aufgrund der externen Kommunikationskontrolle, separat entwickelte Einzelsysteme nicht ohne weiteres zu einem Gesamtsystem zusammengesetzt werden können. Die Kombination eines Gesamtsystems aus den zeitlich korrekten Teilsystemen, kann dazu führen, dass die Teilsysteme sich gegenseitig blockieren und so ihre zeitlichen Eigenschaften nicht einhalten können. Für ein solches System müssen die zeitlichen Bedingungen erneut angepasst und überprüft werden. Handelt es sich um ein prioritätsbasiertes ereignisgesteuertes Kommunikationsprotokoll müssen für das zusammengesetzte Gesamtsystem eventuell die Prioritäten der Nachrichten angepasst werden. Diese Aspekte erschweren die Wiederverwendbarkeit von bestehenden Teillösungen.

Da das Kommunikationssystem bei zeitgesteuerter Kommunikation die alleinige Kontrolle über die Kommunikation besitzt und somit die Anwendung nicht direkt auf die Kommunikation Einfluss nehmen kann, wird das zeitliche Verhalten des Kommunikationssystems vom Verhalten des Applikationsrechners getrennt [Obe05]. Die Integration von zeitgesteuerten Teilsystemen zu einem Gesamtsystem ist damit einfacher als für ereignisgesteuerte Teilsysteme. Für das Zusammensetzen von Teilsystemen wird ein gemeinsamer Kommunikationsplan erstellt. Lässt sich kein globaler Plan erzeugen, können die Teilsysteme nicht zusammengesetzt werden. Dies tritt dann auf, wenn das Kommunikationsmedium im zusammengesetzten System überlastet wird. Die Möglichkeit der Kombination von Teilsystemen vereinfacht die verteilte Entwicklung, z.B. im Automobil- oder Fugzeugbereich [PK99, Pla02], wo unterschiedliche Komponenten von verschiedenen Zulieferern entwickelt werden. Hierbei kann zuerst der globale Kommunikationsplan in der Designphase erzeugt werden. Die Umsetzung der Teilanwendungen wird dann unabhängig voneinander realisiert. Auch bereits bestehende Teillösungen können mit zeitgesteuerten Kommunikationssystemen, aufgrund der Zusammensetzbarkeit der Kommunikation, einfacher wiederverwendet werden.

Determinismus von Replikaten

In [Obe05] wird auf die Schwierigkeit des identischen Verhaltens von replizierten Komponenten in ereignisgesteuert kommunizierenden Systemen eingegangen. Der Determinismus von Replikaten (*replica determinism*) bedeutet, dass korrekt replizierte Systeme sich komplett identisch verhalten. Sie geben die gleiche Ausgabe in derselben Reihenfolge und dies innerhalb des gleichen Zeitintervalls. Dieses identische Verhalten ist dann notwendig, wenn zur Fehlererkennung und Fehlertoleranz ein einzelnes System repliziert wird. Kann der Determinismus der Replikate durch das Systemmodel nicht erreicht werden, wird vom Nichtdeterminismus der Replikate (*replica nondeterminism*) gesprochen. Das Problem bei ereignisgesteuerter Kommunikation ist, dass es Quittierungen, Timeouts und wiederholte Übermittlungen von Nachrichten gibt. Hierbei kann sich das Verhalten der Replikate schnell unterscheiden, wenn z.B. ein Replikat die Quittung noch innerhalb des Timeouts bekommt und ein anderes Replikat nicht. Soll ein verteiltes System mit Replikaten eine ereignisgesteuerte Kommunikation unterstützen, müssen sich die Replikate untereinander abstimmen, um sich identisch zu verhalten.

Wie zuvor beschrieben ist das zeitgesteuerte Kommunikationsmodell, aufgrund des a priori geplanten Kommunikationsverhaltens, vorhersagbar. Es ermöglicht daher eine einfachere Verwendung von replizierten Komponenten. Dennoch kann es dazu kommen, dass ein Knoten eine Nachricht noch als gültig ein anderer sie aber als außerhalb des Sendefensters erkennt (*slightly-off-specification fault*). Dieses Problem muss beim Einsatz von Replikaten durch entsprechende Mechanismen gelöst werden [Kop97].

Flexibilität

Der Vorteil der ereignisgesteuerten Kommunikation ist deren Flexibilität, denn das Kommunikationsmedium wird nur in Anspruch genommen, wenn tatsächlich eine Nachricht zu senden ist. Das verteilte System entscheidet zur Laufzeit darüber, welche Nachricht wann gesendet wird. Damit unterliegt die zeitliche Dynamik des Systems nur wenigen festen Einschränkungen (z.B. Bandbreite). Dies ist von Vorteil, wenn im verteilten System zur Laufzeit einzelne Komponenten hinzugefügt oder entfernt werden. Dadurch kann, indem einige Nachrichten öfter oder seltener gesendet werden bzw. Nachrichten hinzukommen oder entfernt werden, das Nachrichtenaufkommen zur Laufzeit verändert werden. Ein ereignisgesteuertes System vereinfacht somit die Veränderung (z.B. Umverteilung der Funktionalitäten auf die Knoten) und Erweiterung eines verteilten System zur Laufzeit. Nach einer Änderung muss das zeitliche Verhalten des Systems erneut getestet werden, um sicherzustellen, dass die zeitlichen Bedingungen auch im geänderten System[2] erfüllt werden. Die Planung der Kommunikation des geänderten Systems kann auf den Planungsergebnissen des Nachrichtenaufkommens im Ausgangssystem aufbauen. So können unveränderten Knoten von der Neuplanung ausgeschlossen werden. Es ist keine komplette Neuplanung des Kommunikationsaufkommens für alle Knoten erforderlich. Diese Flexibilität ermöglicht weiterhin die optimale Unterstützung von nicht periodisch auftretenden Nachrichten (z.B. Alarmmeldungen), weil diese Nachrichten nur zu dem Zeitpunkt vom Kommunikationssystem berücksichtigt werden müssen, an dem sie auftreten. Dies ist nach [Alb04] der entscheidende Vorteil ereignisgesteuerter Modelle, denn dadurch können minimale

[2] Bei einer reinen Umverteilung der Aufgaben auf die Knoten verändert sich das Kommunikationsaufkommen nicht, es bedarf daher auch keines Tests der zeitlichen Eigenschaften.

Verzögerungszeiten beim Auftreten von ungeplanten nicht periodischen Nachrichten erreicht werden.

Ein zeitgesteuertes Modell besitzt keinerlei Flexibilität bei der Kommunikation. Alle Nachrichten sind im Kommunikationsplan periodisch eingeplant und werden genau so vom Kommunikationssystem übermittelt. Die Verwendung von periodischen Zeitschlitzen für aperiodische und sporadische Nachrichten ist nicht optimal, weil die Zeitschlitze aufgrund der fehlenden Periodizität der Nachrichten nicht immer belegt sind. Die Ausnutzung der Zeitschlitze ist abhängig von der zeitlichen Gültigkeit der Informationen und der daraus gewählten Periodendauer der zugeordneten Zeitschlitze. Eine andere Möglichkeit nicht periodische Nachrichten zu senden, ist die Verwendung eines Zeitschlitzes pro Knoten für alle im Knoten anfallenden Nachrichten. Dieser Ansatz von [CES03] weist im Vergleich zur ereignisgesteuerten Kommunikation eine erhöhte mittlere Latenz für alle Nachrichten auf, denn es steht beim Mediumzugriff eine lokale Nachrichtenwarteschlange im zeitgesteuerten Modell der globalen Warteschlange im ereignisgesteuerten Modell gegenüber. Diese Eigenschaft der zeitgesteuerten Kommunikation bedingt eine höhere aber konstantere Latenz der nicht periodischen Nachrichten. Daher können sporadische Nachrichten mit sehr kurzer Gültigkeit nicht so gut gehandhabt werden wie bei der ereignisgesteuerten Kommunikation.

Die Veränderung des Kommunikationsaufkommens zur Laufzeit des Systems mit zeitgesteuerter Kommunikation ist nur dann möglich, wenn diese Änderungen im Kommunikationsplan bereits zuvor berücksichtigt wurden und damit ein entsprechender ungenutzter Zeitschlitz zur Verfügung steht. Der zuvor erstellte Kommunikationsplan erfüllt bereits die zeitlichen Bedingungen und muss nicht bei der Erweiterung des Systems erneut überprüft werden. Ist bei der Erweiterung des Systems kein entsprechender Zeitschlitz mehr im Plan verfügbar, muss ein neuer globaler Kommunikationsplan erzeugt und an alle Knoten übergeben werden. Damit kann eine kleine Änderung in einer Teilkomponente die komplette Neuplanung des Kommunikationsverhaltens auslösen. Die eingeschränkte Flexibilität des Kommunikationsmodells erschwert die Erweiterung eines verteilten Systems zur Laufzeit enorm.

Ausfallerkennung

Die Erkennung von Knotenausfällen in ereignisgesteuerten Kommunikationssystemen ist, aufgrund des eingeschränkten Wissens über das zeitliche Kommunikationsverhalten der Knoten, nur schwer zu realisieren. Es kann nicht sicher festgestellt werden, wenn eine längere Zeit keine Nachrichten von einem Knoten empfangen werden, ob der Knoten ausgefallen ist oder ob er keine neuen Informationen zu senden hat. Die Erkennung eines Ausfalls ist in diesem Fall in einem begrenzten Zeitrahmen nicht möglich. Um den Zeitraum bis zum Erkennen eines Knotenausfalls einzugrenzen, können periodisch Lebenszeichen gesendet werden.

Durch das zyklische Senden der Nachrichten bei einer zeitgesteuerten Kommunikation, ist die Erkennung von Knotenausfällen bedeutend einfacher. Durch den globalen Kommunikationsplan weiß jeder Empfänger einer Nachricht, wann diese gesendet werden muss. Diese Nachricht dient für alle empfangenden Knoten als Lebenszeichen des Senders. So kann ein empfangender Knoten, wenn eine Nachricht nicht oder nicht rechtzeitig gesendet wird, davon ausgehen, dass der sendende Knoten ausgefallen bzw. fehlerhaft oder er selbst fehlerhaft ist.

Mediumauslastung

Das Kommunikationsmedium wird bei ereignisgesteuerter Kommunikation nur dann genutzt, wenn auch tatsächlich Nachrichten von der Anwendung gesendet werden müssen. Wird es von einem Knoten nicht genutzt, kann jeder andere Knoten auf das Medium zugreifen. Damit ergibt sich eine effektiv genutzte Bandbreite. Es kann aber durch den Mediumzugriff der Knoten dazu kommen, dass Nachrichten verzögert werden. Bei der ereignisgesteuerten Kommunikation muss die theoretisch höchst mögliche Belastung des Mediums (*peak load scenario*) eingeplant werden, damit alle kritischen Nachrichten in allen Situationen rechtzeitig empfangen werden. Für das normale Nachrichtenaufkommen außerhalb der Hochlastsituationen wird daher nur ein Teil der Bandbreite des Mediums belegt. Dies führt zu einer mittleren bis geringen Mediumauslastung. In [CES03] wird dargestellt, dass eine ereignisgesteuerte Kommunikation unterhalb von Auslastungen bis 0.6 besser geeignet ist als eine zeitgesteuerte Kommunikation. Unterhalb dieser Auslastung erreicht die ereignisgesteuerte Kommunikation im Gegensatz zur zeitgesteuerten minimale mittlere Verzögerungen. Dies gilt jedoch nur für sporadische Nachrichten. Bei periodischen Nachrichten ist die zeitgesteuerte Kommunikation gegenüber der ereignisgesteuerten aufgrund fehlender Latenzschwankungen bei gleicher Auslastung überlegen. Die ereignisgesteuerte Kommunikation eignet sich daher besonders bei sporadischen Nachrichten und einer mittleren Auslastung der verfügbaren Bandbreite.

Die Kommunikation in zeitgesteuerten Systemen folgt einem festen Plan. Dieser ist so dimensioniert, dass er der höchsten Lastsituation (*peak load scenario*) der Anwendung genügt. Die Kommunikation von solchen Systemen ist effektiver planbar als bei der ereignisgesteuerten Kommunikation, da das Kommunikationsverhalten voraussagbar ist. Daher kann ein Medium mit der zeitgesteuerten Kommunikation höher ausgelastet werden. Die zeitgesteuerte Kommunikation hat gerade für Anwendungen mit hoher Mediumauslastung (größer als 0.7) gegenüber der ereignisgesteuerten Kommunikation große Vorteile. Bei solch hohen Auslastungen kommt es, aufgrund von Kollisionen oder langen Warteschlangen, in ereignisgesteuerten Systemen zu langen Verzögerungen der Nachrichten. Es kann dann nicht mehr zugesichert werden, dass alle Nachrichten rechtzeitig ankommen. In zeitgesteuerten Kommunikationssystemen ist die Kommunikationsverzögerung auch bei hoher Auslastung des Mediums konstant. Somit eignet sich die zeitgesteuerte Kommunikation besonders für Anwendungen mit hoher Mediumauslastung.

Modulkosten

Die ereignisgesteuerte Kommunikation stellt geringe Ansprüche an die verwendete Hardware. Für die zeitgesteuerte Kommunikation hingegen wird in den einzelnen Knoten ein genauer Zeitgeber benötigt, damit der Kommunikationscontroller zum vom Plan vorgegebenen Zeitpunkt die Nachrichten versenden kann. Dazu müssen die Knoten sich auf eine globale Zeit synchronisieren. Dies kosten einen Teil der Rechenleistung. Weiterhin erfordern zeitgesteuerte Knoten eine spezielle Architektur des Kommunikationscontroller und des Applikationsrechner, bei der es zwischen beiden Komponenten keinen Kontrollfluss gibt. Diese Ansprüche an die Hardware führen im Vergleich zu den ereignisgesteuerten Knoten zu höheren Modulkosten. Für die Verkabelung werden in zeitgesteuerten Systemen meist redundante Kanäle verwendet, damit steigen auch die Kosten für die Verkabelung. Die Mehrkosten der Module und der Verkabelung für die zeitgesteuerte Kommunikation sind für sicherheitskritische Anwendungen tolerierbar.

Eigenschaft	ereignisgesteuert	zeitgesteuert
Vorhersagbarkeit	gering	hoch
Zusammensetzbarkeit	schwierig	einfach
Determinismus von Replikaten	schwierig	einfach
Flexibilität	hoch	gering
Ausfallerkennung	langsam	schnell
Mediumauslastung	für mittlere Auslastung	für hohe Auslastung
Modulkosten	gering	hoch

Tabelle 2.1: Vergleich ereignis- und zeitgesteuerter Kommunikation

2.2.4 Hybride Architekturen

Es gibt Bestrebungen das zeitgesteuerte und das ereignisgesteuerte Kommunikationsmodell in einem hybriden Modell zu vereinen, um so die Vorteile beider Ansätze in einem System nutzen zu können. Einen solchen Ansatz verfolgt z.B. FlexRay, welches neben der zeitgesteuerten Kommunikation Mechanismen für die Unterstützung von sporadischen Nachrichten vorsieht. Das Protokoll wird in Unterabschnitt 2.3.6 genauer betrachtet. Ein weiteres Beispiel beschreibt [PEP02]. Es wird ein Ansatz erläutert mit dem sich ein verteiltes System bestehend aus ereignis- und zeitgesteuerten Teilen ganzheitlich planen lässt. Eine ausführliche Beschreibung der Integration von zeit- und ereignisgesteuerter Kommunikation in einem Gesamtsystem ist Gegenstand von Kapitel 3.

2.3 Kommunikationsprotokolle

In diesem Abschnitt werden verschiedene Kommunikationsprotokolle aus dem Bereich der eingebetteten Systeme vorgestellt. Dabei wird mit dem CAN-Protokoll ein weit verbreitetes ereignisgesteuertes Protokoll dargestellt. Die Unterabschnitte Profibus, TTP/A und LIN zeigen kurz drei Master-Slave Protokolle aus dem automotiven und industriellen Bereich. Mit TTP/C wird in Unterabschnitt 2.3.5 ein zeitgesteuertes Protokoll vorgestellt. Im Anschluss daran werden die beiden hybriden Kommunikationsprotokolle FlexRay und TTCAN beschrieben.

2.3.1 CAN

Das CAN-Protokoll (*Controller Area Network*) wurde in der erweiterten Version 1991 von Bosch [Rob91] definiert und ist heute weit verbreitet. Aufgrund der Eigenschaften dieses Protokolls wird es im Automobilbereich aber auch als schneller Feldbus in der Automatisierung und Gebäudeleittechnik eingesetzt. Das CAN-Protokoll ist ereignisgesteuert und verwendet priorisierte Nachrichten. Die Priorisierung sichert in Hochlastsituationen wichtigen Nachrichten eine geringe Latenz zu und ermöglicht damit die eingeschränkte Funktion des Kommunikationssystems in Überlastphasen. Die Übertragungsrate des CAN-Protokolls ist bis zu 1MBit/s, wobei mit steigender Bitrate die maximale Ausdehnung des Busses abnimmt. Die Busausdehnung ist durch die Signallaufzeit der Bits begrenzt. Eine Bitzeit beträgt die Zeit, die ein Bit benötigt, um sich einmal über den gesamten Bus auszubreiten und wieder vom Sender empfangen zu werden. Dies ist notwendig, da der entfernteste Busteilnehmer das aktuelle Bit lesen und

eventuell durch ein anderen Bitwert austauschen kann. Die Änderung des Bitwertes muss vom Sender noch während das Bit auf dem Bus liegt wahrgenommen werden. Ein Sender legt damit beim Senden einen Bitwert auf den Bus und prüft den Wert des eigenen Bits darauf, ob es z.B. durch einen Fehler oder einen anderen Knoten verändert wurde. Damit erreicht ein 1MBit/s CAN-Bus eine maximale Ausdehnung von 40m und bei 300kBit/s eine Länge von 200m. Im Folgenden wird auf die Busarbitrierung, die Nachrichtenstruktur und die Fehlererkennung im CAN-Protokoll eingegangen.

Das CAN-Protokoll erlaubt jedem am Bus angeschlossenen Knoten einen dezentralen asynchronen Buszugriff mittels kollisionsauflösender Arbitrierung. Das Medienzugriffsverfahren des CAN-Protokolls wird daher mit *Carrier Sense Multiple Access / Consistent Arbitration* bezeichnet. Bevor der Knoten den Bus belegen kann, muss der Bus frei sein. Durch den unkoordinierten Buszugriff der Knoten ist es möglich, dass mehrere Knoten versuchen gleichzeitig den Bus mit einer neuen Nachricht zu belegen. Konflikte, die durch den gleichzeitigen Zugriff auf den Bus entstehen können, werden durch das CAN-Protokoll während der **Arbitrierungsphase** (Auswahlphase) aufgelöst. Dabei gewinnt der Teilnehmer mit der Nachricht mit der höchsten Priorität. Die Priorität wird durch die Bitwerte des Identifiers im Arbitrierungsfeld der Nachricht bestimmt. Im CAN-Protokoll werden die zwei verschiedenen Buspegel unterschieden: rezessiv und dominant. Der rezessive Pegel (*rezessives Bit*) wird bei gleichzeitigem Aufschalten des dominanten Pegels (*dominantes Bit*) durch einen anderen Teilnehmer unterdrückt und der dominante Pegel liegt dann auf dem Bus. Damit kann der CAN-Bus vereinfacht als eine *Wired-And*-Schaltung dargestellt werden [Ets94].

Die Idee der Busarbitrierung ist es nacheinander die einzelnen Bits der Priorität der zu sendenden Nachricht aufzuschalten und zu überprüfen, ob das gesendete Bit mit dem empfangenen übereinstimmt. Man spricht hierbei von der bitweisen Arbitrierung. Das höchste Bit (*MSB*, *most significant bit*) des Identifiers wird bei der Arbitrierung zuerst übertragen. Ist der Bus unbelegt, kann ein Knoten mit der Arbitrierung beginnen. Ein leerer Bus ist durch 11 rezessive Bits in Folge gekennzeichnet[3]. Um Anzuzeigen das ein Knoten mit der Arbitrierung beginnt, wird ein dominantes Bit, das *Start-of-Frame* Bit (*SOF-Bit*), aufgeschaltet, siehe dazu Abbildung 2.1. Damit weiß jeder Knoten das die Busarbitrierung begonnen hat. Knoten die kein SOF-Bit gesendet haben lesen als Empfänger die Nachricht mit. Haben mehrere Knoten das SOF-Bit gesendet, wird während der Arbitrierung entschieden, welcher Knoten seine Nachricht senden darf. Dazu sendet jeder Knoten, der ein SOF-Bit gesendet hat, bitweise sein Arbitrierungsfeld und vergleicht das gesendete Bit mit dem tatsächlich auf dem Bus vorhandenen. Stellt ein Knoten fest, dass statt eines aufgeschalteten rezessiven Bits ein dominantes gelesen wird, scheidet er aus der Arbitrierungsphase aus und wird zum Empfänger der Nachricht. Am Ende der Arbitrierungsphase steht ein Sender der Nachricht fest, dieser beginnt dann mit dem bitweisen Übertragen der Nachricht. Voraussetzung hierbei ist, dass die Identifier eindeutig einer Nachricht, welche nur von einem Knoten gesendet werden kann, zugeordnet ist. Im Allgemeinen ist, durch die Eigenschaften des Mediums, das dominante Bit mit 0 belegt, so dass der Identifier mit dem niedrigsten Wert die Arbitrierung gewinnt und damit die höchste Priorität hat.

Tritt der Fall auf, dass eine Nachricht und eine Anforderung für dieselbe Nachricht gleichzeitig gesendet werden, entscheidet nicht der Identifier der Nachricht über den Erfolg der Arbitrierung, sondern das RTR-Bit (*Remote Transmission Request Bit*). Da eine Nachricht wichtiger

[3] Wenn der Bus längere Zeit unbenutzt ist, sind es entsprechend mehr rezessive Bits

als eine Anforderung für dieselbe Nachricht ist, gewinnt der Sender der Nachricht die Arbitrierung. Der anfordernde Knoten, wird dann zum Empfänger der Nachricht und erhält die angeforderte Nachricht ohne die Anforderung gesendet zu haben. Versuchen mehrere Knoten gleichzeitig eine Anforderung für die gleiche Nachricht zu senden und der Sender der eigentlichen Nachricht ist nicht an der Arbitrierung beteiligt, gewinnen alle die Arbitrierung. Dabei müssen alle Teilnehmer die Anforderung mit identischem Inhalt senden. Andernfalls kommt es zu Fehlern auf dem Bus. Bei beiden zuvor dargestellten Beispielen wird angenommen, dass die betrachteten Nachrichten die höchste Priorität unter den an der Arbitrierung teilnehmenden Nachrichten haben.

Nach der Arbitrierung sendet der Knoten, welcher den Bus erhalten hat, entweder eine Datennachricht (*Data Frame*) und eine Anforderungsnachricht (*Remote Frame*). Datennachrichten übermitteln eine Nutzlast (*Payload*) vom Sender an einen oder mehrere Empfänger. Mittels der Anforderungsnachrichten können Knoten Datennachrichten anfordern. Für beide Nachrichtentypen ist, aufgrund der Erweiterung des CAN-Protokolls, ein Standard- und erweitertes Format definiert. Zu diesen Nachrichtentypen gibt es mit den Fehlernachrichten (*Error Frame*) und Überlastnachrichten (*Overload Frame*) zwei Kontrollnachrichtenformate. Sender und Empfänger können Fehlernachrichten senden, um alle Knoten über einen erkannten Fehler zu informieren. Mit Überlastnachrichten wird eine Verzögerung zwischen zwei aufeinander folgenden Daten- oder Anforderungsnachrichten ermöglicht.

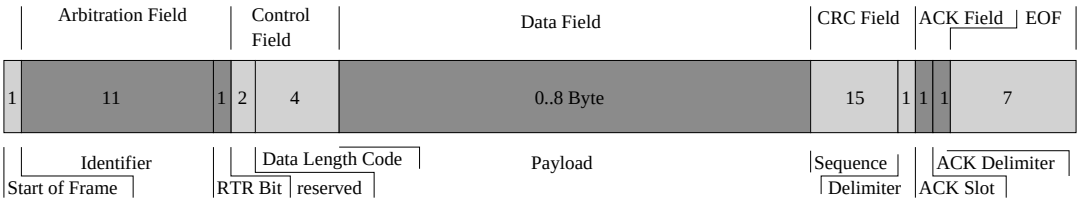


Abbildung 2.1: CAN-Standardrahmen nach [Rob91]

Eine CAN **Datennachricht** im **Standardformat** ist in Abbildung 2.1 dargestellt. Ein solcher Rahmen (*Frame*) beginnt mit dem Start-Of-Frame Bit. Das SOF-Bit darf nur auf den Bus gelegt werden, wenn er unbenutzt ist, dies wird durch eine Sequenz von 11 rezessiven Bits gekennzeichnet. Der Platz zwischen den Rahmen wird als *Interframe Space* bezeichnet. Das zur Arbitrierung notwendige Arbitrierungsfeld wird aus dem Nachrichtenindentifier und dem RTR-Bit (*Remote Transmission Request Bit*) gebildet. Das RTR-Bit unterscheidet Anforderungsnachrichten von Datennachrichten, wobei Datennachrichten bei der Arbitrierung Vorrang haben. Der Nachrichtenidentifier (Identifier) ist im CAN-Standardformat 11 Bits und im erweiterten Format 29 Bits lang.

Das sich an das Arbitrierungsfeld anschließende Kontrollfeld enthält zwei reservierte Bits sowie die Länge der Nutzlast in Byte (*Data Length Code*). Das erste der beiden reservierten Bits wird im erweiterten CAN-Protokoll als IDE-Bit (*Identifier Extension Bit*) bezeichnet und macht die Unterscheidung zwischen Standard- und erweitertem Format möglich. Ist dieses Bit auf den dominanten Pegel gesetzt wird, handelt es sich um das Standardformat. Beim erweiterten Format ist dieses Bit auf den rezessiven Pegel gesetzt und befindet sich nicht im Kontroll- sondern im Arbitrierungsfeld (siehe Abbildung 2.2). Die Nutzlastlänge wird mit 4 Bits dargestellt. Ein Wert dieses Feldes von 0 entspricht 0 Datenbytes und ein Wert von 8 entspricht 8 Datenbytes. Werte großer als 8 sind nicht erlaubt, da das CAN-Protokoll nur maximal 8 Byte

Nutzlast pro Rahmen übertragen kann. In der Nutzlast wird das höchstwertige Byte zuerst übertragen. Ist die Länge der Nutzlast 0 schließt sich an das Kontrollfeld direkt das CRC-Feld an.

Das CRC-Feld dient der Datensicherung und enthält eine 15 Bit CRC-Prüfsequenz. Anhand dieser Prüfsumme kann ein Empfänger überprüfen, ob die Nachricht nachdem der Sender sie verschickt hat z.B. durch Störungen verändert wurde. Das sich anschließende Begrenzungsbit (*Delimiter*) ist der Spezifikation nach mit dem rezessiven Pegel belegt.

Das vorletzte Feld des Nachrichtenrahmens ist das Bestätigungsfeld (*ACK-Field*), welches aus einem Bestätigungsbit und einem Begrenzungsbit besteht. Das Bestätigungsbit (*ACK-Slot*) wird vom Sender mit einem rezessiven Pegel belegt und von jedem Empfänger, dessen CRC-Prüfung erfolgreich war, mit einem dominanten Bit überschrieben. So weiß der Sender und auch jeder Empfänger, bei dem die CRC-Prüfung fehlgeschlagen ist, dass es mindestens einen Empfänger gibt, welcher die Nachricht unverfälscht erhalten hat. Mit diesem Mechanismus kann erkannt werden, ob der Sender die Nachricht fehlerhaft gesendet hat bzw. die Nachricht auf dem ganzen Bus gestört wurde. Ein lokaler Fehler eines Empfängers kann hierbei nicht erkannt werden, da ein dominantes Bit empfangen wird, wenn nur ein Empfänger die Nachricht korrekt erhalten hat. Ein Empfänger, der einen Fehler in der CRC-Prüfung findet, sendet direkt im Anschluss an das Bestätigungsfeld eine Fehlernachricht. Dadurch erhält der Sender die Information, dass ein Empfänger die Nachricht nicht korrekt erhalten hat. Nach dem Bestätigungsbit folgt das rezessive ACK-Begrenzungsbit (*ACK-Delimiter*). Damit ist das Bestätigungsbit (*ACK-Slot*) durch zwei rezessive Bits, dem CRC-Begrenzungsbit und dem ACK-Begrenzungsbit, eingerahmt. Dies ist notwendig um eine positive Empfangsbestätigung von einer gleichzeitig beginnenden Fehlernachricht zu unterscheiden. Jede Datennachricht wird mit 7 rezessiven Bits (*EOF*, End Of Frame) abgeschlossen.

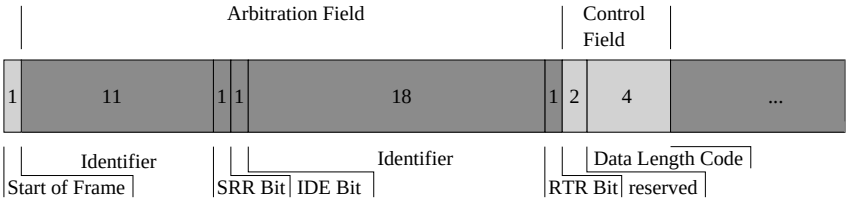


Abbildung 2.2: Erweiterter CAN-Rahmen nach [Rob91]

Eine CAN-Nachricht im **erweiterten Format** (*extended format*) unterscheidet sich durch Anpassungen im Arbitrierungs- und Kontrollfeld von einer Nachricht im Standardformat. Diese Unterschiede sind in Abbildung 2.2 dargestellt. Aufgrund der Kompatibilität des erweiterten Formats zum Standardformat ist der 29 Bit Identifier des Datenrahmens unterteilt. Das 12. Bit des Arbitierungsfeldes, welches im Standardformat dem RTR-Bit entspricht ist im erweiterten Format mit dem SRR-Bit (*Substitute Remote Request Bit*) belegt. Es ist immer rezessiv und dient nur als Platzhalter, da das eigentliche RTR-Bit nach dem Identifier platziert ist. Die Nutzung des RTR-Bits an der Position des SRR-Bits würde beim erweiterten Format dazu führen, dass die Priorität der Nachricht nicht mehr der Priorität des Identifiers entspricht. Es könnte dann dazu kommen das weniger wichtige Datennachrichten wichtigere Anforderungsnachrichten verdrängen. Das 13. Bit des Arbitrierungsfeldes wird durch das IDE-Bit belegt. Ist dieses Bit rezessiv handelt es sich um eine erweiterte CAN-Nachricht und es folgt der 18 Bit lange zweite Teil des Identifiers. Im Kontrollfeld unterscheiden sich beide Formate nur dadurch, dass

im erweiterten Format das erste Bit reserviert ist und im Standardformat dieses Bit dem IDE-Bit entspricht und dominant belegt sein muss. Der weitere Aufbau des erweiterten Formates ist identisch mit dem Standardformat.

Anforderungsnachrichten unterscheiden sich von Datennachrichten durch das RTR-Bit, welches bei Anforderungsnachrichten rezessiv und bei Datennachrichten dominant ist. Mit einer Anforderungsnachricht wird eine Datennachricht mit dem selben Identifier angefordert. Eine Anforderungsnachricht enthält keine Nutzlast. Die Längenangabe für die Nutzlast im Kontrollfeld wird auf den Wert gesetzt, welcher der Nutzlastlänge der angeforderten Datennachricht entspricht.

Fehler innerhalb von Daten- und Anforderungsnachrichten werden durch eine **Fehlernachricht** signalisiert. Dabei verletzt diese Nachricht die **Bitstuffingregel** und teilt so jedem Busteilnehmer den Fehler mit. Das CAN-Protokoll verwendet die NRZ-Bitcodierung (*Non Return to Zero*), bei der sich für gleiche aufeinanderfolgende Bitwerte der Buspegel nicht ändert. Um die Busteilnehmer auf die Bitzeiten zu synchronisieren, muss nach einer gewissen Anzahl von gleichen Bitwerten ein Pegelwechsel durchgeführt werden. Die Bitstuffingregel des CAN-Busses setzt daher durch, dass maximal 5 Bit selben Pegels aufeinander folgen dürfen. In der Fehlernachricht wird deshalb mit der Fehlersignalisierung (*Error Flag*) eine Sequenz von 6 gleichen Bits auf den Bus gelegt. Im CAN-Protokoll wird zwischen zwei Signalisierungen unterschieden: der aktiven und der passiven Fehlersignalisierung. Die **aktive Fehlersignalisierung** ist der Normalfall, bei der der Knoten im fehleraktiven (*Error Active*) Zustand ist. Dabei legt ein Knoten beim Erkennen eines Fehlers 6 dominante Bits als Error Flag auf den Bus. Hat ein Knoten eine mittlere Fehlererkennungsrate, welche über einem gewissen Grenzwert liegt, ist die Wahrscheinlichkeit höher, dass der Knoten selbst fehlerhaft ist. Ein solcher Knoten kann durch eine aktive Fehlersignalisierung den Bus blockieren und wird deshalb in einen fehlerpassiven (*Error Passive*) Zustand versetzt. Der fehlerpassive Knoten darf nur die **passive Fehlersignalisierung**, bei der 6 rezessive Bits zur Kennzeichnung eines Fehlers gesendet werden, nutzen. Dadurch kann ein sendender fehlerpassiver Knoten, beim Erkennen eines Fehlers in der eigenen Nachricht, den Sendevorgang sofort abbrechen und die Empfänger über den Fehler informieren. Einem fehlerpassiven Knoten ist es jedoch nicht möglich eine Nachricht eines anderen Knotens durch die Fehlersignalisierung zu zerstören. Dies ist nur fehleraktiven Knoten erlaubt.

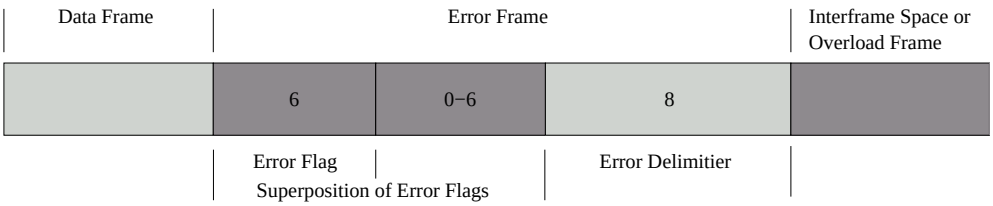


Abbildung 2.3: CAN-Fehlernachricht nach [Rob91]

Erkennt ein fehleraktiver Knoten einen Fehler, sendet er sofort eine aktive Fehlernachricht (primäre Fehlersignalisierung). Dadurch verletzt er das Format der Nachricht (von SOF-Bit bis CRC-Feld) oder des ACK- bzw. EOF-Feldes. Die anderen Knoten reagieren auf diese Verletzung der Bitstuffingregel mit dem Senden der Fehlernachricht (sekundäre Fehlersignalisierung). Wann diese zweite Fehlernachricht von den Knoten gesendet wird, ist abhängig davon wie viele dominante Bit direkt vor der aktiven Fehlersignalisierung in der Nachricht enthalten waren. Diese Überlagerung der Fehlersignalisierungen (*Superposition of Error Flags*) ist in Abbildung

2.3 dargestellt. Die sekundäre Fehlersignalisierung erfolgt spätestens nach den 6 dominanten Bits der primären Fehlersignalisierung, weil spätestens dort eine Verletzung der Bitstuffingregel festgestellt wird. Es wird durch die Fehlersignalisierung also eine Folge von dominanten Bits mit einer Länge von 6 bis 12 Bits erzeugt. Alle Busteilnehmer müssen aufgrund der globalen Datenkonsistenz eine fehlerhafte Nachricht verwerfen und der Sender muss die Nachricht erneut übermitteln.

Nach Ende der Fehlersignalisierung wird der Bus mit 8 rezessiven Bits zur Terminierung (*Error Delimiter*) belegt. Dazu sendet ein Knoten, der die 6 dominanten Bits zur Fehlersignalisierung gesendet hat, ein rezessives Bit. Stellt der Knoten fest, dass das Bit nicht durch ein dominantes Bit überschrieben wurde, sendet er 7 weitere Bit zur Beendigung der Fehlernachricht. In diesem Fall hat der Knoten den Fehler sekundär signalisiert. Ist das Aufschalten des ersten rezessiven Bits nicht erfolgreich, hat der Knoten den Fehler der Nachricht als erstes erkannt (primäre Fehlersignalisierung) und die anderen Knoten senden ihre sekundäre Fehlersignalisierung. Eine Ausnahme hierbei ist, wenn alle Busteilnehmer gleichzeitig einen Fehler erkennen, dann ist das Aufschalten des ersten rezessiven Bits der Fehlerterminierung immer erfolgreich, weil keine sekundäre Fehlersignalisierung erfolgt. Die Unterscheidung von primärer und sekundärer Fehlersignalisierung ist für die Erkennung fehlerhafter Knoten notwendig. Erkennt ein Knoten überdurchschnittlich oft einen Fehler als erstes und scheitert deshalb bei der Aufschaltung des ersten rezessiven Bits der Fehlerterminierung, ist davon auszugehen das er selbst fehlerhaft ist, er wird dann in den fehlerpassiven Zustand versetzt.

Jede Datennachricht und jede Datenanforderungsnachricht wird mit 7 rezessiven Bits abgeschlossen. Zählt man das rezessive ACK-Begrenzungsbit mit hinzu ergibt dies eine Sequenz von 8 rezessiven Bits am Ende der Nachricht. Damit ist es auch Empfängern, welche z.B. durch eine verfälschte Nutzlastlänge im Steuerfeld noch auf weitere Nutzdaten warten, noch vor Beendigung der Nachricht möglich eine Fehlernachricht zu senden. Diese Knoten erkennen anhand der Verletzung der Bitstuffingregel, dass es sich nicht mehr um das Nutzdatenfeld handelt sondern das EOF-Feld bereits erreicht ist. Sie beginnen dann sofort mit dem Senden ihrer Fehlernachricht und teilen dem Sender und allen anderen Knoten mit, dass die Nachricht nicht korrekt empfangen wurde. Damit können alle Knoten die Nachricht verwerfen und der Sender die Übertragung wiederholen.

Eine **Überlastnachricht** enthält zwei Felder: eines zum Kennzeichnen einer Überlastsituation (*Oberload Flag*), mit 6 dominanten Bits, und ein 8 Bit langes Feld zum Begrenzen der Überlastnachricht (*Overload Delimiter*). Die Überlastnachricht wird von einem Knoten dazu genutzt die nächste Daten- oder Anforderungsnachricht zu verzögern. Zur Verzögerung der Nachrichtenübermittlung muss die Überlastnachricht zu Beginn im Interframe Space[4] gesendet werden. Dabei dürfen maximal zwei Überlastnachrichten nacheinander zur Verzögerung der Nachrichtenübermittlung gesendet werden, ansonsten wird der Bus durch die Verzögerung zu lange blockiert. Hat ein Knoten ein dominantes Bit einer Überlastnachricht im Interframe Space empfangen, beginnt er ab den nächsten Bit selbst eine Überlastnachricht zu senden. Eine Überlastnachricht unterscheidet sich von einer Fehlernachricht dadurch, dass die Fehlernachricht innerhalb einer Daten- oder Anforderungsnachricht beginnt und eine Überlastnachricht danach.

[4] Damit beginnt die Überlastnachricht direkt nach dem EOF-Feld einer Daten- oder Anforderungsnachricht mit dem ersten Bit des Interframe Space.

Im CAN-Protokoll sind verschiedene Mechanismen zur **Fehlererkennung** enthalten [Ets94]:

- Bitüberwachung (durch den Sender)
- Überwachung des Nachrichtenformates (durch die Empfänger)
- CRC-Überprüfung (durch die Empfänger)
- Überwachung der Nachrichtenbestätigungen (durch den Sender)
- Überwachung der Bitstuffingregel (durch die Empfänger)

Wie bereits beschrieben, überwacht jeder sendende Knoten den Bus und prüft dadurch für jedes Bit, ob der Buspegel dem gesendeten entspricht (*Bit Monitoring*). Nur während der Arbitrierung und im ACK-Feld wird ein Überschreiben eines rezessiven Bits durch ein dominantes akzeptiert. Ansonsten handelt es sich um einen Fehler im sendenden Konten, um eine globale Störung des Busses oder eine lokale Busstörung beim Sender. Während des Empfangs einer Nachricht überprüft jeder Knoten die Form der Nachricht (*Message Frame Check*). Das Nachrichtenformat enthält, wie zuvor beschrieben, an definierten Stellen rezessive Begrenzungsbits. Befindet sich an der durch das Nachrichtenformat definierten Stelle kein solches Bit handelt es sich um einen Formatfehler (*Form Error*) der Nachricht. Mit Hilfe des CRC-Feldes in der Daten- und Anforderungsnachricht lassen sich mit einer hohen Wahrscheinlichkeit Fehler innerhalb der gesamten Nachricht finden. Der Empfänger kann durch die CRC-Überprüfung Fehler erkennen, welche durch lokale Störungen hervorgerufen wurden. Dabei lassen sich fünf beliebig verteilte oder ein Burst-Fehler mit einer Länge von bis zu 15 Bit korrekt erkennen [Ets94]. Damit ist die Fehlererkennung im CAN-Protokoll anderen Feldbussen überlegen.

Weiterhin überwacht der Sender einer Nachricht deren Empfang durch einen Empfänger. Dazu erwartet er ein dominantes Bit im ACK-Slot. Wurde eine Nachricht nicht bestätigt (das rezessive Bit im ACK-Slot wurde nicht durch ein dominantes überschrieben) geht der Sender davon aus, dass er einen Fehler beim Senden verursacht hat. Ein weiterer wichtiger Mechanismus zur Fehlererkennung ist die Überwachung der Bitstuffingregel im Empfänger. Die Bitstuffingregel wird neben der Erkennung von Fehlern auf dem Bus (z.B. durch einen defekten Bitstuffer oder durch vorübergehende Störungen des Busses) besonders bei der Fehlersignalisierung genutzt. Um einen erkannten Fehler den anderen Knoten zu signalisieren wird dabei eine Verletzung der Bitstuffingregel durch 6 aufeinander folgende Bits gleichen Pegels herbeigeführt. Wird ein Fehler in einer Daten- oder Datenanforderungsnachricht durch den Sender oder einen Empfänger erkannt, wird eine Fehlerbenachrichtigung gesendet, worauf jeder Knoten die Nachricht verwirft und der Sender die Nachricht nochmals zu übermitteln versucht. Dadurch kann die Anforderung der globalen Datenkonsistenz erfüllt werden. Durch die umgehende Fehlersignalisierung beim Erkennen eines Fehlers hat das CAN-Protokoll gegenüber anderen Feldbusprotokollen, welche teilweise erst nach Ablauf einer definierten Zeitspanne (z.B. *Timeout*) die Wiederholung einer gestörten oder nicht erhaltenden Nachricht initiieren, grundlegende zeitliche Vorteile.

Es wurde zuvor bereits angedeutet, dass Knoten, welche überdurchschnittlich viele Fehler erkennen, Fehler anders signalisieren als die restlichen Knoten. Dieser Mechanismus dient der **Fehlereingrenzung**, da ein fehlerhafter Knoten die gesamte Kommunikation auf dem Bus

stören oder gar blockieren kann. Die im CAN-Protokoll realisierte Fehlersignalisierung ist problematisch, weil dabei jeder Knoten zu jedem Zeitpunkt seine Fehlernachricht einfügen kann. Dadurch stört er den Empfang einer Nachricht und zwingt den Sender zu einer wiederholten Übermittlung. Somit muss ein fehlerhafter Knoten erkennen, dass er fehlerhaft ist und sich von der aktiven Fehlersignalisierung ausschließen. Im CAN-Protokoll erfolgt dies über zwei Zähler die jeder Knoten für sich verwaltet. Ein Zähler wird für den Empfang und der andere für das Senden von Nachrichten genutzt. Die Zähler werden jeweils erhöht, wenn ein Fehler beim Empfang bzw. Senden aufgetreten ist. Mit dem Empfang und dem Senden von fehlerfreien Nachrichten werden die Zählerstände wieder verringert. Dabei ist der Wert, der bei einem fehlerhaften Vorgang addiert wird, größer als der Wert, welcher beim Empfang oder Senden ohne Fehler, abgezogen wird. In Phasen in denen fehlerhafte Nachrichten auftreten, wird der entsprechende Zählerwert ansteigen, auch wenn deren Zahl unter der der fehlerfreien Nachrichten liegt. Erst wenn bedeutend weniger oder keine Fehler auftreten, kann sich der Zählerwert verringern. Mit diesen beiden Zählern lässt sich daher die relative Häufigkeit von Störungen abbilden.

Die Zähler werden abhängig davon, ob ein Knoten den Fehler zuerst (primäre Fehlersignalisierung) oder nur aufgrund einer Signalisierung eines anderen Knotens (sekundäre Fehlersignalisierung) erkannt hat, unterschiedlich erhöht. Dabei wird der Zähler des Knotens, der den Fehler zuerst erkannt hat, um ein vielfaches gegen über den anderen Knoten erhöht. Dies ist dadurch begründet, dass der oder die Knoten entweder lokal gestört werden oder selbst fehlerhaft sind. Da fehlerhafte Knoten von der aktiven Signalisierung ausgeschlossen werden sollen, werden die Knoten, welche den Fehler erkannt haben (primäre Fehlersignalisierung), bestraft.

Ein Empfänger, welcher überdurchschnittlich viele Fehler erkennt wird bei einem bestimmten Zählerstand seines Empfangszählers (> 127) in den passiven Fehlersignalisierungsmodus versetzt. Der Knoten nimmt, abgesehen von der Fehlersignalisierung, aber noch komplett an der Kommunikation teil. Ein fehlerhafter Sender wird beim Überschreiten einer Schwelle (> 127) erkannte Fehler nur noch passiv signalisieren. Fallen beide Zähler unter den Schwellwert (< 128), wird der Knoten wieder in den fehleraktiven Zustand versetzt und erkannte Fehler werden wieder aktiv signalisiert. Darüber hinaus werden sendende Knoten vom Bus getrennt (*bus off*), sofern ihr Sendezähler über eine weitere Schwelle (> 255) steigt. Vom Bus getrennte Knoten können keine Nachrichten empfangen oder senden. Sie können nur durch Reset, Konfiguration oder durch den Empfang einer $128 \cdot 11$ Bit langen Folge rezessiver Bits in den fehleraktiven Status wechseln und wieder an der Kommunikation teilnehmen.

Das CAN-Protokoll ermöglicht durch seine verlustfreie kollisionsauflösende Busarbitrierung einen priorisierten Medienzugriff, welcher wichtige Nachrichten bevorzugt übermittelt. Dabei wird vorausgesetzt, dass die Nachrichtenidentifier eindeutig sind. Aussagen über die Latenz der Nachrichten sind nur für die Nachrichten mit der höchsten Priorität möglich, da alle anderen Nachrichten die Arbitrierung verlieren, sofern eine Nachricht mit höherer Priorität zum selben Zeitpunkt an der Arbitrierung teilnimmt. Für die Nachricht mit der höchsten Priorität ist die Latenzzeit gleich der Summe aus der Restsendedauer der Nachricht, die zum Sendezeitpunkt den Bus belegt, und der Übermittlungszeit der zu sendenden Nachricht. Für andere Nachrichten muss, wie bei allen ereignisgesteuerten Modellen eine anwendungsbedingte statistische Abschätzung der Verzögerung vorgenommen werden. Bei der Betrachtung der Verzögerungen muss beachtet werden, dass für verschiedene Nachrichten unterschiedlich viele Stopfbits (durch den Bitstuffer) eingefügt werden. Dadurch entstehen nachrichtenspezifische zusätzliche Verzö-

gerungen. Die Interpretation des Nachrichtenidentifiers als Beschreibung des Inhaltes der Nachricht ermöglicht eine hohe Flexibilität der Kommunikation. Durch die Fehlererkennungs- und -signalisierungsmethoden des CAN-Protokolls können Fehler sicher erkannt und schnell kommuniziert werden. Damit wird eine globale Konsistenz der Daten erreicht, welche mit anderen Feldbussen, die eine serielle Übertragung von Nachrichten an die einzelnen Knoten nutzen, nur schwer realisierbar ist. Für weitere Informationen zum CAN-Protokoll wird auf die Quellen [Rob91, Ets94, Rei02] verwiesen.

2.3.2 Profibus

Das Profibus-Protokoll (*PROcess FIeld BUS*) ist ein klassischer Feldbus und wurde zwischen 1987 und 1991 von verschiedenen Forschungseinrichtungen und Unternehmen aus der Automatisierungstechnik entwickelt. Aus diesem Protokoll haben sich durch veränderte Anforderungen und die Erweiterung des Einsatzgebietes drei Ausprägungen entwickelt: Profibus-FMS, Profibus-DP und Profibus-PA.

Der **Profibus-FMS** (*Fieldbus Message Specification*) unterstützt eine Vielzahl von Diensten und ermöglicht die Verbindung von Sensoren/Aktoren mit Steuerungen sowie die Kommunikation der Steuerungen untereinander. Er kann in verschiedensten dezentralen Automatisierungssystemen eingesetzt werden. Im Gegensatz zu den vielfältigen Möglichkeiten des Profibus-FMS ist der **Profibus-DP** (*Decentralized Peripherals*) für den schnellen Austausch von Daten zwischen Sensor/Aktor und Steuerung entwickelt. Der **Profibus-PA** (*Process Automation*) ist dafür konzipiert in explosionsgefährdeten Bereichen eingesetzt zu werden und muss daher eine dem Explosionsschutz genügende physikalische Übertragung nach dem Standard IEC 61158-2 [IEC03] verwenden. Der Profibus-PA ist damit eigensicher und ermöglicht eine Fernspeisung der verteilten Knoten über den Bus. Damit kann beim Profibus-PA ein Feldgerät gefahrlos während des Betriebs einer explosionsgefährdeten Anlage an den Bus angeschlossen oder vom Bus entfernt werden. Der Profibus-PA ist aufgrund der Sicherheit mit 31.25 kBit/s im Vergleich zum Profibus-DP mit maximal 12 MBit/s (bei höchstens 100m Busausdehnung) sehr langsam.

Der Buszugriff bei Profibus verhindert durch das Token-Passing Verfahren Konflikte beim Medienzugriff. Dabei werden alle Knoten in passive Teilnehmer (*Slaves*) und aktiver Teilnehmer (*Master*) unterteilt. Nur Master-Knoten können direkt auf den Bus zugreifen. Slave-Knoten haben kein Zugriffsrecht und können somit nicht eigenständig Nachrichten versenden. Sie können nur den Empfang einer Nachricht quittieren und Anforderungen eines Master-Knotens beantworten. Befinden sich mehrere Master-Knoten an einem Bus, wird die Berechtigung zum Buszugriff (*Token*) zyklisch getauscht. Profibus bietet für die Kommunikation vier Dienste an: das Senden von Daten ohne Quittierung (*SDN, Send Data with No Acknowledge*), das Senden von Daten mit Quittierung durch den Empfänger (*SDA, Send Data with Acknoledge*), das Anfordern und Abholen sowie gleichzeitiges Senden[5] von Daten (*SRD, Send and Request Data with Reply*) und das periodische Anfordern, Abholen und Senden von Daten (*CSRD, Cyclic Send and Request Data with Reply*).

Profibus unterstützt aufgrund der Umsetzung des Master-Slave-Prinzips keine Multicast- oder Broadcast-Nachrichten, da der Master-Knoten zur Kommunikation die direkte Adresse des Kommunikationspartners benötigt. Der Master-Knoten stellt damit einen kritischen Punkt im Netz (*Single Point of Failure*) dar. Wenn dieser ausfällt, können die von ihm abgefragten

[5] Das Senden von Daten ist bei SRD optional.

Slave-Knoten (z.B. Sensoren, Aktoren) nicht mehr kommunizieren, obwohl sie intakt sind. Befinden sich mehrere Master-Knoten am Bus und ein Master-Knoten gibt das Token aufgrund einer Fehlfunktion nicht weiter, kann es zu langen Verzögerungen der Kommunikation kommen. Die anderen Master-Knoten müssen dann eine gewisse Zeitspanne nach dem geplanten Tokenwechsel abwarten, um den Fehler zu erkennen. Erst dann kann ein neues Token erzeugt und die Kommunikation wieder aufgenommen werden. Der Profibus ist für die Übermittlung von kurzen häufigen Nachrichten aufgrund der Effizienz des Multi-Master-Slave-Prinzips nicht gut geeignet. Er ist daher für die Übertragung von Echtzeitinformationen nicht optimal [Ets94]. Das Anwendungsgebiet von Profibus ist aufgrund seiner Funktionalität die Kommunikation auf Feld- oder die Prozessleitebene. Eine ausführliche Beschreibung von Profibus ist in [Rei02] zu finden.

2.3.3 TTP/A

TTP/A ist ein zeitgesteuertes Protokoll für die Kommunikation von intelligenten Sensoren und Aktoren in einer eingebetteten Echtzeitumgebung. Es bietet Dienste zur zeitgerechten Kommunikation, zur Laufzeitdiagnostik und einfachen Integration von Knoten (*Plug-And-Play*) an. TTP/A ist so konzipiert, dass es sich gut in eine Lösung mit dem TTP/C Protokoll (siehe Unterabschnitt 2.3.5) integrieren lässt. Es werden wie bei Profibus Master- und Slave-Knoten unterschieden. Da TTP/A zeitgesteuert ist und die Nachrichten anhand eines Zeitplans übermittelt werden, muss der Master-Knoten die Synchronisation übernehmen. Er verfügt daher über einen genauen Zeitgeber und definiert für jede Runde den Rundenbeginn, so dass die Slave-Knoten nur innerhalb jeder Runde synchron bleiben müssen. Sie können daher einfache Mikrocontroller mit freilaufenden Oszillatoren verwenden. TTP/A unterstützt Master-Slave- und Multipartner-Runden. Die Master-Slave-Runde ist eine Punkt-zu-Punkt Kommunikation, die für Verwaltungs- und Konfigurationszwecke (*Configuration Planning Interface*) oder als Diagnoseschnittstelle (*Diacnostic and Maintainence Interface*) dient. In einer Multipartner-Runde werden bis zu 64 Byte Echtzeitdaten übertragen (*Real-Time Service Interface*). Die Daten in der Multipartner-Runde richten sich nicht an einen speziellen Knoten, es handelt sich vielmehr um Broadcasts.

In TTP/A liegen Daten im so genannten *IFS* (*Interface File System*). Das IFS ist ein index-sequenzielles Dateisystem mit Datenfeldern einer festen Größe von 4 Byte Daten und 1 Prüf-Byte. Ein Knoten kann intern bis zu 60 Dateien mit je 255 Datenfeldern adressieren. Das IFS eines Clusters setzt sich aus der Vereinigungsmenge der IFS der sich im Cluster befindenden Knoten zusammen. TTP/A definiert drei Operationen auf dem IFS: Lesen oder Schreiben eines Datenfeldes und das Ausführen einer Datei.

Eine Runde wird mit einem speziellen Befehlsrahmen (*Command Frame*) vom Master-Knoten begonnen. Der Master-Knoten gibt im Befehlsrahmen an, wie die Kommunikation der Runde organisiert ist. Das erste Byte dieses Rahmens (*Fireworks Byte*) dient der Synchronisierung der Slave-Knoten für die beginnende Runde. In den folgenden 3 Byte wird definiert, ob es sich um eine Master-Slave-Runde oder um eine Multipartner-Runde handelt. Das letzte Byte ist ein Prüf-Byte. In **Master-Slave-Runden** wird entweder direkt ein Wert aus dem IFS eines Slave-Knoten gelesen oder geschrieben. Eine Master-Slave-Runde besteht dabei aus einem Befehlsrahmen und einem Datenrahmen. Im Befehlsrahmen wird festgelegt welcher Wert in welchem Knoten geschrieben bzw. gelesen werden soll. Innerhalb des folgenden 5 Byte (4

Byte Daten und 1 Prüf-Byte) großen Datenrahmen sendet entweder der Master-Knoten oder der betreffende Slave-Knoten den entsprechenden Wert.

Zu Beginn der **Multipartner-Runden** definiert der Master-Knoten welchen Nachrichtenplan (*RODL, Round Decription List*) die Slave-Knoten für diese Runde nutzen sollen. Die RODL ist im IFS abgelegt und wird über einen Befehl (`execute-RODL-file`) aktiviert. Im Nachrichtenplan sind für jede Nachricht unter anderem folgende Informationen enthalten: die Nachrichtenlänge, welcher Knoten die Nachricht wann sendet und welcher Knoten die Nachricht empfängt. Die Adressierung der Daten erfolgt damit über die RODL, bei der ein Zeitschlitz in einer Multipartner-Runde einer eindeutigen IFS-Adresse in einem Knoten zugeordnet ist. Im Gegensatz zur Master-Slave-Runde enthält die Multipartner-Runde nicht nur eine Nachricht als Antwort sondern kann bis zu 64 Byte übertragen. Damit können mehrere Slave-Knoten ihre Echtzeitdaten in einer Multipartner-Runde senden.

Im Normalfall wird bei TTP/A eine feste Reihenfolge der Multipartner- und Master-Slave-Runden definiert. Die periodischen Multipartner-Runden tauschen Echtzeitdaten aus und die sporadisch auftretenden Master-Slave-Runden bedienen die Diagnose- und die Konfigurations- und Wartungsschnittstelle. Dabei ist die Zahl der Master-Slave-Runden davon abhängig wie viel Bandbreite der Diagnoseschnittstelle bzw. der Konfiguration zugewiesen werden soll. Die Master-Slave-Runden können die Echtzeiteigenschaften der in den Multipartner-Runden übertragenen Daten beeinträchtigen. Dazu ist je nach Belastung des Busses durch Echtzeitdaten die Anzahl der Master-Slave-Runden zu begrenzen oder in kritischen Situationen sind sie zeitweise komplett auszulassen.

TTP/A wurde für die Übertragung von Echtzeitinformationen sowie die Verwaltung, Konfiguration und Diagnose von intelligenten Sensoren oder Aktoren entwickelt. Es hat daher andere Anforderungen als z.B. TTP/C. So wird z.B. eine geringere Zuverlässigkeit toleriert. Für die Übertragung von Sensordaten genügt im Allgemeinen eine geringere Bandbreite. TTP/A kann mit kostengünstigen Übertragungsmedien wie z.B. einer Ein-Draht-Verbindung (ISO-K [ISOa]) bis zu 20 kBit/s Bandbreite bereitstellen[6]. Weiterhin wurde bei der Protokollentwicklung darauf Wert gelegt, dass die Slave-Knoten möglichst einfache und dadurch preisgünstige Hardwareanforderungen haben. Obwohl TTP/A ein zeitgesteuertes Protokoll ist, genügt es daher die Slave-Knoten mit einfachen freilaufenden Uhren zu betreiben, da der Master-Knoten jede Runde eine Synchronisation der Slave-Knoten vornimmt. Die Möglichkeit der Konfiguration und der Diagnose von TTP/A-Knoten ist die wesentliche Eigenschaft mit der sich TTP/A von LIN abgrenzt. Das Design von TTP/A erlaubt es z.B. zur Laufzeit Diagnoseinformationen von einem Sensor zu erhalten, ohne dabei die Echtzeitkommunikation auf dem Bus zu beeinträchtigen. Auch eine dynamische Integration neuer Knoten ist in TTP/A möglich. Im Vergleich zu Profibus ist TTP/A für Echtzeitkommunikation entwickelt und ermöglicht über Multipartner-Runden das gleichzeitige und damit effiziente Abfragen mehrerer Slave-Knoten. TTP/A ist dafür ausgelegt mit TTP/C zusammenzuarbeiten, wobei der Master-Knoten im TTP/C-Netz zu finden ist und dort die Sensor-Daten publiziert. Weitere Informationen zu TTP/A sind in [Kop97, KEM00, EHK+05] zu finden.

[6] Mittels Glasfasertechnik kann TTP/A Bandbreiten von über 1 MBit/s erreichen.

2.3.4 LIN

LIN (*Local Interconnect Network*) ist von den Anwendungsgebieten und dem Konzept mit TTP/A vergleichbar. LIN ist ein zeitgesteuerter Master-Slave Feldbus der für die Echtzeitkommunikation mit Sensoren entwickelt wurde. Dabei stellt auch LIN nur geringe Anforderungen an die Feldgeräte und verringert somit deren Kosten. Der Master-Knoten übernimmt daher die Synchronisation und Koordination der Feldgeräte (Slave-Knoten). LIN unterstützt wie TTP/A auch besonders preisgünstige Ein-Draht-Medien, da bei Feldbussen, neben den Kosten für die Feldgeräte, auch der Kosten-Faktor des Mediums entscheidend ist. Wie bei allen Master-Slave-Protokollen wird die Kommunikation durch Befehle vom Master-Knoten gesteuert. Auf einen Befehl vom Master-Knoten folgt ein Datenrahmen. Im LIN Protokoll ist der zeitliche Abstand (*Interframe Gap*) zwischen dem Befehls- und dem Datenrahmen im Gegensatz zu TTP/A nicht konstant. Daraus folgt, dass der Sendebeginn der Daten veränderlich ist und ein Jitter, der Zeiten zu denen die Daten gesendet werden, von maximal 40% der Rundenzeit entsteht [KEM00]. Durch einen solchen Jitter wird die Echtzeitqualität der Daten eingeschränkt. Im Gegensatz zu TTP/A ist die Diagnose der Knoten durch eine entfernte Komponente nicht vorgesehen. Daher gibt es keine Möglichkeit einen Knoten in LIN direkt zu adressieren. Auch die dynamische Konfiguration der Knoten ist nicht möglich. Die Knoten müssen damit vor Inbetriebnahme mit den notwendigen Informationen versorgt werden.

LIN ist ein Protokoll für die Echtzeitdatenübertragung im Feldbereich. Im Gegensatz zu TTP/A liegt der Fokus des Protokolls auf der Datenübertragung. Durch die Einschränkung der Funktionen von LIN, wird die Implementierung der Slave-Knoten (z.B. Sensoren) sehr einfach gehalten. Weiterhin verfügt LIN nur über begrenzte Fehlererkennungs- und Fehlerbehandlungsmechanismen. Für die Beschreibung der Nachrichtenstruktur sowie weiterführende Informationen zum LIN Protokoll, wird auf die LIN Spezifikation [LIN00] und auf [ZS06] verwiesen.

2.3.5 TTP/C

TTP/C ist ein zeitgesteuertes Echtzeitkommunikationsprotokoll und gehört mit TTP/A zur TTA (*Time Triggered Architecture*), welche von [Kop97, KB03] beschrieben wird. TTP/A richtet sich an einfachere low-cost Anwendungen, wie z.B. für Feldbusse. TTP/C zielt auf fehlertolerante sicherheitskritische Echtzeitsysteme und wurde entwickelt, um den hohen Anforderungen an Sicherheit, Verfügbarkeit und Zusammensetzbarkeit speziell von automotiven und avionischen Systemen sowie der industriellen Automatisierungstechnik zu genügen. TTP/C versucht diese gewünschten Eigenschaften mit Hilfe verschiedener Konzepte wie physische Redundanz, Temporal Firewall, Bus-Guardian, fehlertolerante Uhrensynchronisation und Membershipsservice zu erreichen.

Der Medienzugriff in TTP/C erfolgt mit TDMA auf zwei physisch voneinander unabhängigen Kanälen über Zeitschlitze. Um eine einfache Fehlertoleranz zu erreichen, wird im Allgemeinen jede Nachricht gleichzeitig über beide Kanäle gesendet. Es gibt zwei grundlegende Topologien für TTP/C (Bus und Stern) sowie mögliche Kombinationen daraus z.B. Multi-Star Topologie. Eine Reihenfolge von Zeitschlitzen, bei denen jeder Knoten Nachrichten senden kann, bildet eine TDMA-Runde (*TDMA round*). Eine feste Aneinanderreihung von TDMA-Runden wird als Clusterzyklus (*cluster cycle*) bezeichnet. Der Clusterzyklus definiert die maximale Periode einer Nachricht. Innerhalb eines Clusterzyklus muss jeder Knoten mindestens eine Nach-

richt einmal senden. Der Zugriff auf das Medium wird durch den Kommunikationscontroller realisiert. Die zu sendenden Daten erhält der Kommunikationscontroller vom Applikationsrechner (*host computer*) durch das CNI (*communication network interface*), welches durch eine zweiseitig beschreibbare Speicherstruktur (*dual ported random access memory, DPRAM*) umgesetzt wird. Die Datenintegrität im CNI wird durch das *Non-Blocking-Write* Protokoll [KR93] garantiert.

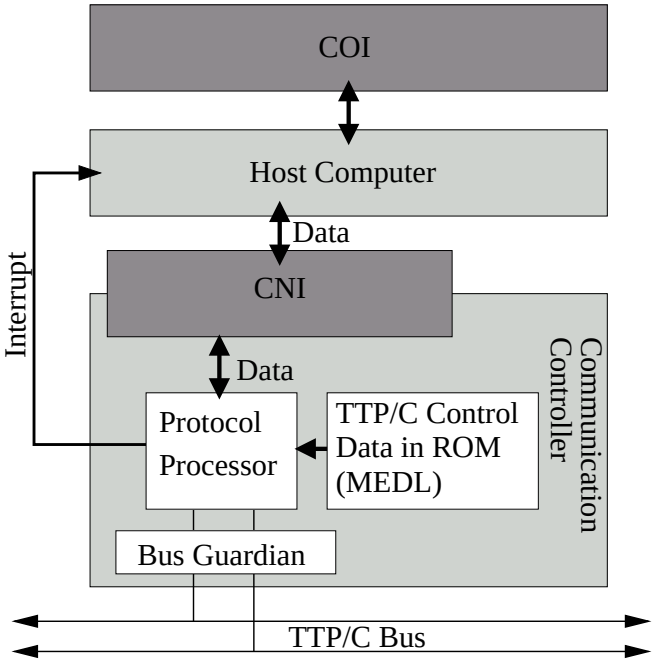


Abbildung 2.4: TTP/C-Knoten/Netzwerkstruktur nach [TTA03]

Der Kommunikationscontroller jedes Knotens kennt den Kommunikationsplan und greift daher auf das Medium zu. Empfängt oder sendet der Controller eine Nachricht werden die Daten in das DPRAM abgelegt bzw. daraus gelesen. Eine Applikation muss, wenn eine Nachricht abgesendet werden soll, sie an das CNI übergeben (*information push interface)* und wenn eine Nachricht gelesen werden soll, sie aus dem CNI lesen (*information pull interface*). Es findet damit, wie in Abbildung 2.4 dargestellt, kein **Kontrollfluss** zwischen der Anwendung und dem Kommunikationscontroller statt. Mit der Entkopplung des Kontrollflusses wird die Ausbreitung von Fehlern der Anwendung begrenzt. Die einzigen Kontrollnachrichten die vom Kommunikationscontroller zum Applikationsrechner gelangen sind die periodischen Unterbrechungen der globalen synchronisierten Zeit und Ausnahmen durch Fehler oder asynchrone Ereignisse.

Der **Bus-Guardian** (*BG*) stellt sicher, dass nur zu den im Kommunikationsplan definierten Zeiten auf das Medium zugegriffen wird. Versucht der Kommunikationscontroller, z.B. durch einen Fehler, eine Nachricht zu senden, wenn der Kommunikationsplan dies nicht erlaubt, wird er vom Bus-Guardian daran gehindert. Der Bus-Guardian setzt die Direktive, dass im Fehlerfall keine Ausgaben produziert werden (*fail-silent*), für das zeitliche Verhalten des Kommunikationscontrollers durch. Damit wird sichergestellt, dass ein Knoten durch Fehler anderer Teilnehmer (*babbling idiot*) in seiner Kommunikation nicht gestört wird. Der Bus-Guardian kann

im Kommunikationscontroller physisch integriert oder als eine eigene Komponente umgesetzt werden.

Der Kommunikationscontroller hält den Kommunikationsplan, welcher in TTP/C als *Message Descriptor List* (*MEDL*) bezeichnet wird. In dieser Liste ist genau ein Clusterzyklus enthalten, welcher aus mehreren TDMA-Runden bestehen kann. Eine **MEDL** enthält nach [Kop97] mindestens folgende Einträge:

- **Zeit**: Wann die Nachricht gesendet wird.
- **Adresse**: Wo die zu übertragenden Daten zu finden sind oder wo die empfangenen Nachrichten abgelegt werden.
- **D-Attribut**: Die Richtung (*Direction*) der Nachricht: Eingang (Empfangen) oder Ausgang (Senden).
- **L-Attribut**: Die Länge der Nachricht.
- **I-Attribut**: Die Art der Nachricht: Entweder handelt es sich um eine normale oder um eine Initialisierungsnachricht.
- **A-Attribut**: Weitere Attribute z.B. für den Wechsel des Cluster-Modus.

Der Kommunikationsplan wird global erzeugt und jeder TTP/C-Knoten erhält eine eigene individuelle Version des globalen Plans, bei der die entsprechenden Einträge (z.B. Adresse und D-Attribut) für den Knoten abgestimmt sind. Die einzelnen individuellen lokalen MEDLs müssen zusammen wieder den globalen Plan konsistent beschreiben. Die Zuordnung der Nachrichten im Knoten wird über die lokale MEDL realisiert. Empfangene Nachrichten werden in den festgelegten Speicherbereich im CNI abgelegt und können so von der Applikation gelesen werden. Die Verknüpfung des Speicherbereichs und der Nachricht, erfolgt über die in der MEDL festgelegte Empfangszeit der Nachricht.

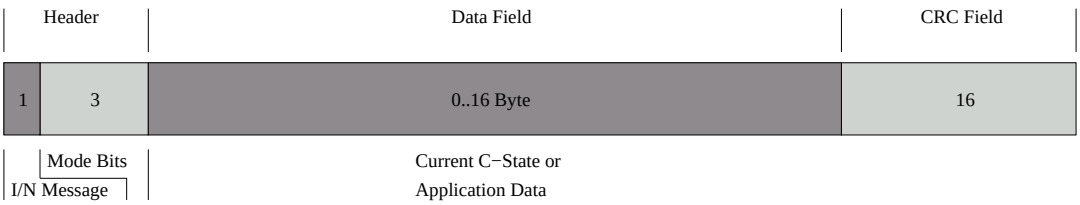


Abbildung 2.5: TTP/C-Rahmen nach [Kop97]

In der Abbildung 2.5 ist der allgemeine Aufbau eines TTP/C-Rahmens dargestellt. Aus der Abbildung wird schnell ersichtlich, dass keine Sender oder Empfangsadresse in einer Nachricht enthalten ist. Deren Angabe ist aufgrund der Zuordnung der Nachrichten über die MEDL nicht notwendig. Daher enthalten TTP/C-Nachrichten einen hohen Anteil an Nutzdaten. Der Kopf der Nachricht (*Header*) ist nur 4 Bits groß und am Ende des Rahmens wird eine 16 Bit Prüfsumme (*CRC*) angehängt. Eine TTP/C-Nachricht kann 16 Bytes Nutzdaten aufnehmen.

Das erste Bit (*I/N-Bit*) des Headers gibt an, ob es sich bei der Nachricht um eine Initialisierungsnachricht (*I-Frame*) oder um eine normale Nachricht (*N-Frame*) handelt. Normale

Nachrichten enthalten Anwendungsdaten für den Informationsaustausch. Initialisierungsnachrichten dienen der Initialisierung des Systems. Wenn auf die Initialisierungsdaten eines I-Frames noch Nutzdaten folgen, wird von einem *X-Frame* (*extended Frame*) gesprochen. Initialisierungsnachrichten enthalten den Status des Kommunikationscontrollers (*C-State*) und ermöglichen es so Knoten in die Kommunikation zu (re-)integrieren. Der **C-State** eines Controllers umfasst fünf Angaben:

- **Globale Zeit**: Entspricht der globalen Sendezeit der Nachricht. Wird zu Beginn des Zeitschlitzes gesetzt.
- **Position des Zeitschlitzes**: Gibt den aktuellen Zeitschlitz im Clusterzyklus bzw. den aktuellen Eintrag im Kommunikationsplan an. Wird zu Beginn des Zeitschlitzes gesetzt.
- **Cluster-Modus**: Beschreibt den aktuellen Cluster-Modus. Ein Cluster-Modus ist an einen Clusterzyklus (und die entsprechenden MEDLs) gebunden. Mit dem Wechsel des Cluster-Modus (*Mode Change*) kann zur Laufzeit das Kommunikationsverhalten geändert werden. Wird zu Beginn des Zeitschlitzes gesetzt.
- **DMC** (*Deferred Pending Mode Changes*): In diesem Feld wird ein angeforderter Cluster-Moduswechsel vermerkt. Der Moduswechsel wird zu Beginn des nächsten Clusterzyklus durchgeführt. Eine Änderung wird nach dem Lesen der anfordernden Nachricht vorgenommen.
- **Mitgliedschaftsinformationen** (*Membership*): Enthält die konsistente Sicht auf den Gruppenstatus der Knoten. Ein Knoten gehört zur Gruppe, wenn er in seinem letzten Zeitschlitz eine Nachricht korrekt gesendet hat. Ein Knoten wird aus der Gruppe ausgeschlossen, sofern er zuletzt keine Nachricht oder eine fehlerhafte Nachricht gesendet hat. Der Algorithmus zur Berechnung der Mitgliedschaft wird nach dem Empfang einer Nachricht ausgeführt. Für weiterführende Informationen zum Membershipsservice sei auf [TTA03] verwiesen.

Der C-State ist in jedem kommunizierenden Knoten identisch. Daher kann anhand des Vergleichs dieser Statusinformationen festgestellt werden, ob ein Knoten im Netz fehlerhaft ist. Dazu wird in der Prüfsumme jeder normalen Nachricht der C-State implizit übertragen. Initialisierungsnachrichten, welche den C-State direkt übermitteln, werden für die Integration von Knoten oder für den Start des Clusters benötigt. Die Häufigkeit mit der Initialisierungsnachrichten gesendet werden bestimmt die Dauer eine (Re-)Integration von Knoten in die Kommunikation. Bei der Integration wird der C-State des Senders vom zu integrierenden Knoten übernommen. Neben den beschriebenen Nachrichtentypen gibt es *cold start frames*, welche beim Clusterstart gesendet werden. Sie enthalten die globale Sendezeit, die Sendeposition in der TDMA-Runde sowie die Prüfsumme.

Im TTP/C-Protokoll ist es möglich den Kommunikationsplan zur Laufzeit zu wechseln. Damit kann eine TTP/C-Anwendung sich auf verschiedene vorher definierte Kommunikationssituationen anzupassen. So ist es für unterschiedliche Phasen der Applikation (z.B. im Flugzeug die Phasen: Am Boden, Start, Flug, Landung) möglich speziell angepasste Kommunikationspläne zu verwenden. Dazu werden verschiedene Cluster-Modi mit unterschiedlichen Kommunikationsplänen in der MEDL definiert. Bei einem Moduswechsel muss garantiert werden, dass

alle Knoten zur selben Zeit in den gleichen Modus wechseln. Um den korrekten Moduswechsel sicherzustellen, ist im C-State das DMC-Feld enthalten. Ein Knoten kann mit den 3 *Mode-Bits* im TTP/C-Rahmen (siehe Abbildung 2.5) eine Anforderung für einen Moduswechsel stellen. Dabei sind aus dem aktiven Cluster-Modus maximal 7 Folgemodi zu erreichen. Die Folgemodi sind abhängig vom aktuellen Cluster-Modus, d.h. für jeden Modus sind verschiedene Folgemodi möglich. Ein Moduswechsel wird zu nur Beginn des Clusterzyklus durchgeführt.

Im letzten Feld des TTP/C-Rahmens ist die 16 Bit **Prüfsumme** abgelegt. In die Berechnung der Prüfsumme geht bei jeder Nachricht (außer bei den *cold start frames*) der aktuelle C-State des Senders mit ein. N-Frames übertragen somit den C-State implizit in der Prüfsumme. Zur Überprüfung des CRC-Wertes durch den Empfänger, muss daher für N-Frames neben der empfangenen Nachricht auch der eigene C-State in die Prüfsummenberechnung mit einbezogen werden. Zusätzlich dazu wird in die Berechnung des CRC-Wertes immer (auch bei *cold start frames*) eine eindeutige *Schedule-ID* des Clusters eingefügt. Diese ID verhindert die Kommunikation von Knoten, welche z.B. aufgrund von Fehlern unterschiedliche Kommunikationspläne verwenden. Dadurch dass jeder Knoten beim Empfang jeder Nachricht seinen C-State mit den C-State des sendenden Knoten vergleicht, lässt sich eine konsistente Sicht jedes Knotens auf das gesamte System erreichen und Fehler in den C-States anderer Knoten können schnell erkannt werden. Dabei muss darauf hingewiesen werden, dass bei der impliziten C-State Übermittlung eine fehlerhafte Prüfsumme nicht auf einen unterschiedlichen C-State des Senders zurückgehen muss, es ist ebenso möglich, dass die Prüfsumme oder die Nachricht durch einen Übertragungsfehler verfälscht wurde.

Ein zeitgesteuertes Kommunikationsprotokoll erfordert die **Synchronisation** der einzelnen lokalen Uhren. Im TTP/C-Protokoll wird dazu beim Empfang jeder Nachricht die tatsächliche Empfangszeit mit der in der MEDL erwarteten Empfangszeit verglichen. Dadurch kann für jeden Zeitschlitz und damit für jeden Sender ein eigener Korrekturterm erzeugt werden. Aufgrund des Vergleichs der Empfangszeiten, werden für die Anpassung der Zeiten keine speziellen Synchronisationsnachrichten oder Zeitinformationen innerhalb der Nachrichten benötigt. Die Nachrichtenlänge oder der Verbrauch der Bandbreite wird also nicht durch die Synchronisation beeinflusst. Für die Synchronisierung der einzelnen Uhren nutzt das TTP/C-Protokoll den *Fault-Tolerant Average* (*FTA*) Algorithmus. Als Synchronisationszeitquelle können die internen Uhren der Knoten oder eine externe Zeitquelle verwendet werden. Da eine detaillierte Beschreibung der Uhrensynchronisation im TTP/C-Protokoll den Rahmen dieser Einführung verlässt, wird auf [TTA03, KHK+97] verwiesen.

TTP/C bietet die Möglichkeit mehrere Knoten als eine **fehlertolerante Einheit** (*FTU*) zu betreiben. In einer FTU muss sichergestellt werden, dass nur permanente Nachrichten[7] in die CNI der FTU gelangen. Das TTP/C-Protokoll stellt auf der FT-COM Schicht (*fault-tolerance communication layer*) die notwendigen Dienste zum Redundanzmanagement und zur Permanenz der Nachrichten zur Verfügung. Weitere Informationen zu den FTUs sind in [TTA03, Kop97] zu finden.

Das TTP/C-Protokoll bietet aufgrund des zeitgesteuerten Medienzugriffs eine deterministische Kommunikation und dadurch eine konstante Nachrichtenverzögerung. Die Bandbreite von

[7] Nach [Kop97] beschreibt die Permanenz die Abhängigkeit empfangener Nachrichten von der Reihenfolge ihrer Sendezeitpunkte. Eine Nachricht wird erst dann permanent, wenn alle zuvor gesendeten Nachrichten empfangen wurden oder sichergestellt ist, dass sie nicht ankommen werden. Es ist für eine detailliertere Darstellung dieser Abhängigkeit auf [Ver94] verwiesen.

TTP/C ist mit 2 MBit/s spezifiziert, es sind aber auch höhere Bitraten möglich. Durch die Adressierung über die MEDL, werden keine Adressinformationen oder Daten zur Beschreibung der Nutzlast benötigt. Eine TTP/C-Nachricht hat daher nur einen kleinen Protokollüberbau. Durch die in das Protokoll integrierten Mechanismen zur Fehlererkennung und Fehlertoleranz können verschiedene Fehlersituationen erkannt und behandelt werden. Über den Vergleich der MEDL mit den eingehenden Nachrichten, kann festgestellt werden, ob ein Knoten seine Nachrichten rechtzeitig übermittelt. Wird in einem Zeitschlitz keine Nachricht empfangen, muss davon ausgegangen werden, dass der diesem Zeitfenster zugeordnete Knoten entweder ausgefallen oder fehlerhaft ist. Mit Hilfe des Membership-Dienstes des TTP/C-Protokolls ist jeder Kommunikationsteilnehmer jederzeit darüber informiert, welche Knoten fehlerhaft sind. Somit kann zur Laufzeit die TTP/C-Anwendung so angepasst werden, dass der Ausfall einzelner Komponenten kompensiert wird. Fehler innerhalb einer Nachricht werden weitestgehend durch die CRC-Summe erkannt. Die Reparatur einer Nachricht ist aber nicht möglich. Auch ein wiederholtes Senden der Nachricht im Fehlerfall ist in TTP/C nicht vorgesehen, da dies die Vorhersagbarkeit des Protokolls einschränken würde. Um die Wahrscheinlichkeit zu verringern, dass eine Nachricht durch einen Fehler nicht gelesen werden kann, können in TTP/C die Nachrichten mehrfach gesendet werden. Dabei werden die Nachrichten nacheinander und/oder parallel auf beiden Kanälen gesendet. Ist z.B. ein Kanal gestört, kann die Nachricht aus dem anderen Kanal verwendet werden. Für weiterführende Informationen zum TTP/C-Protokoll werden die Quellen [TTA03, Kop97, KB03, Rus03] genannt.

2.3.6 FlexRay

Das FlexRay Protokoll ist speziell für den automotiven Bereich entwickelt worden. Die von FlexRay verwendeten Konzepte finden sich auch in den Protokollen TTP/C [TTA03] und ByteFlight [PBG99] wieder. Wie bei TTP/C sind für FlexRay unterschiedliche Netztopologien wie Bus und Stern sowie verschiedene Kombinationen daraus möglich. Das Protokoll kann sowohl einkanalig als auch zweikanalig betrieben werden. Im Weiteren wird von einem Betrieb mit zwei Kanälen ausgegangen. FlexRay sichert die feste Nachrichtenlatenz für zeitkritische Nachrichten und ermöglicht dazu die flexible Handhabung asynchroner Nachrichten. Dabei erfolgt die Kommunikation abwechselnd zeit- und ereignisgesteuert. Im periodischen Kommunikationszyklus stehen dazu zwei Segmente zur Verfügung: das statische Segment, in dem feste Zeitschlitze für die Kommunikation vorgesehen sind und das dynamische Segment, bei dem Nachrichten nach ihrer Priorität gesendet werden. Beide Segmente sind durch ein TDMA-Schema voneinander getrennt.

Innerhalb des **statischen Segments** wird der Buszugriff ebenfalls mit TDMA realisiert. Das statische Segment besteht aus Zeitschlitzen gleicher Länge, welche im Voraus fest konfiguriert werden. Dadurch müssen im Kommunikationsplan nicht die Zeiten der Sende- bzw. Empfangszeitschlitze abgelegt werden, sondern es genügt die Nummer des Slots. Der Zugriff wird daher in den Knoten über zwei Zähler (einen für jeden Kanal) ermöglicht. Beide Zähler werden hochgezählt, wenn das Ende eines Zeitschlitzes erreicht ist. Durch die zwei separaten Slotzähler, kann ein Knoten Nachrichten auf einem oder auf beiden Kanälen senden bzw. empfangen.

Das **dynamische Segment** steuert den Medienzugriff mittels eines *Mini-Slotting-Protokolls*. Wie beim statischen Segment hält jeder Kanal einen eigenen Zähler, welche aber unabhängig voneinander erhöht werden können. Dies ist notwendig, da beide Kanäle separat

benutzt werden können. Das dynamische Segment wird in eine feste Anzahl von gleich großen Mini-Slots eingeteilt. Jeder Nachricht, welche in einem dynamischen Segment versendet wird, ist eine eindeutige ID zugeordnet. Diese ID entspricht der Position eines Mini-Slots im dynamischen Segment. Der Medienzugriff ist mittels des Zählers für die Mini-Slots realisiert. Erreicht der Zähler den Wert einer ID wird die zugehörige Nachricht gesendet. Je nach Länge der Nachricht werden unterschiedlich viele Mini-Slots für die Übermittlung benötigt. Der Zähler wird entweder nach einer gesendeten Nachricht oder, sofern keine Nachricht mit der ID des aktuellen Zählerstandes verfügbar ist, nach Ende des Mini-Slots erhöht. Ist die im dynamischen Segment festgelegte Anzahl der Mini-Slots erreicht, ist das dynamische Segment beendet. Es spielt dabei keine Rolle, ob die Mini-Slots durch Nachrichten belegt oder nur zur Erhöhung des Zählers genutzt wurden. Das Mini-Slotting-Protokoll setzt eine prioritätsbasierte Nachrichtenübermittlung im dynamischen Segment durch. Nachrichten mit kleiner ID entsprechen einer hohen Priorität und werden bevorzugt gesendet. Ist einer Nachricht eine hohe ID zugewiesen, muss der Zähler im dynamischen Segment einen hohen Wert erreichen, um die Nachricht zu senden. Dabei ist nicht sichergestellt, dass der Zähler im aktuellen dynamischen Segment diesen Wert auch erreicht und die Nachricht gesendet werden kann. Wenn durch die Übermittlung von Nachrichten mit höherer Priorität (und kleinen IDs) bereits so viele Mini-Slots verbraucht wurden, dass der Zählerstand nicht alle IDs vor Ablauf des dynamischen Segmentes erreicht, können ausstehende Nachrichten in dieser Periode nicht gesendet werden. Werden in der darauf folgenden Periode weniger hochpriore Nachrichten gesendet, dann können diese Nachrichten mit geringer Priorität übermittelt werden. Es kann somit nicht garantiert werden, dass eine bestimmte Nachricht in einem dynamischen Segment gesendet wird. Es sei denn die Nachricht gehört zu den n Nachrichten mit kleinster ID, welche komplett in das dynamische Segment passen. Das CAN-Protokoll und das Mini-Slotting-Protokoll von FlexRay ermöglichen beide eine priorisierte Nachrichtenübermittlung. Der Vorteil des Mini-Slotting-Protokolls im Vergleich zum CAN-Protokoll ist die höhere mögliche Bandbreite, da beim CAN-Protokoll die Bandbreite durch die Signallaufzeit der Bits begrenzt wird.

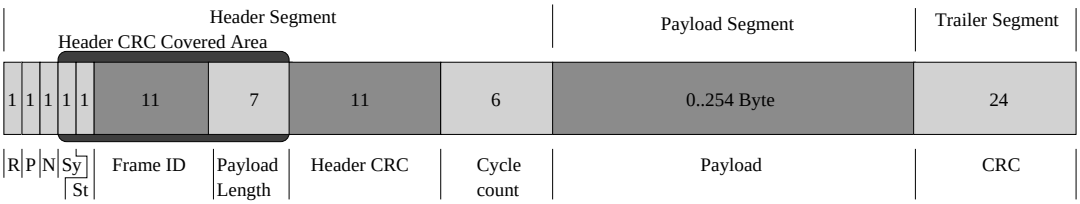


Abbildung 2.6: FlexRay-Rahmen nach [Fle05]

Die Abbildung 2.6 zeigt einen FlexRay-Rahmen. Auf den ersten Blick fällt auf das dieser Rahmen komplexer als das Nachrichtenformat im TTP/C-Protokoll ist. Der FlexRay-Rahmen setzt sich aus drei Segmenten zusammen: dem 5 Byte langen Header, der Nutzlast und dem daran angehängten 24 Bit CRC-Segment.

Der **Header** beginnt mit 5 Bitwerten. Das erste Bit (R-Bit, *reserved bit*) ist für zukünftige Veränderungen am Protokoll freigehalten und sollte daher auf Null gesetzt werden. Das P-Bit (*payload preamble indicator*) gibt an, ob innerhalb der Nutzlast ein optionaler Vektor, der z.B. eine Nachrichten-ID enthält oder dem Netzmanagement dient, enthalten ist. Das folgende N-Bit ist das *null frame indicator* Bit und definiert, ob es sich um einen Rahmen ohne Nutzlast handelt. Um anzuzeigen, dass ein Rahmen für die Synchronisation verwendet wird, ist das

Sy-Bit (*sync frame indicator*) im Header enthalten. Ist das St-Bit (*startup frame indicator*) gesetzt, handelt es sich bei dem Rahmen um einen Initialisierungsrahmen. Den 5 Bits folgt eine 11 Bit lange *Frame-ID*. Die Frame-ID bestimmt in welchem Zeitschlitz die Nachricht übertragen wird und kann daher nur einmal pro Kommunikationszyklus in einem Kanal verwendet werden. Das sich anschließende 7 Bit Längenfeld der Nutzlast (*payload length*) gibt an, wie viele Bytes Nutzlast in der Nachricht enthalten sind. Dabei wird die tatsächliche Nutzlastgröße in Byte im Nutzlastlängenfeld durch 2 dividiert gespeichert. FlexRay hat im Gegensatz zu den bisher vorgestellten Protokollen eine extra Prüfsumme für den Header. Dieses CRC-Feld ist 11 Bit lang und sichert, wie in Abbildung 2.6 dargestellt, die Felder Sy-Bit, St-Bit, Frame-ID und das Längenfeld der Nutzlast. Die zusätzliche CRC-Berechnung für den Header dient vor allem dazu das Sy-Bit und das St-Bit gegen einen fehlerhaften Kommunikationscontroller zu sichern. Der letzte Teil des Headers enthält den Wert des Zählers (*cycle counter*), welcher die Nummer des Kommunikationszyklus zum aktuellen Zeitpunkt angibt.

Das **Nutzlastsegment** fasst zwischen 0 und 254 Byte Daten. Durch das *payload length* Feld können nur gerade Bytezahlen in der Nutzlast übermittelt werden. Die Nutzlast enthält daher zwischen 0 und 127 zwei-Byte-Wörter. In der Nutzlast einer im statischen Segment zu sendenden Nachricht kann über das Setzen des *payload preamble indicator* Bits (P-Bit) ein bis zu 12 Byte größer Netzmanagementvektor eingefügt werden. Für Nachrichten, welche im dynamischen Segment gesendet werden, kann mit Hilfe des P-Bits ein 2 Byte großer Nachrichtenidentifier in die Nutzlast eingefügt werden. Über diesen Identifier kann der Empfänger die Nachricht besser filtern oder entsprechend weiterleiten. Das 24 Bit **CRC-Feld** schließt den Rahmen einer FlexRay Nachricht ab. Diese Prüfsumme beinhaltet die Nutzlast und alle Felder des Headers, einschließlich des Header CRC-Feldes.

Da FlexRay die Zuweisung der statischen Zeitschlitze und das dynamische Segment zeitgesteuert plant, muss eine Uhrensynchronisation vorgenommen werden. Dabei ist im Protokoll keine globale Referenzzeit vorgesehen, vielmehr hat jeder Knoten eine lokale Sicht auf die Zeit. Diese Sicht muss aber bei allen Knoten mit einer gewissen Genauigkeit einer gemeinsamen globalen Zeit entsprechen. Dazu zeichnet jeder Knoten ähnlich wie bei TTP/C die Differenzzeit vom tatsächlichen und dem erwarteten Eintreffen der Nachrichten mit gesetztem Sy-Bit im statischen Segment auf. Daraus lässt sich ein Korrekturterm berechnen und die Uhren der einzelnen Knoten können synchronisiert werden. Im Gegensatz zu TTP/C bietet FlexRay keinen Membership-Dienst an. Erfordert eine Anwendung eine konsistente Sicht auf den Status der Clustermitglieder und den Ausschluss fehlerhafter Knoten von der Kommunikation, muss dies in der Applikation umgesetzt werden.

FlexRay ermöglicht zeit- und ereignisgesteuerte Kommunikation und kann so Eigenschaften beider Ansätze in einem Protokoll vereinen. Periodische Nachrichten werden zeitgesteuert, mit festen Sendezeiten und geringem Jitter, im statischen Segment übermittelt. Im dynamischen Segment können mit FlexRay nicht-periodische Nachrichten prioritätsbasiert gesendet werden. Dadurch ist das Protokoll sehr flexibel und für dynamisches Nachrichtenaufkommen effektiv einsetzbar. FlexRay erreicht laut der Spezifikation Bandbreiten von bis zu 10 MBit/s. Die Fehlertoleranz bei FlexRay ist skalierbar, so können z.B. redundante Kommunikationskanäle, Bus-Guardians und verschiedene Netztopologien verwendet werden. Damit lassen sich mit FlexRay sicherheitskritische Anwendungen ebenso wie unkritische Anwendungen mit Best-Effort Kommunikation realisieren. Die Spezifikation [Fle05] und die Quellen [ZS06, Rus03]

geben eine ausführliche Beschreibung von FlexRay und eine Einordnung in andere Bussysteme für sicherheitskritische Systeme.

2.3.7 TTCAN

TTCAN ist eine zeitgesteuerte Erweiterung des CAN-Protokolls (siehe Unterabschnitt 2.3.1). Das Nachrichtenformat ist mit dem des CAN-Protokolls identisch, der Buszugriff wird in TTCAN jedoch zeitgesteuert umgesetzt. TTCAN ist daher empfangskompatibel mit CAN, d.h. ein CAN-Knoten kann TTCAN-Nachrichten korrekt empfangen, darf aber keine Nachrichten im TTCAN-Netz senden. Mit TTCAN wird ein vorhersehbares Kommunikationsverhalten erreicht, so dass alle sicherheitskritischen Nachrichten garantiert übermittelt werden, auch wenn der Bus voll ausgelastet ist. Die Einführung einer zeitgesteuerten Kommunikation in das CAN Protokoll, erfordert eine globale Zeit. Auf deren Basis lassen sich dann die Buszugriffe abstimmen. Die globale Zeit wird von einem Master-Knoten durch eine Referenznachricht vorgegeben. Diese Nachricht bestimmt den Beginn einer Kommunikationsrunde. Eine Referenznachricht wird durch einen speziellen Identifier erkannt. In TTCAN werden zwei Referenznachrichten unterschieden, die erste enthält nur Steuerinformationen, während die andere zusätzlich dazu weitere Informationen wie die Zeit des Master-Knotens beinhaltet. Alle Knoten speichern beim Empfang einer Referenznachricht, welche die globale Zeit[8] enthält, ihren lokale Zeit ab. Mit der Differenz zwischen der lokalen und globalen Zeit kann die Synchronisation im Knoten vorgenommen werden.

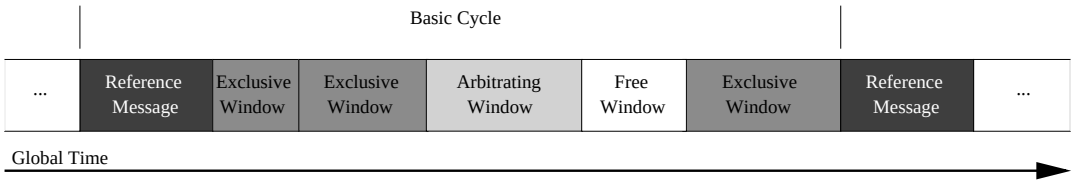


Abbildung 2.7: TTCAN-Basiszyklus nach [FMD+00]

Die Zeitspanne zwischen zwei Referenznachrichten wird in TTCAN als Basiszyklus (*basic cycle*) bezeichnet. Ein solcher Basiszyklus ist in Abbildung 2.7 dargestellt. Innerhalb dieses Zyklus wird der Buszugriff über TDMA realisiert. In einem Zyklus sind dabei mehrere Zeitschlitze unterschiedlicher Größe enthalten. Mit TTCAN können periodische wie auch nicht-periodische Nachrichten gesendet werden. Ein Zeitschlitz für periodische Nachrichten wird als **exklusiver Zeitschlitz** (*exclusive time window*) bezeichnet. Der Anfang dieses Zeitschlitzes definiert die Sendezeit der zugehörigen Nachricht. Die Definition der Zeitschlitze und der zu den exklusiven Zeitschlitzen gehörenden Nachrichten ist vor der Inbetriebnahme fest zu definieren. Es kommt daher zu keinen Medienzugriffskonflikten in den exklusiven Zeitschlitzen. Damit die Vorhersagbarkeit und der kleine Kommunikationsjitter der periodischen Nachrichten in den exklusiven Zeitschlitzen durchgesetzt werden kann, ist eine Neuübermittlung der Nachrichten im Fehlerfall nicht erlaubt.

Das CAN-Protokoll zeichnet sich besonders durch seine kollisionsauflösende Arbitrierung und die prioritätsbasierte Übermittlung von nicht-periodischen Nachrichten aus. Um diese Mechanismen im TTCAN-Protokoll weiter zu nutzen, werden **arbitrierbare Zeitschlitze** (*arbi-*

[8] Die Zeit des Master-Knotens wird als global korrekte Zeit definiert.

trating time window), siehe Abbildung 2.7, in den Basiszyklus eingefügt. Innerhalb eines solchen Zeitschlitzes erfolgt der Buszugriff wie beim CAN-Protokoll über die bitweise Arbitrierung. Bei der Erstellung des Kommunikationsplans, muss der genaue Inhalt eines arbitrierbaren Zeitschlitzes nicht bekannt sein, da die Arbitrierung innerhalb des Zeitschlitzes die Übermittlung der wichtigsten Nachrichten sicherstellt. So ist es möglich mehrere Nachrichten, ggf. auch mehr Nachrichten als der Zeitschlitz auf einmal aufnehmen kann, einem arbitrierbaren Zeitschlitz zuzuordnen. Es kann dabei im Voraus nicht bestimmt werden, ob und wann eine Nachricht gesendet wird. Auch in diesen Zeitschlitzen ist die automatische Neuübermittlung im Fehlerfall nicht erlaubt, wenn sie nicht innerhalb des aktuellen Zeitfensters abgeschlossen werden kann. Wie in der Abbildung 2.7 zu erkennen ist, ist es in TTCAN auch möglich innerhalb des Basiszyklus Zeitschlitze für spätere Nutzung (*free time window*) zu reservieren.

Die Knoten im TTCAN-Netz enthalten im Gegensatz zu anderen zeitgesteuerten Kommunikationsprotokollen nur das Wissen über die Nachrichten, welche sie empfangen und senden. Dies hat zwei Vorteile: Zum einen spart dieser Ansatz in den Knoten Speicher und zum anderen müssen bei einer Änderung des Kommunikationsplans nur die betreffenden (also empfangenden oder sendenden) Knoten ihren Kommunikationsplan anpassen. Der einfache Basiszyklus bietet für Nachrichten mit unterschiedlichen Perioden nicht genügend Flexibilität. TTCAN erlaubt es daher mehrere Basiszyklen in einer Systemmatrix zusammenzufügen. Damit ist es möglich periodische Nachrichten mit unterschiedlichen Perioden optimal in den Kommunikationsplan einzubinden.

Der Master-Knoten ist als Zeitgeber und zur Definition des Beginns der Basiszyklen ein ausfallkritische Komponente (*single point of failure*). Es muss daher, z.B. durch redundante Master-Knoten, eine Ausfalltoleranz des Master-Knotens zugesichert werden. Weiterhin ist es in TTCAN nicht vorgesehen, dass Bus-Guardians den Zugriff auf die Zeitschlitze überwachen. Dadurch kann ein fehlerhafter Knoten, welcher Nachrichten mit hoher Priorität sendet, die exklusiven Zeitschlitze belegen oder sogar falsche Referenznachrichten senden.

TTCAN ermöglicht eine zeitgesteuerte Kommunikation auf dem CAN-Bus. Daher ist TTCAN besser für verteilte Systeme, welche deterministisches Kommunikationsverhalten erfordern, geeignet als das CAN-Protokoll. Durch die Integration der arbitrierbaren Zeitschlitze in die statische Kommunikationsstruktur, wird die Flexibilität der ereignisgesteuerten Kommunikation erreicht. Damit können Anwendungen neben periodischen Nachrichten auch nichtperiodische Nachrichten effektiv einsetzen. Ein weiterer Vorteil von TTCAN ist es, dass die bereits bekannten CAN-basierten Analysetools genutzt werden können und Entwickler wie Anwender, die sich mit dem CAN-Protokoll bereits auskennen, sich schnell in die TTCAN-Erweiterung einarbeiten können. TTCAN genügt aufgrund der geringen Datenrate und der eingeschränkten Mechanismen zur Fehlertoleranz nicht allen Anforderungen an sicherheitskritische verteilte Anwendungen. TTCAN ist deshalb für Systeme ausgelegt, die keine hohen Anforderungen an die Fehlertoleranz stellen aber kleine Jitter und festgelegte Kommunikationsverzögerungen benötigen. Es lassen sich daher mit TTCAN Systeme wie z.B. x-by-wire Systeme mit mechanischen oder hydraulischen Rückfallsystemen realisieren. Die TTCAN-Protokollerweiterung für CAN wird in [FMD+00] beschrieben. Weiterführende Beschreibungen sind in [ZS06, HMFH02, MFH+02, HMFH00] zu finden.

2.4 Gateway

Ein **Gateway** ist eine netzverbindende Komponente, welche Nachrichten zwischen verschiedenen Netzen weiterleitet. Ein **Router** ist ein spezieller Gateway zur Nachrichtenvermittlung. Begrifflich wird ein Gateway vom Router durch die zusätzliche Protokollumsetzung von Nachrichten abgrenzt. Ein Router hat nur die Aufgabe der Weiterleitung von Nachrichten, dabei finden die Nachrichten mit Hilfe von Routing-Protokollen ihr Ziel über die Grenzen von verschiedenen Netzwerken hinweg. Ein Router verändert den Inhalt der Nachrichten die er vermittelt nicht, ausgenommen sind Änderungen von Steuerinformationen (wie z.B. *hop count*). Damit ist ein Router im ISO/OSI-Referenzmodell [ISOb] in der Vermittlungsschicht (Schicht 3) einzuordnen. Ein Gateway kann je nach Art verschiedenen Schichten zugeordnet werden. Ein *Transport-Gateway* ist auf Transportschicht (Schicht 4) angesiedelt und verbindet Knoten mit unterschiedlichen Transportprotokollen. Mit Hilfe eines Transport-Gateways kann z.B. ein TCP/IP-Knoten mit einem CAN-Knoten Nachrichten austauschen. Ein *Application-Layer-Gateway* (ALG, Schicht 7 Gateway) interpretiert die Nutzlast der Nachrichten auf Anwendungsebene und formt dort die Kommunikation entsprechend um. Ein ALG kann z.B. E-Mails in SMS-Nachrichten umwandeln und sie aus dem Internet zu einem mobilen Telefon weiterleiten.

Netzwerke, die durch Gateways verbunden sind, können unterschiedliche Adressierungsmechanismen verwenden, so dass eine direkte Adressierung über die Netzgrenze hinaus nicht möglich ist. Wird, wie im vorherigen Beispiel angedeutet, eine E-Mail aus dem Internet in eine SMS umgewandelt und zu einem Telefon gesendet, kann der sendende Knoten den Empfänger nicht direkt adressieren. Der Sender nutzt Internetadressen zur Adressierung und dem Empfänger ist eine Telefonnummer zugeordnet. Der Sender kann die Telefonnummer des Empfängers somit nicht in das Empfängeradressfeld des Vermittlungsprotokolls eintragen, da dort nur Internetadressen erlaubt sind. Die Zieladresse muss so gewählt werden, dass ein Gateway die Nachricht empfängt, umwandelt und korrekt weiterleitet. Ein Gateway muss für weiterzuleitende Nachrichten die Empfängeradresse bestimmen, damit die Nachricht zum richtigen Empfänger gesendet wird. Diese Adresszuordnung (*adress mapping*) kann auf verschiedene Weise realisiert werden: die Empfängeradresse ist direkt in der Nutzlast enthalten, es gibt eine statische Zuordnung der Nachricht zu der Empfängeradresse (z.B. durch Berechnung) oder diese Zuordnung wird dynamisch vom Gateway selbst getroffen.

Ein Gateway führt im Allgemeinen[9] eine Protokollumsetzung der passierenden Nachrichten durch. Dazu werden z.B. die Absender- und Zieladressen der Nachrichten verändert oder die Nutzlast in einen anderen Protokollrahmen eingekapselt. Weiterhin können Gateways den Bereich der erreichbaren Informationen aus einem anderen Netz eingrenzen (*scope*). So können z.B. nur bestimmte Knoten aus dem einen Netz mit Knoten aus dem anderen mit vom Gateway freigeschalteten Diensten kommunizieren. Die Filterung der Information geht dabei je nach Art des Gateways bis auf die Applikationsschicht (Schicht 7). Damit kann vom Gateway entschieden werden, welche Informationen zwischen den Netzen ausgetauscht werden können. Ein Gateway ist damit eine Form einer Firewall.

In eingebetteten Systemen weist ein Gateway die gleichen Eigenschaften auf. Es verbindet zwei Netze und kann dabei lokale Nachrichten vor dem anderen Netz verstecken, denn oft ist nur ein kleiner Teil der Nachrichten aus einem Netz auch für das andere interessant. Des-

[9] Sofern es in Schicht 4 oder höher einzuordnen ist und das Quell- und Zielprotokoll sich unterscheidet.

halb passieren nur Informationen das Gateway, welche für die Weiterleitung freigegeben sind. Dies verhindert eine Überlastung und die dadurch mögliche Blockierung eines Netzes durch Nachrichten aus dem anderen Netz. Weiterhin wird in [Kop97] beschrieben, dass mit Gateways durch die Kapselung der internen Kommunikation Altsysteme (*legacy systems*) transparent in ein verteiltes System eingebunden werden können.

Im Folgenden sind die Dienste aufgezählt, welche ein Gateway den Knoten in den verbundenen Netzen anbieten kann:

- **Netzübergreifende Adressierung:**
  Die netzübergreifende Adressierung, ermöglicht es Knoten aus dem einen Netz die Knoten aus dem anderen Netz zu adressieren und erlaubt so eine direkte Kommunikation zwischen den Netzen.

- **Filterung der Nachrichten:**
  Durch die Filterung kann bestimmt werden welche Information das andere Netz erreichen kann. Dabei erlaubt die Filterung der Nachrichten, dass nur Nachrichten das Gateway passieren, die im Zielnetz auch relevant sind. Das Gateway kann dafür entweder einen statischen Filter nutzen, der angibt welche Nachrichten es weiterleitet oder einen dynamischen Filter, welcher zur Laufzeit konfiguriert werden kann. Mit einem dynamischen Filter kann z.B. das Gateway die Weiterleitung einer Nachricht nur dann erlauben, wenn ein Knoten aus dem anderen Netz diese anfordert. ALGs können die Filterung bis zur Applikationsschicht (Schicht 7) anwenden und die Nachrichten anhand der Anwendungsdaten filtern.

- **Steuerung des Informationsflusses:**
  Das Gateway kann den Informationsfluss zwischen den Netzen direkt steuern. Es kann z.B. in prioritätsbasierten Netzen den eingeleiteten Nachrichten Prioritäten zuweisen. Damit werden die Nachrichten in die Struktur des lokalen Netzes eingeordnet. Dies ist notwendig, wenn zwei prioritätsbasierte Netze verbunden werden sollen und sie unterschiedliche Prioritäten für die gleiche Information nutzen. Es ist einem Gateway auch möglich den Nachrichtentyp zu ändern. So können z.B. aperiodische Ereignisnachrichten in periodische Statusnachrichten umgewandelt werden. Dazu muss das Gateway die Nutzlast jeder Ereignisnachricht interpretieren, um den Statuswert zu berechnen, und diesen Statuswert periodisch in das Zielnetz übertragen.

- **Pufferung von ausgehenden Informationen:**
  Die Pufferung von Nachrichten ermöglicht es dem Gateway Nachrichten entgegen zu nehmen, obwohl sie aufgrund einer Überlastsituation nicht in das Zielnetz geleitet werden können. Ist im Zielnetz wieder Bandbreite verfügbar, werden die zwischengespeicherten Nachrichten versendet. Das Gateway kann die Puffer so organisieren, dass die ausgehenden Nachrichten z.B. nach der Eingangsreihenfolge oder prioritätsbasiert sortiert sind. Hält die Überlastsituation lange an oder sollen viele Nachrichten das Gateway passieren, kann der Nachrichtenpuffer überlaufen und Nachrichten müssen verworfen werden.

- **Entkopplung des Kontrollflusses:**
  Die Entkopplung des Kontrollflusses ist besonders bei zeitgesteuerten Netzen wichtig, da sich Kontrollfehler wie z.B. zeitliche Fehler zwischen den Netzen nicht ausbreiten dürfen. Deshalb werden bei der Kopplung zweier Netze beide Netze von zwei unabhängigen

Kommunikationscontrollern im Gateway verbunden. Beide Kommunikationscontroller des Gateways sind auch vom Applikationsrechner des Gateways entkoppelt (vgl. Abbildung 2.4). Bei dieser Anordnung der Komponenten gibt es im Gateway keinen Kontrollfluss zwischen den Kommunikationscontrollern und dem Applikationsrechner. Damit kann es beim Fehlverhalten eines Kommunikationscontrollers des Gateways zu keinen zeitlichen Fehlern im anderen Netz kommen. Weiterhin werden zeitliche Sendefehler von Knoten aus dem einen Netz durch die Entkopplung des Kontrollflusses nicht in das andere propagiert. Damit trägt das Gateway zur Fehlerbegrenzung bei.

Für weiterführende Informationen zu Gateways werden die Quellen [Tan03, Tv03] genannt.

2.5 Middleware

Die *Middleware* stellt eine Abstraktion dar, welche heterogene Architekturen, wie Netzwerke, Betriebssysteme und Hardware, vor der Anwendung verbirgt. Dazu wird die Middleware zwischen der Anwendung und dem (Netzwerk-)Betriebssystem [CDK02] positioniert. Durch die Middleware kann die Zugriffstransparenz für eine verteilte Anwendung erreicht werden, so dass die Anwendung keine Informationen darüber benötigt, ob eine Funktion, ein Objekt oder eine Komponente lokal oder über das Netzwerk zu erreichen ist. Die Schnittstellen der Middleware zur Anwendung und zum Betriebssystem sind dabei klar definiert. Im ISO/OSI-Referenzmodell kann nach [Ser99] die Middleware die Sitzungs-, Darstellungs- und Anwendungsschicht (Schichten 5 bis 7) umfassen. Diese Einordnung definiert eine Kommunikation innerhalb der verteilten Anwendung auf Basis der Anwendungsschnittstelle der Middleware. Damit wird die Anwendung vom eigentlichen Betriebs- und Kommunikationssystem entkoppelt. Die Middleware bietet der Anwendung dazu transparente Kommunikationsfunktionen und je nach Implementierung weitere Dienste zur Namensgebung, Persistenz, verteilten Transaktion, Ereignisbenachrichtigung und Sicherheit an.

Für die Realisierung einer Middleware gibt es unterschiedliche Modelle. Ein einfaches Modell ist es alle Ressourcen als **Datei** zu behandeln. Bei diesem Modell macht es keinen Unterschied, ob die Datei lokal oder entfernt zu finden ist. Die Anwendung verwendet nur Lese- und Schreibzugriffe auf Dateien und die Middleware setzt die Anfragen in lokale oder entfernte Ressourcenzugriffe um. Ein anderes Middleware-Modell verwendet die Abstraktion von **Funktionen**. Hierbei verbirgt die Middleware die Kommunikation dadurch, dass die Anwendung transparent auf Funktionen zugreift. Dabei ist es unabhängig, wo sich die Funktion tatsächlich befindet. Ein Beispiel hierfür ist *RPC* (Remote Procedure Call).

Durch die Dominanz der objektorientierten Softwareentwicklung wurde neben der funktionsbasierten Middleware die **objektorientierte Middleware** eingeführt. In diesem Modell wird statt des transparenten Funktionsaufrufes ein Zugriff auf verteilte Objekte realisiert. Der Anwendung ist dabei nicht bekannt, wo die Objekte zu finden sind, denn die Lokalisierung übernimmt die Middleware. Die Anwendung kennt nur die Schnittstelle für die verteilten Objekte und kann auf die darin definierten Methoden transparent zugreifen. Oft ist jedes verteilte Objekt auf nur einem Knoten abgelegt und kann von mehreren anderen Knoten erreicht werden. Beispiele für dieses Konzept sind z.B. CORBA und Jini, welche in den folgenden Unterabschnitten kurz dargestellt werden.

Eine Middleware kann auch auf Nachrichten oder Ereignissen basieren. Auch in diesen Konzepten wird durch die Middleware die direkte Kommunikation vor der Anwendung verborgen. So kann die Anwendung ohne den Sender oder Empfänger zu kennen an der Kommunikation teilnehmen. Damit greift die Anwendung transparent auf Informationen in Form von Nachrichten oder Ereignissen zu. Dieser Zugriff kann z.B. knoten-, betreff- oder inhaltsbasiert realisiert sein. Die Publish/Subscribe-Middlewarearchitekturen gehören zu der Klasse der nachrichten- bzw. ereignisorientierten Middleware. Ein Beispiel für eine **Publish/Subscribe-Middleware** ist COSMIC, welches in einem folgenden Unterabschnitt beschrieben ist.

Weiterführende Betrachtungen zur Middleware sind in [CDK02, Ser99, Tv03] zu finden. Im Folgenden werden verschiedene Middleware-Konzepte und Umsetzungen kurz dargestellt.

2.5.1 CORBA

CORBA (*Common Object Request Broker Architecture*) ist eine objektorientierte Spezifikation für eine Middleware verteilter Systeme. Die OMG (*Object Management Group*) hat CORBA Anfang der 1990er Jahre entwickelt und betreut auch die Weiterentwicklung dieser Spezifikation. CORBA ist dafür ausgelegt die Entwicklung von wiederverwendbaren verteilten Anwendungen zu unterstützen. Dafür werden die Schnittstellen für die Anwendung und die Implementierungen voneinander getrennt sowie verschiedene Dienste für die verteilten Objekte angeboten. Im Zentrum der CORBA-Architektur steht der **ORB** (*Object Request Broker*). Er ermöglicht die Kommunikation in der verteilten Anwendung und verbirgt die Verteilung und die Heterogenität des Systems.

Neben dem ORB beschreibt CORBA **Applikations-Objekte**, **CORBA-Facilities** (vertikale und horizontale Facilities) und allgemeine **Objektdienste**. Die Applikations-Objekte entsprechen den verteilten Objekten in der Anwendung. Die horizontalen Facilities bestehen aus anwendungsunabhängigen hochwertigen (High-Level) Diensten, wie z.B. Dienste für Benutzeroberflächen oder zur Verwaltung von Informationen. Die vertikalen Facilities setzen anwendungsspezifische hochwertige Dienste um. Die Dienste sind für jeden Anwendungsbereich verschieden, und können nur in dem entsprechenden Anwendungsbereich sinnvoll umgesetzt werden. Die allgemeinen Objektdienste sind in Tabelle 2.2 dargestellt.

Das **CORBA-Objektmodell** basiert auf verteilten Objekten, welche durch den ORB transparent lokalisiert werden. Die Definition der Schnittstellen der Objekte und deren Dienste wird mit der **CORBA-IDL** (*Interface Definition Language*) realisiert. Die IDL beschreibt dabei nur die Methoden, welche von den entsprechenden verteilten Objekten angeboten werden. Wie in der Abbildung 2.8 zu erkennen ist, werden die Knoten in Clients und Server unterteilt. Clients führen Applikationen aus, welche durch die in der IDL definierten Schnittstellen transparent auf verteilte Objekte zugreifen. Server stellen die Implementierungen der verteilten Objekte für die Clients bereit. Die Objektimplementierungen werden im Server in einer entsprechenden Programmiersprache wie z.B. C++, Java oder Smalltalk realisiert.

Client und Server besitzen jeweils eine ORB-Komponente, welche die Kommunikation zwischen ihnen transparent übernimmt. Aus Sicht des Clients und des Servers bietet der ORB nur die allgemeinen Dienste (siehe Tabelle 2.2) zur Handhabung von Objekten an. Alle anderen Funktionalitäten des ORB werden für Client und Server nicht sichtbar durchgeführt. Die Client-Applikation hält **Proxies** für die Verarbeitung von Methodenaufrufen, welche dieselben IDL-Schnittstellen besitzen wie die dazugehörigen implementierten Objekte auf dem Server.

Dienst	Beschreibung
Abfrage (*Query*)	Abfrage von Objektmengen
Auflistung (*Collection*)	Gruppierung von Objekten
Benachrichtigung (*Notification*)	Ereignisbasierte asynchrone Kommunikation
Beziehungen (*Relationship*)	Beschreibung von Beziehungen zw. Objekten
Eigenschaften (*Property*)	Zuordnung von Eigenschaften zu Objekten
Ereignis (*Event*)	Asynchrone Kommunikation durch Ereignisse
Handel (*Trading*)	Veröffentlichung und Suche von Objekt-Diensten
Lebenszyklus (*Life Cycle*)	Erzeugen, Zerstören, Kopieren von Objekten
Lizenzierung (*Licensing*)	Zuordnung von Objekten zu Lizenzen
Namensgebung (*Naming*)	Systemübergreifende Namensgebung von Objekten
Nebenläufigkeit (*Concurrency*)	Gleichzeitiger Zugriff auf Objekte
Persistenz (*Persistence*)	Persistentes Speichern von Objekten
Serialisierung (*Externalization*)	Ver- und Entpacken von Objekten
Sicherheit (*Security*)	Sichere Übertragung und Autorisierung
Transaktion (*Transaction*)	Transaktionen für Methodenaufrufe
Zeit (*Time*)	Bereitstellung der aktuellen Zeit

Tabelle 2.2: CORBA-Dienste nach [Tv03]

Der Proxy ist ein Stub der die Aufrufanforderung der Applikation entgegennimmt und über den ORB an den Server übermittelt. Der Server führt die Methode mit den vom Client spezifizierten Parametern über den **Skeleton** aus und sendet die Ergebnisse an den Client zurück. Die Implementierungen der Proxies und Skeletons können in unterschiedlichen Wirtssprachen vorliegen, da aus der IDL-Schnittstelle mit dem IDL-Compiler der Skeleton in der Wirtssprache des Servers und der Proxy in der Wirtssprache des Clients erzeugt werden. Die Objektimplementierung und die Client Applikation können dann in unterschiedlichen Programmiersprachen realisiert werden. Die Proxies und Skeletons werden zur Compilezeit erzeugt und liegen statisch vor.

Der **Objektadapter** (siehe Abbildung 2.8) leitet die Anforderungen der Clients an die jeweilige Objektimplementierung weiter. Er hat damit die Aufgabe die Aktivierung der Objekte im Server zu steuern. Weiterhin verbirgt er die eigentlichen Objekte vor dem Client. Der Objektadapter ist eine generische konfigurierbare Komponente, der eine bestimmte Aktivierungsstrategie verfolgt. Ein Objektadapter kann die Aktivierung von ein oder mehreren Objekten verwalten.

CORBA unterstützt Anwendungen bei denen ein Client auf zur Compilezeit nicht bekannte Objekte zugreift. Dem Client steht in diesem Fall keine statisch definierte Schnittstelle für einen Methodenaufruf zur Verfügung. Diese Schnittstelle muss erst zur Laufzeit erzeugt werden. Dazu gibt es in CORBA clientseitig die **dynamische Aufrufschnittstelle** (DII, *Dynamic Invocation Interface*), welche den entfernten Methodenaufruf ohne einen zugehörigen statischen Proxy ermöglicht. So können zur Laufzeit hinzugefügte Objekte durch eine Client-Applikation genutzt werden. Werden in einen Server Objekte zur Laufzeit hinzugefügt, besitzt er kein statisches Skeleton, welches den Aufruf eines Clients annehmen kann. Die **dynamische Skeleton-Schnittstelle** (*Dynamic Skeleton Interface*) erlaubt es Servern Methodenaufrufe anzunehmen

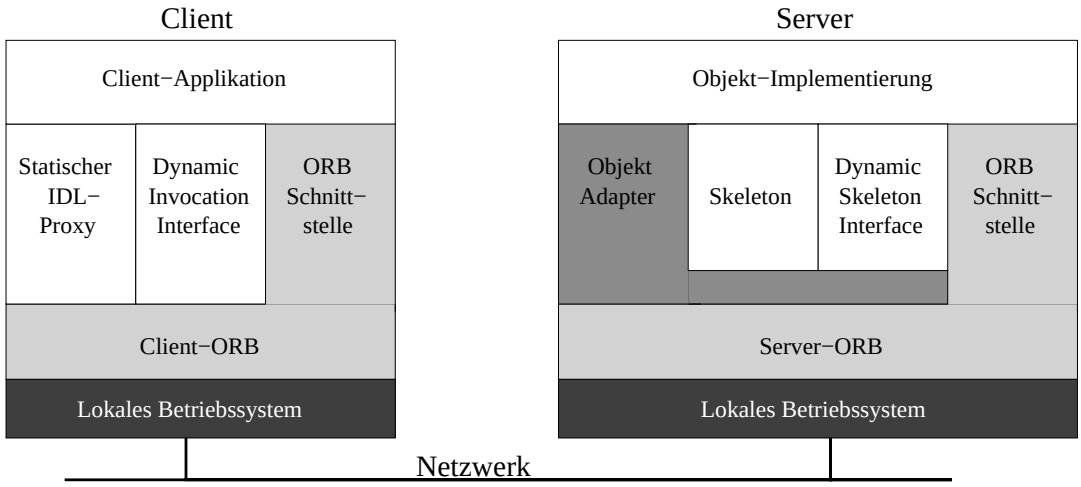


Abbildung 2.8: CORBA-System nach [Tv03]

und auszuführen für die sie kein Skeleton besitzen, da das zugehörige Objekt zur Compilezeit noch nicht bekannt war. Damit können in ein verteiltes CORBA-System zur Laufzeit neue Objekte eingefügt und von der Anwendung genutzt werden.

Neben diesen Komponenten umfasst CORBA ein Implementierungs- und ein Schnittstellen-Repository. Das **Implementierungs-Repository** enthält die Namen von Objektadaptern und die dazugehörigen Pfadnamen der Objektimplementierungen. Es ermöglicht so die Lokalisierung von verteilten Objekten. Dazu werden die Objektimplementierungen vom Server beim Implementierungs-Repository registriert. Neben der Objektadapternamen und dem Pfad der Objektimplementierung, wird dabei auch der Name und der dazugehörige Port des registrierenden Servers hinterlegt. So kann im verteilten System die Objektimplementierung zu einem Objektadapternamen gefunden werden, ohne dass der aufrufende Client den Server kennt, der das aufgerufene Objekt hält. Das **Schnittstellen-Repository** stellt Clients wie auch Servern auf Anfrage Informationen über registrierte IDL-Schnittstellen zur Verfügung. Ein Client kann damit für ein ihm unbekanntes CORBA-Objekt Methoden und deren Parameter erhalten und es verwenden. Das Schnittstellen-Repository wird nur für Methodenaufrufe von Objekten benötigt, für die der Client keinen statischen Proxy besitzt. Für weiterführende Beschreibungen des CORBA-Referenzmodell wird auf folgende Literatur verwiesen [Tv03, CDK02, Ser99, Pop98].

2.5.2 Jini

Jini von Sun Microsystems ist eine koordinationsbasierte objektorientierte Middleware. Jini basiert auf Java. Die Prinzipien von Jini sind aber auch mit anderen Programmiersprachen umsetzbar. In Jini werden die Kommunikationssysteme **temporär** und **referentiell** entkoppelt. Die referentielle Entkopplung charakterisiert das Fehlen von direkten Referenzen in der Kommunikation. Die Kommunikationspartner greifen dabei z.B. themenorientiert auf die Informationen zu und bleiben anonym. Die zeitliche Entkopplung erlaubt, dass Sender und Empfänger bei einer Kommunikation nicht gleichzeitig aktiv sind. Ein Sender kann somit eine Information zu einem beliebigen Zeitpunkt bereitstellen und der Empfänger kann sie dann unabhängig von der Bereitstellungszeit lesen. Dafür wird ein gemeinsamer von allen Kommunikationsteilneh-

mern genutzter Datenraum benötigt. Das von Jini genutzte Koordinationsmodell mit referentiell und temporär entkoppelter Kommunikation wird in [Tv03] als *generative Kommunikation* bezeichnet. Bei diesem Modell kommunizieren unabhängige Prozesse über einen gemeinsam genutzten Datenraum aus Tupeln. Ein Tupel ist Datensatz mit null oder mehr typisierten Feldern.

Die zentrale Komponente in Jini sind die **JavaSpaces**. Ein JavaSpace ist der gemeinsam genutzte Datenraum in dem Referenzen auf Java-Objekte als serialisierte (verpackte) Tupel abgelegt sind. Es werden drei Funktionen für Tupel in dem JavaSpace angeboten: `write`, `read` und `take`. Mit `write` wird eine Tupel-Instanz in ein JavaSpace abgelegt, wobei es durch die Serialisierung um eine Kopie handelt. Die Funktionen `read` und `take` greifen lesend auf die Tupel-Instanzen zu. Sie unterscheiden sich dadurch, dass `take` vernichtend liest. Das Lesen von Tupeln aus dem JavaSpace wird mit Schablonen realisiert. Dabei erstellt der Empfänger ein Tupel für das Tupel, welches gelesen werden soll. Dieses Tupel wird als Schablone für den Vergleich der Tupel im JavaSpace verwendet. Sind Schablone und Tupel-Instanz vom selben Typ, kann das Tupel gelesen werden. Dazu wird die Instanz entpackt (*deserialisiert*) und an den Empfänger weitergegeben. JavaSpaces erlauben neben dem Lesen von Tupeln auch die Benachrichtigung des Clients, wenn ein passendes Tupel in den JavaSpace abgelegt wird. Die Tupel-Instanzen im JavaSpace haben eine definierte **Gültigkeitsdauer** (Lease), ist diese Dauer abgelaufen wird die Instanz aus dem JavaSpace entfernt. Dies verdeutlicht die temporale Entkopplung, denn Sender und Empfänger einer Tupel-Instanz müssen nicht gleichzeitig aktiv sein. Es ist sogar möglich JavaSpaces als persistenten Speicher umzusetzen.

Die Architektur von Jini besteht aus drei Schichten. Die **unterste Schicht** bildet die Jini-Infrastruktur, welche z.B. die Kommunikation und den Suchdienst (*Lookup Service*) zur Verfügung stellt. Die Kommunikation in Jini wird komplett durch **Java RMI** (*Remote Method Invocation*) realisiert. Mit Hilfe des **Suchdienstes** können Clients Dienste (Objekte, Komponenten) suchen. Dazu werden ähnlich wie beim Lesen in JavaSpaces Schablonen erstellt, welche dann mit den bereitstehenden Diensten verglichen werden.

Der Suchdienst ermöglicht auch die Benachrichtigung von Clients, wenn ein gesuchter Dienst, der mit der Schablone übereinstimmt, hinzugefügt wird oder seine Gültigkeit verliert und nicht mehr genutzt werden kann. Prinzipiell kann der Suchdienst auch in der obersten Schicht mit den JavaSpaces implementiert werden. Jini verwendet aber einen spezialisierten Suchdienst, der im Gegensatz zu den JavaSpaces, innerhalb der untersten Schicht realisiert ist. In Jini ist es möglich mehrere für verschiedene Dienstklassen spezialisierte Suchdienste zu definieren, so kann die Überlastung eines zentralen Suchdienstes vermieden werden. Der Suchdienst wird in Jini mit Hilfe eines **Discovery-Protokolls** gefunden. Damit kann ein hinzugefügter Knoten den Suchdienst finden und sich in die Jini-Kommunikation integrieren.

Die **mittlere Schicht** der Jini-Architektur definiert Funktionen, welche die Infrastruktur der untersten Schicht erweitern. Diese Funktionen umfassen unter anderem **Ereignis- und Benachrichtigungsfunktionen**, Funktionen für das Setzen von **Gültigkeiten** und eine Schnittstelle für **Transaktionen**. Die mittlere Schicht stellt diese Funktionalitäten für die Anwendungen auf der oberen Schicht bereit.

In der **obersten Schicht** sind die Clients und benutzerdefinierte Dienste zu finden. Die Jini-Architektur definiert nicht, welche Dienste in dieser Schicht enthalten sein müssen. Es sind z.B. Dienste wie ein JavaSpace-Server oder ein Transaktionsmanager in dieser Schicht umgesetzt. Anwendungen der oberen Schicht können die Funktionen der mittleren und der unteren Schicht

nutzen. Mit Jini lassen sich so z.B. verteilte Ereignisdienste umsetzen, bei denen sich ein Client für verschiedene Ereignisse interessiert und benachrichtigt wird, wenn sie eintreten.

Durch die Kopplung mit Java ist Jini nur auf javabasierten verteilten Systemen einsetzbar. Jini verfolgt mit seinen Kernkonzepten (automatische Eingliederung in die Kommunikation, Suchdienst, Gültigkeitsdauer, entfernte Ereignisse und Transaktionen) einen Ansatz, der sich besonders an die verteilte Kommunikation von Client/Serversystemen für Java richtet. In [Tv03, Edw99] wird das Jini-Konzept im Detail erläutert.

2.5.3 COSMIC

COSMIC (COoperating SMart devICes) ist eine Publish/Subscribe-Middleware speziell für die Echtzeitkommunikation von intelligenten Sensoren und Aktoren in verteilten eingebetteten Systemen. In vielen Anwendungen werden Sensoren direkt von einer Komponente abgefragt und Steuerungsdaten an die Aktoren übermittelt. Mit COSMIC publizieren und abonnieren die Sensor- und Aktor-Komponenten Ereignisse eigenständig und die Interaktion aller Komponenten mit dem gesamten verteilten System wird ermöglicht. Dadurch kann ein modulares System ohne eine zentrale Koordinierungskomponente realisiert werden.

Die Kommunikation wird in COSMIC über **Ereigniskanäle** (*event channel*) realisiert. Ein Ereigniskanal besteht dabei aus einem eindeutigen Betreff (*subject*), Attributen zur Beschreibung der Eigenschaften des Kanals (wie z.B. Typ, Periodizität oder Latenz) und Handlern für Benachrichtigungen oder Fehler. In COSMIC wird die Adressierung der Informationen über den Betreff realisiert (*subject-based addressing*). Aus diesem Grund muss der Betreff eindeutig einer Information zugeordnet sein. Die Ereignisnachrichten werden mit Hilfe dieser Kanäle vom Produzent zum Konsument transportiert. Ereigniskonsumenten abonnieren (*subscribe*) dazu Ereigniskanäle und Produzenten publizieren (*publish*) die Ereignisse durch die Kanäle. Der Begriff Ereignis beschreibt in COSMIC dabei eine getypte Information. Ereignisnachrichten[10] sind typisierte Nachrichten und enthalten den Betreff, kontextbasierte Attribute, qualitätsbasierte Attribute und die Nutzdaten. Die kontextbasierten Attribute beschreiben z.B. den Ort, wo das Ereignis produziert wird, oder die Zeit des Auftretens. Das Gültigkeitsintervall (*validy interval, expiration time*), die Deadline der Ereignisnachricht und in wie weit diese zeitlichen Bedingungen verletzt werden dürfen (*tolerated ommision degree*) sind unter anderem in den qualitätsbasierten Attributen enthalten.

COSMIC unterstützt drei verschiedene Ereigniskanaltypen, um den Ansprüchen unterschiedlicher Nachrichten nachzukommen. Es sind zwei Echtzeitkanäle (harte und weiche Echtzeitkanäle) und ein Nicht-Echtzeitkanal in COSMIC definiert. **Harte Echtzeitkanäle** (*HRT, hard real-time*) bieten Garantien für den rechtzeitigen Empfang der zugehörigen Ereignisnachrichten unter definierten Fehlerbedingungen. Für sie wird ein exklusives Sendezeitfenster eingeplant, in dem die Ereignisnachrichten definitiv zugestellt werden. Ereignisse in **weichen Echtzeitkanälen** (*SRT, soft real-time*) werden über EDF (*earliest deadline first*) priorisiert und versendet. Es kann bei Überlastung des Kommunikationsmediums dazu kommen, dass Deadlines für weiche Echtzeitnachrichten verpasst werden. Die rechtzeitige Übertragung der Nachrichten in weichen Echtzeitkanälen wird daher nicht garantiert. Ereignisnachrichten, die **Nicht-Echtzeitkanälen** (*NRT, non-real-time*) zugeordnet sind, haben keine zeitlichen Anfor-

[10] Die Ereignisnachricht bezieht sich in COSMIC aufgrund der Definition des Begriffes Ereignis nicht auf die Unterscheidung von Ereignis- und Statusnachrichten.

derungen und werden somit nur übermittelt, wenn keine Nachrichten der Echtzeitkanäle bereit sind.

Ereigniskanäle werden durch den eindeutigen Betreff identifiziert. Dieser Betreff ist eine global eindeutige ID (*unique identifier*, *UID*), welche von der Adressierung im Netz unabhängig ist. Die Adressierung der Ereigniskanäle erfolgt in COSMIC mit Hilfe einer zentralen Komponente, dem **Event-Channel-Broker** (ECB). Der ECB bindet die lange global eindeutige ID eines Ereigniskanals (*subject*) an eine kurze dynamisch erzeugte Adresse zur Kommunikation (*Event-Tag*). Diese Netzadresse ist nur im jeweiligen Netz gültig. Sollen mehrere Netze miteinander verbunden werden, muss in jedem Netz eine Zuordnung von UID zu Netzadresse realisiert werden. Die Netzadresse wird in jeder Ereignisnachricht übertragen und dient der eindeutigen Zuordnung von Ereignisnachricht zum Ereigniskanal. Durch die Adressbindung im ECB wird die Unabhängigkeit der Adresse zur Kommunikation (*Event-Tag*) von der Identifikation des Ereigniskanals (UID) erreicht. Diese Trennung von UID und Netzadresse ermöglicht die globale Adressierung von Ereigniskanälen.

Jeder COSMIC-Knoten verfügt über eine Komponente, welche eingehende Ereignisnachrichten filtert und die Ereigniskanäle verwaltet. Diese Komponente wird als **Event-Channel-Handler** (*ECH*) bezeichnet und ist im Knoten zwischen der Anwendung und dem Kommunikationscontroller angesiedelt. Der ECH publiziert die Ereignisnachrichten und benachrichtigt die Anwendung, wenn neue Ereignisnachrichten eingetroffen sind. Weiterhin verwaltet diese Komponente auch die Zuordnung der Event-Tags zu den Ereigniskanälen. Stellt eine Anwendung auf einem Knoten einen Ereigniskanal bereit oder abonniert sie einen Kanal, muss der lokale ECH der Anwendung die Kommunikation ermöglichen. Der ECH benötigt für die Übermittlung oder den Empfang der zugehörigen Ereignisnachrichten das Event-Tag für diesen Ereigniskanal. Dazu fordert der ECH vom ECB ein Event-Tag an, indem er eine Anforderungsnachricht mit der globalen ID des Ereigniskanals versieht und an den ECB sendet. Der ECB überprüft, ob der Ereigniskanal bereits ein Event-Tag zugeordnet bekommen hat. Ist ein Kanal noch nicht gebunden, wird ein neues Event-Tag erzeugt. Das Event-Tag für den angeforderten Ereigniskanal wird an den ECH übermittelt. Der ECH kann nun den Ereigniskanal adressieren und für die Anwendung Ereignisnachrichten dieses Kanals empfangen oder der Anwendung erlauben in diesem Kanal zu publizieren. Die Bindung des Event-Tags an die UID erfolgt damit transparent für die Anwendung.

Mit COSMIC erfolgt Kommunikation in einer verteilten Anwendung transparent über das Publish/Subscribe-Konzept. Dadurch sind Sender und Empfänger entkoppelt und die Anwendung kann ohne die Kommunikation zu berücksichtigen entwickelt werden. COSMIC stellt dafür eine API bereit, welche die Ereigniskanäle und die darauf anwendbaren Funktionen (`Announce`, `Subscribe` und `Publish`) implementiert. Zwischen der Anwendung und dem Kommunikationscontroller verwaltet der ECH die Adressbindung, die Ereigniskanäle, die Abonnements und informiert die Anwendung über neue Ereignisnachrichten. Die Adressierung der Ereigniskanäle erfolgt mittels dynamisch zugewiesener Netzwerkadressen (Event-Tag), welche anhand eines global eindeutigen Betreffs durch den zentralen ECB vergeben werden. Damit ist die Adressierung der Ereigniskanäle für die Anwendung transparent realisiert. COSMIC verbirgt eine möglicherweise heterogene Infrastruktur vor der Anwendung und erlaubt so eine dynamische Kommunikation von Knoten aus verschiedenen Netzen. Die COSMIC-Middleware ist in [KBM05, KBM03, KB02, KM99] beschrieben.

Kapitel 3

Ereignis- und zeitbasierte Kommunikation

Dieses Kapitel befasst sich mit Methoden zur gemeinsamen Kommunikation von ereignis- und zeitgesteuerten Kommunikationsmodellen und stellt ein Konzept für die Kopplung beider Modelle vor. Grundlegend gibt es zwei Möglichkeiten eine modellübergreifende Kommunikation zu erreichen: die ereignis- und zeitgesteuerte Kommunikation wird innerhalb eines hybriden Netzes realisiert oder ereignis- und zeitgesteuerte Netze werden über Gateways miteinander verbunden. Innerhalb des ersten Ansatzes können die Knoten zeit- und/oder ereignisgesteuert kommunizieren. Dabei ist es denkbar, dass einige Knoten nur eine der beiden Kommmunikationsarten unterstützen. Der Nachteil dieses Ansatzes ist, dass beiden Kommunikationsmodellen nicht die komplette Bandbreite des Mediums zur Verfügung steht und die Vorteile beider Modelle nicht uneingeschränkt genutzt werden können. So müssen die Knoten, welche nur die ereignisgesteuerte Kommunikation unterstützen, sich trotzdem mit der globalen Zeit synchronisieren, obwohl sie selbst keine Synchronisation benötigen. Der Abschnitt 3.1 beschreibt einen solchen Ansatz genauer. Der Abschnitt 3.2 setzt sich mit der Möglichkeit auseinander, innerhalb eines zeitgesteuerten Netzes eine ereignisgesteuerte Kommunikation zu emulieren.

Der Ansatz ein Gateway zwischen ein ereignis- und zeitgesteuertes Netz zu platzieren, teilt die Kommunikation in Bereiche (Domäne, *domain*) ein. Durch diese Einteilung können Informationen, die zwischen den Bereichen verkehren, kontrolliert werden, so dass nur relevante Nachrichten den Übergang zwischen den Netzen passieren. Ein weiterer Vorteil liegt darin, dass die Netze physisch getrennt sind und so innerhalb der Bereiche unterschiedliche Kommunikationsprotokolle mit allen ihren Vorteilen genutzt werden können. Der entscheidende Nachteil eines Gateways ist, dass es als zusätzliche Komponente, welche die Kommunikation von vielen Knoten handhabt, einen Angriffspunkt für Fehler und Ausfälle (*single point of failure*) bietet. Auf der anderen Seite ermöglicht die Verwendung eines Gateways aber verschiedene höherwertige Dienste, wie netzübergreifende Adressierung oder die Pufferung von Nachrichten bei hoher Netzauslastung (siehe dazu die Beschreibung eines Gateway in Abschnitt 2.4). Der Abschnitt 3.3 spezifiziert die Anforderung an das Konzept eines Gateways. Das im Rahmen dieser Arbeit entworfene Konzept eines Gateways zur Kommunikation von ereignis- und zeitgesteuerten Netzen ist in Abschnitt 3.4 beschrieben.

3.1 Zeit- und ereignisgesteuerte Kommunikation auf einem Medium

Der Ansatz, die ereignis- und zeitgesteuerte Kommunikation innerhalb eines Protokolls zu integrieren, soll die Vorteile beider Modelle in einem Netz vereinen. Die Protokolle FlexRay und TTCAN (siehe Unterabschnitt 2.3.6 und 2.3.7) setzen diese Herangehensweise erfolgreich um. Beide Protokolle verwenden eine rundenbasierte Kommunikation. Eine Runde wird in Phasen für ereignisgesteuerte und zeitgesteuerte Kommunikation eingeteilt. Diese Phasen werden im Weiteren mit dynamischer und statischer Phase bezeichnet. Für deren Unterteilung wird der globale Kommunikationsplan über TDMA aufgeteilt. Die Abbildung 3.1 zeigt die Aufteilung der Kommunikation in die Phasen.

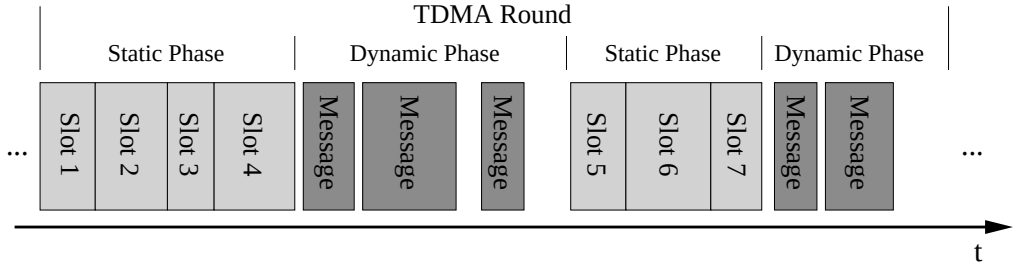


Abbildung 3.1: Aufteilung der Kommunikation in Phasen nach [PEP02]

Der Zugriff auf das Medium ist in beiden Phasen unterschiedlich. In der statischen Phase sind Zeitschlitze für die entsprechenden Knoten reserviert. Innerhalb dieser Zeitschlitze ist eine deterministische Kommunikation ohne Jitter möglich. Statische Phasen eignen sich besonders für periodische Nachrichten.

Im Gegensatz zur festen Zuweisung der Nachrichten in den statischen Phasen, müssen sich die Nachrichten der sendenden Knoten in den dynamischen Phasen prioritätsbasiert um das Kommunikationsmedium bewerben. Die Nachrichten mit den höchsten Prioritäten werden innerhalb einer Phase übermittelt. Nachrichten, welche keinen Platz in einer dynamischen Phase finden, werden bis zur nächsten dynamischen Phase verzögert und versuchen erneut den Bus zu erhalten.

Die prioritätsbasierte Arbitrierung in der dynamischen Phase sichert die Übermittlung wichtiger Nachrichten zu und die ereignisgesteuerte Kommunikation minimiert die mittlere Wartezeit für sporadisch auftretende Nachrichten. Daher ist die dynamische Phase speziell für nicht periodische (Ereignis-)Nachrichten geeignet. Der Unterschied zur rein ereignisgesteuerten Kommunikation liegt darin, dass der Knoten die Nachricht nicht beim Auftreten des Ereignisses übermitteln kann, sondern nur innerhalb einer dynamischen Phase. Es entspricht daher einer eine Einschränkung der Bandbreite, da das Medium nicht jederzeit genutzt werden kann. Für die temporale Kontrolle bedeutet die Aufteilung in die Phasen, dass der Kommunikationscontroller außerhalb der dynamischen Phase eine Versendung von Ereignisnachrichten unterdrücken muss. Damit liegt die Kontrolle über die Kommunikation nicht in der Anwendung, wie bei ereignisgesteuerten Systemen, sondern wie bei der zeitgesteuerten Kommunikation innerhalb des Kommunikationkontrollers. Jeder Knoten agiert somit zeitgesteuert.

Eine Fehlerausbreitung zwischen den beiden Kommunikationsphasen muss beim hybriden Kommunikationsmodell verhindert werden. Eine ereignisgesteuerte Komponente hat somit sicherzustellen, dass sie nicht außerhalb der dynamischen Phase sendet und dadurch Fehler in der zeitgesteuerten Kommunikation verursacht. Damit kann ab einem gewissen Punkt in der dynamischen Phase keine neue Nachricht gesendet werden, da sie sonst in die folgende statische Phase hineinreicht. Der nicht nutzbare Teil am Ende der dynamischen Phase hat die Länge der längsten Nachricht. Das Sendeverbot setzt durch, dass keine Nachrichten aus dynamischen Phasen die zeitgesteuerte Kommunikation der folgenden statischen Phase stören [Obe02].

Die Aufteilung eines globalen Kommunikationsplans in dynamische und statische Phasen ermöglicht zwar die effiziente Übermittlung von periodischen und nicht periodischen Nachrichten, erfordert aber eine zeitliche Synchronisation aller Komponenten. So müssen auch einfache Komponenten, welche nur sporadisch Ereignisse versenden, den globalen Kommunikationsplan kennen und einhalten. Bei einer Änderung des globalen Kommunikationsplans wie Verkleinerung, Vergrößerung, Hinzufügen oder Entfernen einer statischen Phase, erhalten alle Knoten den geänderten Plan. Knoten, welche nur ereignisgesteuert kommunizieren, müssen damit auch an Änderungen der zeitgesteuerten Kommunikation angepasst werden.

Die Nutzung eines globalen Kommunikationsplans erfordert eine Synchronisation jedes Knotens mit der globalen Zeit. Diese Uhrensynchronisation ist aber speziell für sehr einfache Komponenten mit geringer Rechenleistung, wie z.B. einfache Sensoren, nur schwer realisierbar und erhöht die Kosten dieser Komponenten.

Der Nachteil der modellübergreifenden Kommunikation mit einem Protokoll liegt folglich bei der fehlenden Unterstützung von rein ereignisgesteuerten Knoten. Durch die Zielsetzung dieser Arbeit, ereignisgesteuerte Kommunikation mit zeitgesteuerter zu kombinieren, kann ein Ansatz, der die Einschränkung der ereignisgesteuerten Kommunikation bedingt, nicht berücksichtigt werden. Die Vorteile beider Kommunikationsmodelle sollen in das Konzept zur Kopplung mit eingehen. Ein hybrides Kommunikationsprotokoll wird daher für die Erarbeitung des Konzeptes und die Umsetzung nicht betrachtet.

3.2 CAN-Emulation in TTP/C

Das CAN-Protokoll bietet Vorteile wie Flexibilität, kleine mittlere Verzögerungen, Erweiterbarkeit und Fehlersignalisierung. Wegen dieser Eigenschaften hat das CAN-Protokoll, speziell im automotiven Bereich, eine hohe Verbreitung und Akzeptanz. Aufgrund sich ändernder Anforderungen z.B. durch Einführung von x-by-wire Systemen, rücken zeitgesteuerte Protokolle immer mehr in das Anwendungsgebiet des CAN-Protokolls. Die zeitgesteuerten Protokolle ermöglichen eine zeitkritische fehlertolerante vorhersehbare Kommunikation auch in Phasen von hoher Netzbelastung. Diese Anforderungen können mit CAN auf Protokollebene nicht erreicht werden. Im CAN-Protokoll fehlen Mechanismen zur Überwachung von Deadlines, die Unterstützung von deterministischer Kommunikation und Methoden zur Fehlertoleranz durch Redundanz. Auch der Mechanismus für die Selbstabschaltung eines fehlerhaften CAN-Knotens verursacht eine lange Störung (siehe [RV95]) der Kommunikation auf dem Bus und ist daher nicht geeignet für zeitkritische Anwendungen. Da das Bedürfnis besteht, alte bereits erprobte Anwendungen und Erfahrungen mit bekannten Systemen weiter zu nutzen, wird in der aktuellen Entwicklung versucht das CAN-Protokoll mit zeitgesteuerten Eigenschaften auszustatten (siehe TTCAN im Unterabschnitt 2.3.7) oder mit zeitgesteuerten Protokollen zu verbinden.

In [Obe02, Obe05] wird ein Konzept beschrieben, welches das CAN-Protokoll in TTP/C emuliert. Damit können CAN-Anwendungen in eine zeitgesteuerte Umgebung eingebettet werden. Die Verwendung des unterliegenden TTP/C-Protokolls erhöht die Verlässlichkeit der Kommunikation durch die Nutzung der Fehlertoleranzmechanismen von TTP/C. Weiterhin wird durch die Nutzung von knoteneigenen Sendebereichen für CAN-Nachrichten die Nachrichtenübermittlung der einzelnen Knoten entkoppelt. Es vereinfachen sich so die Betrachtungen für die Auslastung der Bandbreite und den entstehenden Latenzzeiten, weil anstatt der Analyse des gesamten CAN-Netzes die Betrachtung für jeden Knoten separat vorgenommen wird. Die CAN-Emulation bietet zwei Kommunikationsdienste für CAN-Anwendungen an:

- Ein einfacher Kommunikationsdienst erlaubt die Verwendung bereits bestehender CAN-Anwendungen.
- Ein erweiterter Kommunikationsdienst bietet eigens für die CAN-Emulation entwickelte Anwendungen mehr Funktionalitäten.

Der einfache Kommunikationsdienst bietet bestehenden CAN-Anwendungen die realistische CAN-Kommunikation im TTP/C-Netz an. Damit können Alt-Anwendungen (*legacy systems*) in die zeitgesteuerte Umgebung überführt werden, ohne dass Aufwand für die Neuentwicklung bzw. Anpassung und das Testen der Anwendung entsteht. Die weiteren Vorteile dieser Lösung sind die transparente Einführung von Fehlertoleranz und Vorhersagbarkeit. Der erweiterte Kommunikationsdienst ist für CAN-Anwendungen konzipiert, welche speziell für die CAN-Emulation entwickelt werden. Er bietet den Anwendungen höhere Bandbreiten (je nachdem mit welcher Bitrate das TTP/C betrieben wird) und weitere durch TTP/C ermöglichte Dienste (z.B. globale Zeit oder Membership-Dienst) an.

Kommunikationsplan

Die Implementierung von ereignisgesteuerter Kommunikation auf Basis eines zeitgesteuerten Protokolls, sieht, wie in Abschnitt 3.1 dargestellt, Phasen von zeit- und ereignisgesteuerter Kommunikation vor. Die CAN-Emulation greift hierbei nicht direkt auf das Medium zu, sondern verwendet die Schnittstelle des zeitgesteuerten TTP/C-Kommunikationscontrollers. Jeder Knoten hat im TTP/C-Protokoll ein eigenes Sendezeitfenster. Soll ein Knoten ereignisgesteuert kommunizieren, wird ein Teil seines Zeitfensters reserviert. Eine solche Reservierung wird als CAN-Zeitschlitz (*CAN Slot*) bezeichnet und ist in Abbildung 3.2 dargestellt. Die Größe des Zeitschlitzes ist abhängig vom Datenaufkommen der CAN-Anwendung und der Bandbreite des Netzes. Dadurch, dass jeder Knoten seinen eigenen exklusiven CAN-Zeitschlitz besitzt, entfällt die Arbitrierung und der Zugriff innerhalb des CAN-Zeitschlitzes wird vereinfacht. Die direkte Zuordnung von CAN-Zeitschlitzen zu Knoten ermöglicht weiterhin eine Bandbreitengarantie für alle CAN-basierten Anwendungen auf einem Knoten. Damit lässt sich der Jitter besser einschätzen und kontrollieren. Andererseits sinkt die gesamte Bandbreitenauslastung aller CAN-Zeitschlitze, durch die fehlende gemeinsame Nutzung der Zeitschlitze.

Struktur

Eine Aufteilung der Kommunikation innerhalb des Knotens erfordert eine spezielle Struktur, welche in Abbildung 3.3 dargestellt ist. Wie zu erkennen ist, werden die CAN-basierten und

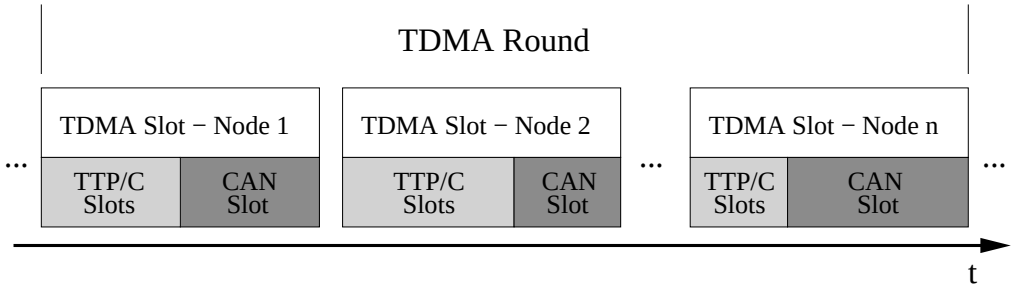


Abbildung 3.2: Kommunikationsrunde der CAN-Emulation in TTP/C nach [Obe05]

die zeitgesteuerten Anwendungen voneinander getrennt. Die zeitgesteuerten Anwendungen verhalten sich wie in einem rein zeitgesteuerten Knoten (vgl. Abbildung 2.4 auf Seite 30) und kommunizieren ihre Statusnachrichten über das CNI zum Kommunikationscontroller. Die CAN-basierten Anwendungen werden so in den Knoten eingebettet, dass sie nur über einen emulierten CAN-Controller Nachrichten empfangen und senden können. Die Schnittstelle (*CAN-CNI*) zu diesem Controller ermöglicht es den CAN-basierenden Anwendungen wie mit einem normalen CAN-Kommunikationscontroller zu interagieren. Dadurch können Alt-Anwendungen transparent und ohne Anpassungen mit der CAN-Emulation in das TTP/C-Netz eingebunden werden.

Da sich zeit- und ereignisgesteuerte Anwendungen auf einem zeitgesteuerten Knoten befinden können, muss in der Planung berücksichtigt werden, dass die zeitgesteuerten Anwendungen harte zeitliche Eigenschaften haben. Sie dürfen durch die CAN-basierten Anwendungen nicht verzögert werden. Dies wird gewährleistet, wenn die zeitgesteuerten Anwendungen und der emulierte CAN-Controller als zeitgesteuerte Tasks realisiert werden. Der emulierte CAN-Controller kann nur als zeitgesteuerter Task umgesetzt werden, weil er auf den CAN-Zeitschlitz zugreift und bis zum Beginn dieses Zeitschlitzes alle dafür vorgesehenen Daten bereitgestellt haben muss. Die CAN-basierten Anwendungen eignen sich hingegen nicht für zeitgesteuerte Tasks, da sie nicht planbar sind. Dies ist mit dem ereignisgesteuerten Anwendungskonzept begründet, welches durch die Verwendung von Interrupts und dem nicht vorhersehbaren Eintreten von Ereignissen keine WCET (*worst case execution time*) für Tasks angeben kann. Die CAN-basierten Anwendungen haben keine harten zeitlichen Begrenzungen und werden daher auf den TTP/C-Knoten in speziellen unterbrechbaren Anwendungstasks untergebracht. Die zeitgesteuerten Tasks können damit die Tasks der CAN-basierten Anwendungen jederzeit unterbrechen. Bei der Planung der Anwendung ist zu beachten, dass genügend Prozessorzeit für die Anwendungstasks der CAN-basierten Anwendungen bereit steht. CAN-basierte Anwendungen dürfen zeitgesteuerte Tasks auch nicht indirekt durch das Halten von Betriebsmitteln blockieren. Das Task- und Betriebsmittelmodell der CAN-Emulation verhindert, dass eine CAN-basierte Anwendung durch ein Fehlverhalten die zeitgesteuerten Anwendungen verzögert und sich Fehler der CAN-basierten Anwendungen auch auf die zeitgesteuerten Anwendungen ausdehnen.

Emulierter CAN-Controller

Der emulierte CAN-Controller (*Emulated CAN Controller*) befindet sich, wie in Abbildung 3.3 dargestellt, zwischen der CAN-basierten Anwendung und dem TTP/C-Kommunikationscontroller. Er bietet der CAN-basierten Anwendung dieselben Schnittstellen wie ein normaler CAN-Controller, welcher an einen CAN-Bus angeschlossen ist, nutzt aber für

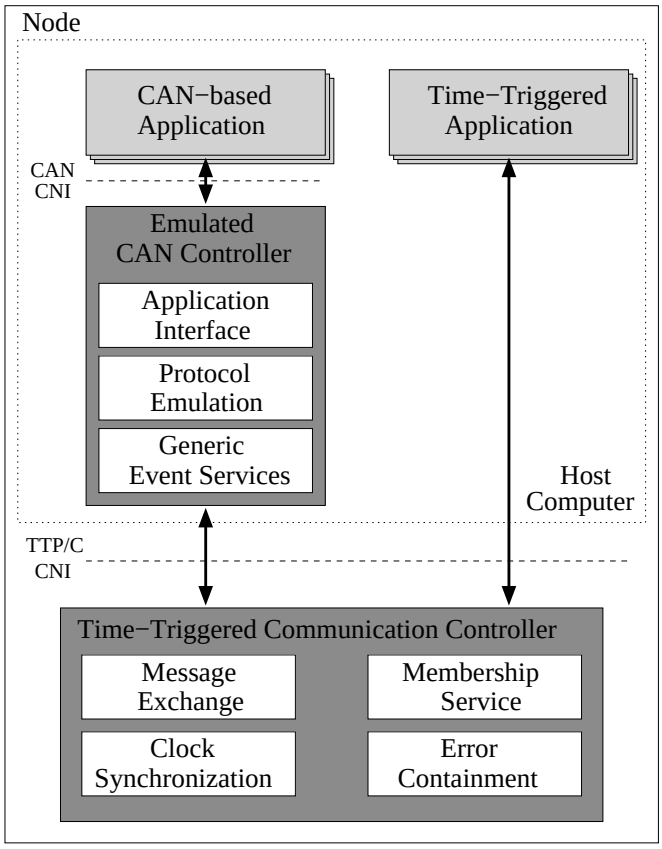


Abbildung 3.3: Struktur eines Knoten mit CAN-Emulation in TTP/C nach [Obe05]

die Kommunikation die Schnittstelle des TTP/C-Kommunikationscontrollers (*TTP/C-CNI*). Der emulierte CAN-Controller besteht aus drei Schichten: der generischen Ereignisdienstschicht (*generic event service layer*), der Protokollemulationsschicht (*protocol emulation layer*) und der Anwendungsschnittstellenschicht (*application interface layer*).

Die **Ereignisdienstschicht** übermittelt die Nachrichten der CAN-basierten Anwendung über den TTP/C-Kommunikationscontroller. Die Ereignisdienstschicht verwendet dafür einen Paketierungsdienst (*packet service*), welcher die Umwandelung der CAN-Nachrichten auf das vom TTP/C-Kommunikationscontroller unterstützte Format übernimmt. Die ausgehenden Nachrichten werden dafür in Pakete (Fragmente) aufgeteilt und durch die TTP/C-CNI an den zeitgesteuerten Kommunikationscontroller übergeben. Dieser übermittelt die Pakete in dem CAN-Zeitschlitz. Je nach Größe des CAN-Zeitschlitzes ist es möglich, dass gleichzeitig Pakete von mehreren CAN-Nachrichten oder nur Pakete mit Teilen einer CAN-Nachricht in einem CAN-Zeitschlitz Platz haben. Damit ist die Größe des CAN-Zeitschlitzes frei wählbar und kann der benötigten Bandbreite der CAN-basierten Anwendung ideal angepasst werden. Die Bandbreite des CAN-Zeitschlitzes wird durch die Verwendung des Paketierungsdienstes optimal ausgelastet, denn es gibt durch die Fragmentierung der Nachrichten (im Gegensatz zu den hybriden Protokollen in Abschnitt 3.1) keinen Teil innerhalb des Zeitfensters, der für ereignisgesteuerte Kommunikation nicht genutzt werden kann.

Empfangene Pakete, aus den CAN-Zeitschlitzen der anderen Knoten, werden vom zeitgesteuerten Kommunikationscontroller an den Ereignisdienst übergeben. Die Ereignisdienstschicht fügt die empfangenen Pakete wieder zu CAN-Nachrichten zusammen und gibt sie an die Protokollemulationsschicht weiter. Durch dieses Konzept haben alle CAN-basierten Anwendungen auf einem Knoten eine gemeinsame lokale Sendewarteschlange und mehrere Empfangswarteschlangen. Die Zahl der Empfangswarteschlangen entspricht dabei der Anzahl der Sender mit denen die CAN-basierten Anwendungen kommunizieren. Die Aufteilung der Empfangswarteschlangen nach den Sendern verhindert, dass ein fehlerhafter Sender (z.B. durch Senden von falschen Nachrichten) den korrekten Empfang von Nachrichten anderer Knoten stört.

Die Ereignisdienstschicht verwendet zwei verschiedene Nachrichtentypen im CAN-Zeitschlitz: erweiterte CAN-Nachrichten und Initialisierungsnachrichten. Die erweiterten CAN-Nachrichten entsprechen den normalen CAN-Nachrichten, welche die CAN-basierte Anwendung verwendet. Zusätzlich zu den im CAN-Protokoll definierten Feldern (siehe Abbildung 2.1) sind in einer erweiterten CAN-Nachricht weitere Flags und ein Zeitstempel enthalten. Die Initialisierungsnachrichten werden für den Start des Systems und die Reintegration von Knoten benötigt. Sie enthalten Teile des Status des emulierten CAN-Controllers. Die Initialisierungsnachrichten sind notwendig, da die CAN-Nachrichten fragmentiert in den CAN-Zeitschlitz übermittelt werden und allen empfangenden Knoten die Position von beginnenden Nachrichten im CAN-Zeitschlitz bekannt sein muss. Knoten können sich nur in die Kommunikation integrieren, wenn sie die Fragmentierungsinformation von allen Knoten erhalten und selbst eine Initialisierungsnachricht senden.

Die **Protokollemulationsschicht** simuliert einen CAN-Bus und bietet so der Anwendungschnittstelle ein realistisches Verhalten der CAN-Kommunikation. In jedem Knoten wird dabei der CAN-Bus unabhängig von den anderen Knoten simuliert. Ein- und Ausgabe für diese Schicht bilden die CAN-Nachrichten, welche über den Ereignisdienst oder die Anwendungsschnittstelle ausgetauscht werden. Die **Anwendungsschnittstellenschicht** wird direkt von der CAN-basierten Anwendung für die Nachrichtenübermittlung genutzt. Dabei wird in diesem Modul eine Schnittstelle eines tatsächlichen CAN-Controllers inklusive des genauen Registersatzes emuliert. So kann eine CAN-basierte Anwendung für einen speziellen CAN-Controller ohne Anpassungen integriert werden. Dieses Modul verfügt über eine Nachrichtenfilterung und ein Warteschlangenmanagement. Das Verhalten und die Schnittstelle der Funktionen hängen davon ab, welcher CAN-Controller in der Anwendungsschnittstellenschicht nachgebildet wird. Die Schichten innerhalb des emulierten CAN-Controllers sind aufgrund der allgemeinen Schnittstellen unabhängig voneinander austauschbar. So kann z.B. durch Austausch der Anwendungsschnittstelle ein anderer CAN-Controller emuliert werden.

Zeitgesteuerter Kommunikationscontroller

Der zeitgesteuerte Kommunikationscontroller greift exklusiv zum definierten Zeitpunkt auf das Medium zu und sendet die dem Zeitschlitz zugeordneten Daten. Der TTP/C-Kommunikationscontroller stellt die in Unterabschnitt 2.3.5 beschriebenen Dienste für die Kommunikationsteilnehmer zur Verfügung. Damit kann auch der emulierte CAN-Controller auf die höherwertigen Dienste des TTP/C-Protokolls zurückgreifen und sie der CAN-Anwendung über den erweiterten Kommunikationsdienst bereitstellen. Da die CAN-Nachrichten durch den zeitgesteuerten Kommunikationscontroller versendet werden, kann eine fehlerhafte CAN-Anwendung weder den Nachrichtentransport von CAN-Anwendungen auf anderen Knoten noch

die zeitgesteuerte Kommunikation stören. Somit werden die Auswirkungen von Babbeling-Idiot-Fehlern auf die Bandbreite des lokalen CAN-Zeitschlitzes begrenzt. Die feste Zuordnung von CAN-Zeitschlitz zu Knoten, ermöglicht es weiterhin den Sender einer empfangenden Nachricht zu identifizieren. Der emulierte CAN-Controller kann diese Funktion für die Anwendung bereitstellen. Der CAN-basierten Anwendung ist es damit möglich zu überprüfen, welcher Knoten keine neuen oder fehlerhafte Nachrichten schickt. Zur Fehlerbegrenzung kann dann die Kommunikation mit diesem Knoten beendet werden. Durch die Zuordnung von Zeitschlitz zu Sender ist es möglich Maskerade-Fehler (*masquerading failure*), bei der ein Sender eine andere Identität vortäuscht, zu erkennen.

Fazit

Die CAN-Emulation stellt eine integrierte Architektur dar, bei der ein oder mehrere CAN-Netze in einem TTP/C-Netz emuliert werden. Mit der CAN-Emulation können ereignisgesteuerte CAN-Teilsysteme mit zeitgesteuerten TTP/C-Anwendungen in einem mit TTP/C vernetzten Gesamtsystem koexistieren. Damit eignet sich dieses Konzept besonders für Systeme, die aus einem zeitkritischen Teilsystem und aus einem zeitunkritischen Teilsystem bestehen und nicht miteinander kooperieren müssen. Die Integration von bestehenden CAN-Anwendungen in TTP/C-Netze erleichtert die Migration von rein ereignisgesteuerten Systemen zu heterogenen Systemen auf Basis einer zeitgesteuerten Kommunikation. Sind nach der Entwicklung von zeitgesteuerten sicherheitskritischen Systemen freie Kapazitäten in Kommunikationsbandbreite und Prozessorzeit auf den Knoten verfügbar, können sie durch ereignisgesteuerte zeitunkritische Systeme einfach ausgenutzt werden. Dabei können die Kosten dieser Systeme durch die Einsparung von Knoten und Verkabelung reduziert werden.

Die CAN-Emulation im zeitgesteuerten TTP/C erlaubt die Kopplung mit einem CAN-Netz. Dabei fungiert ein TTP/C-Knoten als Gateway und leitet Nachrichten von konventionellen CAN-Knoten aus dem CAN-Netz zu den CAN-basierten Anwendungen im zeitgesteuerten Netz und umgekehrt. Diese Gatewaykomponete kann alle Vorteile eines Gateways, wie z.B. die Steuerung des Informationsflusses, anbieten.

Der Nachteil der CAN-Emulation ist, dass eine Kommunikation zwischen dem zeitgesteuerten und dem ereignisgesteuerten Teilsystem nicht vorgesehen ist. In diesem Konzept fehlt eine Komponente, welche die Nachrichten für das empfangende Teilsystem so umwandelt, dass es die Informationen transparent und unabhängig vom Teilsystem empfangen kann. Soll eine zeitgesteuerte Anwendung mit einer CAN-basierten Anwendung kommunizieren, so ist dies nur über die CAN-Emulation möglich. Dazu muss ein emulierter CAN-Controller an die zeitgesteuerte Anwendung angeschlossen werden. Ein solcher Ansatz ist keine erstrebenswerte Lösung, da keine transparente Kommunikation zwischen den Anwendungen umgesetzt wird und diese unnötig komplex gestaltet werden muss. Die Kommunikation eines echten CAN-Knotens mit einer Anwendung aus dem zeitgesteuerten Teilsystem ist in der beschriebenen Gatewaykomponete nicht vorgesehen, da die die CAN-basierten Anwendungen strikt von den TTP/C-Anwendungen getrennt sind. Dieses Konzept eignet sich nicht für Szenarien in denen eine Kommunikation zwischen den Modellen erfolgen soll. Da eine solche Kommunikation zur Zielsetzung dieser Arbeit gehört, kann das Konzept der CAN-Emulation in TTP/C nicht angewendet werden.

3.3 Anforderungen an die Architektur

Die zuvor erläuterten Ansätze zur gemeinsamen Kommunikation von zeit- und ereignisgesteuerten Kommunikationsmodellen mit ganzheitlicher Kommunikationsplanung und der CAN-Emulation in TTP/C eignen sich aufgrund unterschiedlicher Gründe nicht für die gestellten Anforderungen.

FlexRay sowie andere Verfahren zur ganzheitlichen Planung der Kommunikation in einem Netz kombinieren ereignis- wie zeitgesteuerte Kommunikation und erlauben auch zum Teil[1] die Kommunikation untereinander. Dadurch, dass alle gezeigten Ansätze auf zeitlich getrennte Kommunikationsphasen basieren, müssen alle Knoten synchron arbeiten. Die ereignisgesteuert kommunizierenden Knoten müssen daher auch einen hinreichend genauen Zeitgeber besitzen und an der Synchronisation teilnehmen. Dies schränkt die Flexibilität der ereignisgesteuerten Kommunikation ein und verhindert die Einbindung von einfachen preisgünstigen ereignisgesteuerten Knoten in die Kommunikation. Da eine Kopplung eines ereignisgesteuerten Netzes mit einem zeitgesteuerten angestrebt wird, können die in Abschnitt 3.1 vorgestellten Verfahren sowie TTCAN und FlexRay nicht verwendet werden.

Der Ansatz der CAN-Emulation in TTP/C erlaubt keine Kommunikation zwischen zeitgesteuerten und der ereignisgesteuerten Anwendungsteilen. Es können daher nicht die Zielsetzungen dieser Arbeit mit der CAN-Emulation in TTP/C erreicht werden. Weitere vielversprechende Konzepte sind in der Literatur nicht zu finden. Zwar wird in [Bor03] ein Gateway für TTP/C, TTP/A und CAN beschrieben. Die Betrachtung bezieht sich aber auf den Hardwareentwurf einer solchen Komponente und der Bereitstellung der physischen Verbindung. Dienste auf höherer Ebene, wie zur Handhabung von Abonnements, verschiedene Echtzeitklassen oder die Bereitstellung von transparenter Kommunikation zwischen den Netzen, sind nicht im Fokus dieser Quelle.

Aufgrund des Fehlens eines adäquaten Konzeptes zur Kommunikation von zeit- und ereignisgesteuerten Netzen wird eine eigene Lösung auf Basis eines Gateways entwickelt. Bevor das Konzept erläutert wird, sind die Anforderungen daran im Folgenden dargestellt.

- **Globales Adressierungsschema:**
  Ein globales Adressierungsschema verbirgt die Heterogenität der Kommunikationsinfrastruktur und erlaubt die Interoperabilität zwischen den Modellen. Die verteilte Anwendung kann auf Informationen zugreifen, ohne dass sie wissen muss, wo die Informationen zu finden sind.

- **Kontrolle des Informationsflusses:**
  Die Kontrolle des Informationsflusses zwischen beiden Modellen erlaubt eine Kapselung der Informationen in den Netzen. So kann eine sicherheitskritische Echtzeitanwendung in einem zeitgesteuerten Netz und eine zeitunkritische Anwendung in einem ereignisgesteuerten Netz platziert werden. Es sollen dabei nur Informationen die Netzgrenzen passieren, welche auf der anderen Seite auch tatsächlich benötigt werden. Damit wird im jeweiligen Netz keine Bandbreite durch unnötigen Verkehr aus dem anderen Netz verschwendet. Weiterhin kann eine Kontrolle des Informationsflusses beide Netze (zeitlich) entkoppeln und so die Fehlerverbreitung über die Netzgrenze hinaus behindern.

[1] Der Ansatz aus [PEP02] schließt eine Kommunikation zwischen ereignis- und zeitgesteuerten Anwendungen aus.

- **Unterstützung von Echtzeitkommunikation:**
  Innerhalb beider Netze ist eine Echtzeitkommunikation basierend auf qualitativen und temporalen Bedingungen möglich. Eine Kopplung beider Netze soll eine konsistente Echtzeitkommunikation zwischen beiden Kommunikationsmodellen ermöglichen. Dabei sollen die zeitlichen und qualitativen Anforderungen an die Kommunikation durch den Netzübergang nur gering beeinflusst werden. Es soll eine sicherheitskritische Echtzeitkommunikation und eine Best-Effort Kommunikation unterstützt werden.

Ein Gateway kann nach Abschnitt 2.4 die gestellten Anforderungen erfüllen und so die Kopplung der Kommunikationsmodelle ermöglichen. Das Gateway verbindet beide Netze und ermöglicht so ein netzübergreifendes verteiltes System. Im Folgenden werden die Kommunikationsprotokolle, welche vom Gateway unterstützt werden, ausgewählt.

3.3.1 Auswahl der Kommunikationsprotokolle

Das Gateway muss zur Verbindung beider Kommunikationsmodelle zwei Protokolle verwenden. Das ereignisgesteuerte Protokoll soll eine flexible einfache Kommunikation ermöglichen und das zeitgesteuerte einen deterministischen Nachrichtenaustausch für Echtzeitanwendungen erlauben.

Ereignisgesteuertes Protokoll

Das CAN-Protokoll ist das gängigste Protokoll im Automobilbereich [Alb04] und wird auch in anderen industriellen Bereichen wie der Automatisierungstechnik oft angewandt [Rei02]. Es hat durch die Abhängigkeit von der Signallaufzeit der Bits nur eine geringe Bandbreite, kann aber dadurch eine kollisionsauflösende Busarbitrierung realisieren. Die Fehlersignalisierung des CAN-Protokolls stellt sicher, dass alle am Bus angeschlossenen Knoten eine konsistente Sicht auf die Nachrichten haben. Erkennt ein Knoten einen Fehler in einer Nachricht, löst er die Fehlersignalisierung aus, bei der alle Busteilnehmer die Nachricht verwerfen. Damit wird die netzweite Datenkonsistenz durchgesetzt. Eine automatische Neuübermittlung durch den Kommunikationscontroller des Senders versucht die Nachricht im Fehlerfall erneut korrekt zu übertragen. Die Mechanismen der Fehlerbegrenzung verhindern, durch die Unterdrückung der aktiven Fehlersignalisierung oder die Abschaltung defekter Knoten, die Blockierung des Busses durch fehlerhafte Kommunikationsteilnehmer. Diese Eigenschaften erlauben es mit dem CAN-Protokoll eine flexible, sichere, kosteneffiziente aber nicht vorhersagbare Kommunikation zu realisieren. Das in dieser Arbeit vorgestellte Gateway wird aus diesen Gründen auf der ereignisgesteuerten Seite für das CAN-Protokoll ausgelegt.

Zeitgesteuertes Protokoll

Der Abschnitt 2.3 hat verschiedene zeitgesteuerte Kommunikationsprotokolle dargestellt. TTCAN, als Erweiterung des CAN Protokolls, bietet eine geringe Bandbreite und nur eingeschränkte Mechanismen der Fehlertoleranz. TTCAN ist daher nicht optimal für eine sicherheitskritische Echtzeitkommunikation und wird im Weiteren nicht betrachtet. FlexRay entspricht allen Anforderungen an eine deterministische sicherheitskritische Echtzeitkommunikation. Das Protokoll wird aber aufgrund der Einordnung als hybrides Protokoll für das Gateway nicht

in Betracht gezogen. Das TTP/C-Protokoll ist für sicherheitskritische Anwendung (z.B. x-by-wire) im automotiven und avionischen Bereich konzipiert. TTP/C hat eine hohe Bandbreite und besitzt aufgrund seines festen Kommunikationsplanes ein deterministisches Verhalten. Die Fehlertoleranz in TTP/C bietet z.B. die Möglichkeit mehrere Knoten als fehlertolerante Einheit zu betreiben. Damit wird sichergestellt, dass bei Ausfall einer Komponente (Kanal oder Knoten) eine wichtige Funktion in der verteilten Anwendung weiter zur Verfügung steht. Mit der architekturellen Trennung des Applikationsrechners vom Kommunikationscontroller und der Verwendung von Bus-Guardians wird die Verbreitung von zeitlichen Fehlern über das Netz verhindert. Aufgrund der Eigenschaften des TTP/C-Protokolls wird es für die zeitgesteuerte Kommunikation des Gateways verwendet.

3.3.2 Einfaches Gateway

Die grundlegende Verbindung von CAN- und TTP/C-Netzen kann mit Hilfe eines einfachen Gateways realisiert werden. Dabei leitet das Gateway CAN-Nachrichten in einen TTP/C-Zeitschlitz. Das Gateway kann die CAN-Nachrichten filtern oder den kompletten Verkehr des CAN-Netzes weiterleiten. Die Filterung der CAN-Nachrichten ist zu empfehlen, da sonst zu viel Bandbreite im TTP/C-Netz durch Einleitung aller CAN-Nachrichten verbraucht wird. Der CAN-Identifier und das Längenfeld müssen neben den Nutzdaten für jede CAN-Nachricht im TTP/C-Zeitschlitz mitgesendet werden, weil diese Felder zur Identifizierung und zum korrekten Empfang von CAN-Nachrichten benötigt werden. Ein TTP/C-Knoten empfängt Nachrichten aus dem CAN-Netz, indem er die CAN-Nachrichten aus einem vom Gateway befüllten Zeitschlitz liest.

Um Nachrichten an CAN-Knoten zu senden, hat jeder TTP/C-Knoten einen Zeitschlitz mit fester Bandbreite, in dem er Nachrichten mit CAN-Identifier, Länge und Nutzlast ablegt. Das Gateway wandelt diese Nachrichten in CAN-Nachrichten um und sendet sie ins CAN-Netz. Müssen dieselben Informationen von einem TTP/C-Knoten an Empfänger im CAN- und TTP/C-Netz gesendet werden, muss der Sender entweder einen weiteren Zeitschlitz für die Kommunikation mit dem TTP/C-Empfänger verwenden oder der empfangende Knoten muss den CAN-Zeitschlitz des Senders lesen. Die erste Variante versendet die Informationen redundant, während die zweite Lösung aufgrund der begrenzten Bandbreite des CAN-Antwortzeitschlitzes und der eventuellen Verwendung von mehreren Nachrichten in diesem Kanal keine vorhersehbare Kommunikation bietet.

Da das Gateway keine Adressumsetzung bereitstellt, müssen Nachrichten im TTP/C-Netz, welche aus dem CAN-Netz empfangen werden oder ins CAN-Netz weitergeleitet werden sollen, über den CAN-Identifier adressiert werden. Es folgt daher, dass eine TTP/C-Anwendung die Adressierungsmechanismen des CAN-Netzes nutzen muss, wenn sie Nachrichten ins CAN-Netz sendet oder daraus erhält. Dazu müssen die CAN-Identifier der TTP/C-Anwendung bekannt sein und somit im Voraus statisch vergeben werden.

Die CAN-Knoten können mit der in CAN spezifizierten Adressierung Nachrichten mit CAN- und TTP/C-Knoten austauschen. Die TTP/C-Knoten hingegen müssen in der Kommunikation je nach Kommunikationspartner unterschiedlich verfahren. Kommuniziert ein TTP/C-Knoten mit einem CAN-Knoten, müssen die CAN-Identifier verwendet und die ausgehenden Nachrichten speziell formatiert werden. Bei der Kommunikation mit einem anderen TTP/C-Knoten

werden alle Daten wie bei TTP/C üblich aus einem bestimmten Zeitschlitz gelesen und die Antworten in einen bestimmten Zeitschlitz geschrieben.

Die Filterung der eingehenden Nachrichten in das TTP/C-Netz und die einfache Herausleitung von Kanälen durch das Gateway bietet nur eine schwache Kontrolle des Informationsflusses, da unabhängig von der Notwendigkeit der Weiterleitung alle Nachrichten das Gateway passieren. Weiterhin ist eine Unterstützung von Echtzeitkommunikation im CAN-Netz in diesem Ansatz nicht berücksichtigt. Die Begrenzungen und Nachteile dieses Ansatzes sind damit offensichtlich:

- keine konsistente Adressierung
- keine transparente Kommunikation
- keine Unterstützung von Echtzeitkommunikation
- Weiterleitung der Nachrichten ohne Kontrolle des Informationsbedarfs
- feste Vergabe der CAN-Identifier während des Entwurfs
- komplizierte unübersichtliche TTP/C-Anwendung

Wie gezeigt wurde genügt die einfache Konvertierung von Nachrichten zwischen CAN- und TTP/C-Knoten nicht den gestellten Anforderungen. Um die Kernprobleme der Aufgabenstellung zu lösen, wird der Einsatz einer Middleware benötigt.

3.3.3 Auswahl der Middleware

Die Kopplung von Kommunikationsmodellen durch ein Gateway erfordert speziell für das globale Adressierungsschema und die Zusicherung von Echtzeiteigenschaften eine geeignete Middleware. Diese Middleware dient der Abstraktion der Kommunikation und vereinfacht die Umsetzung der Anwendung. Im folgenden Unterabschnitt werden die bereits vorgestellten Middleware Systeme unter dem Aspekt der Echtzeitkommunikation betrachtet. Auf dieser Grundlage wird eine Middleware für das Gateway ausgewählt.

CORBA

Eine Echtzeitkommunikation ist mit CORBA nicht möglich, da in der Architektur die dafür notwendigen Komponenten nicht integriert sind. CORBA fehlt eine Überwachung der zeitlichen Eigenschaften von Nachrichten und eine Umsetzung des effektiven Sendens asynchroner Nachrichten. Es gibt aber die Möglichkeit CORBA so zu erweitern, dass eine Echtzeitkommunikation unterstützt wird. Im Folgenden werden zwei Ansätze zur Erweiterung von CORBA dargestellt.

Eine CORBA-Implementierung für eingebettete Systeme auf Basis des CAN-Busses ist in [KJH+00] beschrieben. Auf das CAN-Protokoll werden dazu drei Transportprotokolle aufgesetzt, welche Netzmanagement, Client-Server- und Publish/Subscribe-Kommunikation ermöglichen. Das **Client-Server-Protokoll** definiert eine verbindungsorientierte Kommunikation und erlaubt die Zusammenarbeit mit anderen konventionellen CORBA-Implementierungen. Das

Publish/Subscribe-Protokoll ist speziell für verteilte Sensor/Aktor-Systeme entwickelt. Solche Systeme benötigen Multicast-Kommunikation und versenden zeitkritische Daten. Daher hat dieses Protokoll auf dem CAN-Bus auch die höchste Priorität. Das Publish/Subscribe-Protokoll verwendet Kanäle zum Publizieren und Empfangen von Nachrichten. Im Gegensatz zum Ereignisdienst von CORBA halten die Abonnenten die (gebundenen) Identifizierungsinformationen der abonnierten Kanäle selbst. Dadurch werden ihnen die Nachrichten direkt zugestellt, ohne dass eine weiterleitende Komponente[2] zwischen Produzent und Konsument erforderlich ist. Dies erhöht die Effizienz der Nachrichtenübermittlung deutlich. Das **Netzmanagement-Protokoll** bindet die Kanäle und verwaltet die Ankündigungen und Abonnements in einer globalen Datenbank.

Die Umsetzung des Konzeptes strebt aufgrund der begrenzten Bandbreite des CAN-Busses ein geringe Netzauslastung an. Zusätzlich dazu bedarf es einer Minimierung der Leistungsanforderungen an die ORB-Implementierung, um das Konzept auch in Umgebungen mit ressourcenbeschränkten Knoten umsetzen zu können. Die Zusage von zeitlichen Garantien und die Bereitstellung von Mechanismen zur Fehlertoleranz sind in das Konzept von [KJH+00] bisher nicht integriert. Dieses Konzept kann daher nicht verwendet werden.

In [HLS97] wird eine Erweiterung für den Ereignisdienst in CORBA beschrieben, die eine ereigniskanalbasierte Echtzeitkommunikation bereitstellt. Das beschriebene Konzept verfügt über einen zentralen **Echtzeitereigniskanal** (*RT event channel*), der die Nachrichten weiterleitet und dafür sorgt, dass die Echtzeiteigenschaften eingehalten werden. Die Kommunikationspartner werden in Produzenten (*supplier*) und Konsumenten (*consumer*) unterteilt. Die Produzenten senden ihre Nachrichten an den Ereigniskanal (*via information push*) und dieser verteilt die Nachrichten an die entsprechenden Konsumenten (*via information push*). Durch die Verwendung der Ereigniskanalkomponente wird die Kommunikation von Produzent zum Konsument entkoppelt. Damit wissen die Komponenten nicht mit wem sie kommunizieren. Sie müssen sich nur beim Ereigniskanal registrieren und erhalten bzw. senden dann ihre Informationen über den Ereigniskanal. Dies ermöglicht eine modulare sich verändernde Kommunikationsstruktur. So können z.B. Sensoren oder informationsverarbeitende Komponenten zur Laufzeit hinzugefügt, entfernt oder ausgetauscht werden. Darüberhinaus wird mit diesem Ereigniskanalkonzept dem Anwendungsprogrammierer die Last genommen sich mit der Kommunikation der Komponenten selbst zu beschäftigen.

Der Echtzeitereigniskanal besteht aus verschiedenen Modulen z.B. zur Filterung und zum Versand von Nachrichten und zur Überwachung der Fristen (*deadlines*). Über das **Filtermodul** können Konsumenten Ereignisse eines bestimmten Typs und/oder von bestimmten Produzenten beziehen. Eine an die Filterung angeschlossene Komponente, ermöglicht es Beziehungen zwischen abonnierten Ereignissen herzustellen. So kann ein Konsument eine Verknüpfung von Disjunktionen und Konjunktionen von Ereignisfiltern definieren. Wird diese Filteregel durch das Eintreffen von Nachrichten wahr, bekommt der Konsument alle entsprechenden Ereignisse auf einmal. Das **Sende-Modul** (*dispatcher*) beinhaltet eine Sendeplanung (*scheduler*), welche die Echtzeitübertragung der Ereignisse zum Konsumenten durchsetzt. Verschiedene Planungsstrategien, wie z.B. Planen nach monotonen Raten oder Earliest-Deadline-First (EDF) können in der Sendeplanung eingesetzt werden. Das **Modul zur Überwachung der Fristen** sendet, sofern sich ein Konsument für diesen Dienst registriert hat, bei Überschreitung einer Frist für ein Ereignis eine Zeitüberschreitungsmitteilung (*timeout*).

[2] In CORBA vermittelt ein zentraler Event Channel die Nachrichten.

[HLS97] beschreibt einen flexiblen objektorientierten Ansatz zur Echtzeitkommunikation mit CORBA. Die für CORBA typische, synchrone zweiwege (Client-Server) Kommunikation des Ereignisdienstes wurde auf asynchrone Nachrichten und für die Gruppenkommunikation (Multicast, Broadcast) erweitert. Es wird durch den zentralen Echtzeitereigniskanal eine Entkopplung der Kommunikationspartner erreicht, dies ermöglicht eine flexibel erweiterbare Kommunikationsstruktur. Der Echtzeitereigniskanal unterstützt komplexe Abonnements mit korrelierenden Ereignissen. Er sichert eine gleichzeitige Zustellung der Ereignisse an den Konsumenten bei Erfüllung, der vom Konsument definierten, komplexen Filterbedingungen zu. Eine spezielle Komponente im Echtzeitereigniskanal bietet es den Konsumenten an, Zeitüberschreitungsmitteilungen zu senden, wenn abonnierte Ereignisse ausbleiben. Dies lagert die Fehlererkennung von ausgefallenen oder zu langsamen Knoten in den Ereigniskanal aus und ermöglicht so eine einfachere Fehlerbehandlung im Konsumenten. Der Echtzeitereigniskanal kann offline oder zur Laufzeit konfiguriert werden. Damit wird die Erweiterungsfähigkeit auch zur Laufzeit gewahrt. Weitere Informationen zur Echtzeitkommunikation in CORBA sind z.B. in der Spezifikation von Real-Time CORBA [OMG03] nachzulesen.

Der Nachteil des in [HLS97] vorgestellten Konzeptes ist, dass von einem zentralen Ereigniskanal ausgegangen wird. Dieser Ansatz ist für eine netzübergreifende Kommunikation nur begrenzt geeignet. So muss der zentrale Ereigniskanal im Gateway lokalisiert sein, da er aus dem TTP/C- und CAN-Netz erreicht werden muss. Alle Nachrichten werden über den Ereigniskanal im Gateway geleitet, auch wenn sich der Empfänger der Nachricht im selben Netz wie der Sender befindet. Dies schränkt die Eigenständigkeit und Unabhängigkeit der Kommunikation im CAN- und TTP/C-Netz ein. Werden mehrere Netze mit mehreren Gateways gekoppelt muss ein Konzept gefunden werden, wie die Ereigniskanäle in dieser Umgebung eingesetzt werden. Es kann z.B. eine hierarchische Struktur von Ereigniskanälen aufgebaut werden. Dabei hat jeder Ereigniskanal eine eigene begrenzte Reichweite (z.B. das lokale Netz). Um eine Kommunikation über die Reichweite eines Kanals hinaus zu ermöglichen, müssen die Ereigniskanäle untereinander die Nachrichten weiterleiten. Ein hierarchischer Aufbau von Ereigniskanälen verzögert damit die Kommunikation innerhalb der verteilten Anwendung und behindert so die Durchsetzung der Echtzeiteigenschaften von Nachrichten. Weiterhin wird durch dieses Konzept die Skalierbarkeit des verteilten Systems eingeschränkt. Daher wird diese Middlewarelösung nicht für das Gateway verwendet.

Jini

In [BK02] wird ein Konzept dargestellt bei dem Jini für verteilte eingebettete Systeme angewendet wird. Jini arbeitet dafür mit dem CAN-Protokoll zusammen und erlaubt es den CAN-Knoten über objekt-orientierter Schnittstellen zu kommunizieren. Fundamentale Probleme für den Einsatz von Jini in eingebetteten Systemen sind die Nutzung von Java-RMI und die ausschließliche Unterstützung der Protokolle TCP und UDP. Die in der Rechnerkommunikation üblichen Protokolle TCP/IP und UDP/IP werden aufgrund ihrer Eigenschaften, z.B. die Adressierungsart und die großen Nachrichtenlängen, nicht in verteilten eingebetteten Systemen eingesetzt. Jini hat keine klar getrennte Schnittstelle zum Kommunikationsprotokoll und verwendet daher Mechanismen von TCP und UDP. So werden in Jini z.B. Knotennamen und Ports für die Adressierung von Knoten genutzt. Da das CAN-Protokoll die Adressierung der Informationen anders umsetzt, wird die Kopplung von CAN und Jini enorm erschwert. Die Verwendung der von Jini vorgegebenen Adressierung behindert die Performance und verrin-

gert durch zusätzliche unnötige Metainformationen die Kommunikationsbandbreite. Auch das von Jini genutzte Java-RMI verwendet für die Kommunikation TCP-Ports, welche im CAN-Protokoll nicht existieren, daher muss Java-RMI durch eine spezielle Implementierung ersetzt werden. Ein weiteres Problem ist, dass CAN-Nachrichten kleiner sind als von Jini spezifizierten Nachrichten. Jini-Nachrichten müssen daher für den CAN-Bus fragmentiert werden. Die in Jini genutzte und im CAN-Protokoll fehlende Unterstützung von Unicast-Nachrichten muss durch empfängerbasiertes Filtern in jedem CAN-Knoten umgesetzt werden.

Die Portierung von Jini auf das CAN-Protokoll zeigt nach [BK02] signifikante zeitliche Probleme. Es ist beschrieben, dass die Registrierung eines Knotens beim Suchdienst (*Lookup-Service*) aufgrund des schlechten Ressourcenmanagemets bei einem nahezu unbenutzten CAN-Bus mehrere Minuten dauern kann. Solche Verzögerungen sind für eingebettete Systeme inakzeptabel. Das in Jini verwendete Leasing ist für zeitkritische Anwendungen ebenfalls ungeeignet, da Komponenten nur als ausgefallen gelten, wenn sie ihr Gültigkeitsintervall (*lease*) nicht verlängern. Der Suchdienst sendet nur bei ausgelaufenen Gültigkeitsintervallen eine Benachrichtigung, solange keine solche Benachrichtigung über eine Fristüberschreitung eintrifft, gilt der Knoten als aktiv. Um eine schnelle Erkennung von ausgefallenen Knoten zu ermöglichen, müssen die Gültigkeitsintervalle sehr kurz gehalten werden. Dies bedingt eine merkliche Erhöhung der Netzlast, da die Nachrichten zur Erneuerung der Gültigkeitsintervalle in sehr kurzen Abständen gesendet werden. Zur Behebung dieses Problems ist in [BK02] ein unzufriedenstellender Mechanismus implementiert: Die Jini-Portierung ist durch einen Dienst erweitert, welcher zyklisch die Komponenten kontaktiert und deren Gültigkeitsintervalle verwirft, wenn sie nicht erreichbar sind. Dieser Dienst muss dazu alle Kommunikationsteilnehmer kennen und unterminiert daher das Jini-Kommunikationskonzept.

Jini fehlt es an Mechanismen für die zeitkritische Kommunikation. Für den Austausch des in Jini verwendeten Kommunikationsprotokolls ist die Aufteilung der Schichten in Jini ungeeignet. Daher ist diese Middleware für den Einsatz in verteilten eingebetteten Systemen nicht nutzbar und wird für eine modellübergreifende Kommunikation nicht berücksichtigt.

COSMIC

Die COSMIC-Middleware verwendet das Publish/Subscribe-Konzept und ermöglicht so die transparente Kommunikation in einem verteilten eingebetteten System. COSMIC unterstützt verschiedene Echtzeitklassen (HRT, SRT und NRT) für Ereigniskanäle und sichert die zeitlichen Eigenschaften der Echtzeitereigniskanäle zu. In [KM99] wird die Umsetzung von COSMIC für das CAN-Protokoll beschrieben. Dabei werden die Eigenschaften des CAN-Protokolls, wie z.B. die priorisierte Arbitrierung, sinnvoll ausgenutzt. COSMIC verwendet den CAN 2.0 B Standard mit dem 29 Bit Identifier. Die Nutzung des langen Identifiers verschlechtert zwar das Verhältnis von Nutzlast zu Metainformationen, ermöglicht aber die bessere Strukturierung der Identifiers. Zusätzlich können Informationen aus der Nutzlast in den Identifier ausgelagert werden. Diese effiziente Nutzung des Identifiers ermöglicht es in COSMIC die kompletten 64 Bit Nutzlast mit Nutz-/Anwendungsdaten zu belegen.

Die CAN-Nachricht in COSMIC baut auf das erweiterte CAN-Nachrichtenformat (siehe Unterabschnitt 2.3.1) auf. In der Abbildung 3.4 werden daher nur der Identifier und die Nutzlast des COSMIC-Nachrichtenformates dargestellt. Es ist dabei zu erkennen, dass der CAN-Identifier durch die COSMIC-Middleware in die Felder Priorität, TxNode und Event-Tag auf-

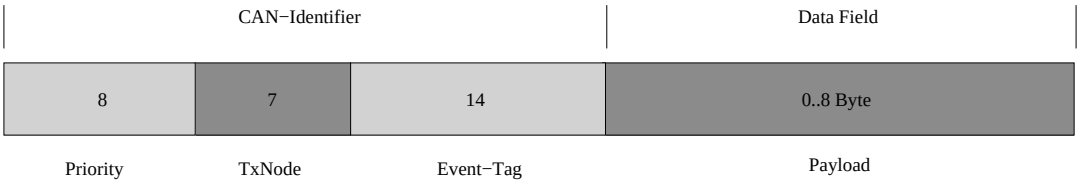


Abbildung 3.4: CAN-Nachricht in COSMIC nach [KB02]

geteilt wird. Das **Prioritäts-Feld** gibt die Priorität der Ereignisnachricht an und aufgrund der Position ist sie entscheidend für die Arbitrierung. Es umfasst 256 verschiedene Prioritätsstufen. Die Prioritätsstufen werden für die Realisierung der verschiedenen Ereigniskanaltypen verwendet. Harten Echtzeitkanälen wird dabei die höchste Priorität (P_{HRT}), weichen Echtzeitkanälen eine mittlere Priorität (P_{SRT}) und Nicht-Echtzeitkanälen eine geringe Priorität (P_{NRT}) zugewiesen. Aufgrund des Arbitrierungsmechanismus werden die höheren Prioritäten durch geringere Werte dargestellt. Die Prioritäten lassen sich also durch die Relation 3.1 darstellen.

$$P_{HRT} < P_{SRT} < P_{NRT} \tag{3.1}$$

Das **TxNode-Feld** dient der eindeutigen Identifizierung jedes Knotens. Mit diesem Feld wird verhindert, dass Knoten, welche den gleichen CAN-Identifier verwenden, einen vom CAN-Protokoll nicht auflösbaren Arbitrierungskonflikt erzeugen und sich gegenseitig blockieren. Die 7 Bit Knoten-ID wird beim Start jedes Knotens durch den ECB zugewiesen. Dazu muss der Knoten seine feste 64 Bit lange global eindeutige Knoten-ID an den ECB übermitteln. Das 14 Bit breite **Event-Tag** (*Etag*) dient der Identifikation der Ereigniskanäle im CAN-Netz. Es wird, wie bereits in Unterabschnitt 2.5.3 dargestellt, unter Vorlage der 64 Bit globalen Ereigniskanal-ID durch den ECB dynamisch zugewiesen.

Die Nachrichten, welche über **harte Echtzeitkanäle** gesendet werden, haben die höchste Priorität und setzen sich bei der Busarbitrierung gegen alle anderen Nachrichten durch. Um sicherzustellen, das die HRT-Kanäle nicht in Konkurrenz auf den Bus zugreifen, werden sie über feste Zeitfenster voneinander getrennt. Hierfür wird wie in zeitgesteuerten Protokollen ein rundenbasiertes TDMA-Schema verwendet. Der Kommunikationsplan für die TDMA-Zeitschlitze muss im Voraus erstellt und an alle Knoten, welche harte Echtzeitkanäle bereitstellen, übergeben werden. Die Reservierung von Zeitschlitzen erfordert eine globale Zeit und die Synchronisation der Knoten. Die Zeitsynchronisation in COSMIC basiert auf dem in [GS94] beschriebenen Verfahren. Mit der expliziten Zuweisung von Zeitfenstern wird zugesichert, dass alle HRT-Ereignisnachrichten zeitlich getrennt sind und so rechtzeitig gesendet werden können.

Im CAN-Protokoll ist die Unterbrechung von Nachrichten, die gerade gesendet werden, nicht erlaubt. Es ist daher möglich, dass zu Beginn des Zeitfensters für eine harte Echtzeitnachricht der Bus bereits belegt ist und die Nachricht verzögert wird. Aus diesem Grund muss das Sendezeitfenster für den schlechtesten Fall, in dem gerade zu Beginn des Fensters die längst mögliche Nachricht den Bus erhalten hat, dimensioniert werden. Sollen in der harten Echtzeitkommunikation Übertragungsfehler toleriert werden, muss das Zeitfenster für den harten Ereigniskanal um die Zeit für die (eventuell mehrmalige) Neuübermittlung der Ereignisnachricht vergrößert werden.

Im Gegensatz zu anderen Protokollen wie TTP/C oder TTCAN ist ein Zeitfenster in COSMIC nur dann belegt, wenn die dazugehörige harte Echtzeitnachricht gesendet wird. Ist die Übermittlung der Nachricht vor Ende des Zeitfensters erfolgreich abgeschlossen, kann der Bus von weichen oder Nicht-Echtzeitkanälen genutzt werden. Dies wird dadurch ermöglicht, dass die harten Echtzeitnachrichten die höchste Priorität besitzen und in ihrem Zeitfenster bei der Arbitrierung definitiv den Bus erhalten. Ist die harte Echtzeitnachricht erfolgreich versendet, kann die höchst priorisierte nicht harte Echtzeitnachricht den Bus durch Arbitrierung erhalten. Eine andere harte Echtzeitnachricht kann den Bus nicht erhalten, da sie nicht innerhalb eines fremden Zeitfensters senden darf. Dadurch schränken große Zeitfenster, die das mehrmalige Neusenden der harten Echtzeitnachricht im Fehlerfall garantieren, im Normalfall[3] die Bandbreite für weiche und Nicht-Echtzeitkanäle nicht ein. Aufgrund der einander ausschließenden Zeitschlitzreservierungen der HRT-Kanäle, wird durch größere Zeitfenster aber die Zahl der möglichen harten Echtzeitkanäle im System verringert.

Durch die prioritätsbasierte Arbitrierung innerhalb der Zeitfenster für harte Echtzeitkanäle, können auch Knoten, welche nicht mit der globalen Zeit synchronisiert werden, an der Kommunikation teilnehmen. Versucht ein solcher Knoten innerhalb eines exklusiven Zeitfensters zu senden, wird er die Arbitrierung gegen die harte Echtzeitnachricht verlieren. Eine Störung der harten Echtzeitkommunikation ist daher durch asynchrone Knoten nicht möglich. Dies gilt aufgrund der Beschränkung, dass asynchrone Knoten keine Nachrichten mit der Priorität von harten Echtzeitnachrichten senden dürfen.

Der Nachteil der Übertragung der harten Echtzeitnachrichten in COSMIC ist, dass die Nachricht zu keinem fest bekannten Zeitpunkt gesendet wird: Nachrichten, die zu Beginn des Zeitfensters den Bus blockieren, können den Beginn der Übermittlung der Echtzeitnachricht verzögern. Durch das wiederholte Senden der HRT-Nachricht im Fehlerfall ist es nicht möglich zu bestimmen, wann genau innerhalb des Zeitfensters die Nachricht korrekt beim Empfänger eintrifft. Es entsteht dadurch ein Jitter in der Übertragungszeit, den es in der Echtzeitkommunikation zu vermeiden gilt. Rein zeitgesteuerte Protokolle beginnen die Nachrichtenübermittlung zu einem definierten Zeitpunkt und senden die Nachricht, um Übertragungsfehler zu tolerieren, immer n-mal. Dies eliminiert den Jitter, allerdings auf Kosten der Netzbelastung. In COSMIC wird der Jitter durch den ECH abgefangen, welcher die Nachricht erst zum Zeitpunkt der Übertragungsdeadline an die Anwendung weiterreicht, unabhängig davon ob die Nachricht bereits früher eingetroffen ist.

Weiche und Nicht-Echtzeitnachrichten werden in COSMIC über die normale CAN-Arbitrierung versendet. Dabei haben **Nicht-Echtzeitnachrichten** eine feste geringe Priorität, so dass sie sich bei der Arbitrierung nicht gegen weiche Echtzeitnachrichten durchsetzen können. Den **weichen Echtzeitkanälen** werden mehrere Prioritäten zugeordnet. Jede weiche Echtzeitnachricht besitzt eine Übertragungsdeadline. Die Priorität für jede Nachricht wird anhand der zeitlichen Entfernung zu dieser Deadline mit Hilfe des EDF-Planungsalgorithmus berechnet [LKJ98]. Da sich die Entfernung zur Deadline zur Laufzeit verringert, muss die Priorität von weichen Echtzeitnachrichten dynamisch erhöht werden. Für jeden Prioritätswert ist ein Prioritätszeitintervall ($\Delta t_{Zeitintervall}$, *priority slot*) der Entfernung zur Übertragungsdeadline zugeordnet. Alle Nachrichten deren Entfernungen zu ihrer Deadline innerhalb eines Intervalls liegen, werden auf einen Prioritätswert abgebildet. Dies führt dazu, dass zwei Nachrichten mit

[3] Der Normalfall bezeichnet die Situation, in der keine mehrfache Übertragung der HRT-Nachricht notwendig ist.

der gleichen Priorität nicht korrekt nach ihrer Entfernung zur Deadline bewertet werden können. Es ist daher möglich, dass eine Nachricht die Arbitrierung gewinnt, welche weiter von ihrer Deadline entfernt ist als eine andere Nachricht gleicher Priorität. Um diesen Effekt zu vermeiden, können die Prioritätszeitintervalle verkleinert werden. Kleine Intervalle führen aber zu einem engen Zeithorizont ($\Delta t_{Horizont}$), welcher durch die Anzahl der Prioritäten und die Länge des Prioritätszeitintervalls ($\Delta t_{Zeitintervall}$) gegeben ist. Die Gleichung 3.2 stellt diesen Zusammenhang dar. Die Anzahl der Prioritäten ($P_{Min} - P_{Max}$) bestimmt sich aus der Differenz zwischen dem niedrigsten (P_{Min}) und dem höchsten Prioritätswert[4] (P_{Max}).

$$\Delta t_{Horizont} = (P_{Min} - P_{Max}) \cdot \Delta t_{Zeitintervall} \tag{3.2}$$

Alle Nachrichten deren Entfernung zur Deadline größer als der Zeithorizont sind, werden auf die niedrigste Priorität (P_{Min}) abgebildet und daher falsch in die Kommunikation eingeplant. Die Definition des Prioritätszeitintervalls und des Zeithorizontes ist daher für die jeweilige Anwendung zu optimieren.

Durch die Umsetzung der Echtzeitkanäle im CAN-Protokoll stellt COSMIC eine sichere Übertragung der HRT-Nachrichten und eine effektive Nutzung der Bandbreite für SRT- und NRT-Nachrichten zur Verfügung. COSMIC setzt damit die Einhaltung der zeitlichen Eigenschaften der Echtzeitkommunikation durch. Die konsistente Adressierung durch den global gültigen Betreff (UID) erlaubt eine Kommunikation zwischen verschiedenen Netzen in einem verteilten System. Aufgrund dieser Eigenschaften wird COSMIC als Middleware für das Gateway verwendet.

3.4 TTP/C-CAN-Gateway mit COSMIC

In diesem Abschnitt wird das Konzept eines TTP/C-CAN-Gateways auf Basis der COSMIC-Middleware beschrieben. Dazu wird COSMIC so erweitert, dass es das TTP/C-Protokoll unterstützt. Es werden Ereigniskanäle eingeführt, welche eine Kommunikation zwischen CAN und TTP/C mit konsistenter Adressierung ermöglichen. Diese Kanäle sind den von COSMIC definierten Echtzeitklassen zugeordnet und setzen die entsprechenden zeitlichen Anforderungen durch. Für die Konvertierung der Ereigniskanäle werden für die Echtzeitklassen verschiedene Möglichkeiten mit unterschiedlichen Eigenschaften dargestellt. Weiterhin wird das Gateway mit Mechanismen zur Kontrolle des Informationsflusses ausgestattet.

Im Folgenden wird das CAN-Protokoll immer in Verbindung mit der COSMIC-Middleware betrachtet. Eine CAN-Nachricht bezeichnet daher eine CAN-Nachricht im COSMIC-Format.

3.4.1 COSMIC-Ereigniskanäle in TTP/C

COSMIC verwendet Ereigniskanäle und erlaubt damit der verteilten Anwendung eine Kommunikation mit den Eigenschaften des Ereigniskanals. Damit ein CAN-Knoten über einen Ereigniskanal mit einem TTP/C-Knoten (und umgekehrt) Nachrichten austauschen kann, muss dieses Middleware-Konzept in die TTP/C-Knoten und das Gateway integriert werden. Für die Nutzung von COSMIC in TTP/C werden Ereigniskanäle und entsprechend formatierte Nachrichten benötigt. Weiterhin bedarf es zum Anmelden bzw. Abmelden von Ereigniskanälen

[4] Es ist zu beachten, dass den höheren Prioritäten geringere Zahlenwerte zugeordnet sind.

entsprechender Kontrollnachrichten. Im Folgenden werden die einzelnen Erweiterungen für die COSMIC-Unterstützung in TTP/C erläutert.

Ereigniskanal

Der Begriff Ereignis wird im weiteren Verlauf der Arbeit als getypte Information gesehen. Das Ereignis steht damit nicht im Zusammenhang mit Ereignisnachrichten oder ereignisgesteuerter Kommunikation. Ein Ereignis kann durch eine Ereignis- oder Statusnachricht beschrieben und durch einen Ereigniskanal publiziert oder konsumiert werden. Aus diesem Grund wird der Begriff Ereigniskanal auch im TTP/C-Netz verwendet. Der Begriff Nachricht wird verwendet, wenn nicht spezifiziert ist, ob es sich um eine Ereignis- oder Statusnachricht handelt.

Im TTP/C-Netz wird ein Ereigniskanal über den global eindeutigen Identifier (UID) referenziert. Der Ereigniskanal hält die zeitlichen Eigenschaften, der durch ihn gesendeten Nachrichten. Zu diesen Eigenschaften gehört die Echtzeitklasse der Nachrichten. Es sind drei verschiedene Echtzeitklassen definiert: harte Echtzeit (HRT), weiche Echtzeit (SRT) und keine Echtzeit (NRT). Weitere Attribute dienen der Beschreibung zeitlicher Anforderungen. Es sind Parameter wie Periode und Gültigkeit der Nachrichten definiert. Als Identifier (UID) wird im gesamten System der aus COSMIC bereits bekannte 64 Bit breite Betreff des Kanals genutzt. Dieser Betreff kann nicht für die Adressierung der Nachrichten im TTP/C-Netz angewendet werden, weil er mit 64 Bit Länge zu viel Overhead in den Nachrichten verursacht.

Ein Ereigniskanal muss der TTP/C-Anwendung verschiedene Dienste anbieten. Dazu gehören die Dienste zur Bekanntmachung eines Ereigniskanal (`announce`) und zum Abonnement eines Ereigniskanal (`subscribe`). Darüberhinaus können Dienste zum Abmelden eines Kanals (`cancel announce`) und zum Zurückziehen eines Abonnements (`cancel subscription`) bereitgestellt werden. Die Übermittlung von Nachrichten durch den Informationsproduzenten wird über das Publizieren in einen Ereigniskanal realisiert (`publish`). Der Konsument benötigt auf der anderen Seite eine Möglichkeit die Nachrichten korrekt aus dem Ereigniskanal zu empfangen. Er muss dazu die Nachricht aktiv lesen (`fetch`). Im TTP/C-Netz sendet somit der Produzent die Nachrichten aktiv an den Ereigniskanal (*information push*) und der Empfänger muss zum Empfangen die Nachricht aus den Ereigniskanal holen (*information pull*). Dies ermöglicht es der TTP/C-Anwendung zu bestimmen, wann und ob eine Nachricht empfangen wird. Ein Ereigniskanal ist immer unidirektional, d.h. ein TTP/C-Knoten kann entweder aus einem Ereigniskanal lesen oder in einen Ereigniskanal schreiben. Für einen Kanal gibt es einen Informationsproduzenten und es können sich mehrere Konsumenten für diesen Ereigniskanal interessieren und Nachrichten daraus erhalten.

Echtzeitkommunikation

In der TTP/C-Kommunikation ist immer bekannt, wann eine Nachricht gesendet und empfangen wird. Zu Verzögerungen von Nachrichten kann es aufgrund des Buszugriffs nicht kommen. Deshalb können alle Ereigniskanäle, die im TTP/C-Netz bereitgestellt werden, als harte Echtzeitkanäle betrachtet werden. Ereigniskanäle, welche durch das Gateway in das TTP/C-Netz hinein geleitet werden, können aufgrund der Charakteristika des CAN-Netzes zu unterschiedlichen Echtzeitklassen gehören. Je nach Echtzeitklasse werden die Nachrichten vom Gateway unterschiedlich behandelt. So können HRT-Nachrichten SRT-Nachrichten und SRT-Nachrichten

NRT-Nachrichten in den Warteschlangen[5] im Gateway verdrängen. Es entscheidet damit die Echtzeitklasse, wie schnell die Nachrichten vom Gateway weitergeleitet werden. Jedem Ereigniskanal muss daher eine Echtzeitklasse zugeordnet werden. Die Zuordnung von Ereigniskanälen hat im TTP/C-Cluster für die Anwendung den Vorteil, dass sie je nach Klasse der Nachricht unterschiedlich auf das Fehlen von Nachrichten reagieren kann.

Stellt die Anwendung fest, dass eine harte Echtzeitnachricht in der aktuellen Periode nicht empfangen wurde, muss sie versuchen in einen sicheren Status (*fail safe*) überzugehen oder eine Grundfunktionalität aufrecht zu erhalten (*fail operational*), um den Schaden für das Systems zu begrenzen. Auf das Fehlen von SRT-Nachrichten kann die Anwendung so reagieren, dass sie eine gewisse Zeit ihre Funktion ohne Eingangssignale weiter ausführt und auf die verspäteten Nachrichten wartet. Werden auch nach einer gewissen Zeit keine neuen Nachrichten empfangen, wird die Anwendung ihre Funktion einstellen und in einen sicheren Zustand übergehen. Erhält die Anwendung abonnierte NRT-Nachrichten nicht, kann sie nicht sicher sein, ob ein Problem vorliegt oder ob die Nachrichten nur durch wichtigere Nachrichten verzögert werden. Da NRT-Nachrichten keinen strengen zeitlichen Bedingungen unterliegen, müssen Verzögerungen der Nachrichten toleriert werden.

Für Ereigniskanäle, die durch das Gateway vom TTP-Netz nach außen geleitet werden können, ist die Zuweisungen von Echtzeitklassen zu den Kanälen notwendig. Das Gateway entscheidet anhand der Echtzeitklasse wie, die in das CAN-Netz zu leitendenen, Nachrichten sortiert werden und mit welcher Priorität sie sich um den CAN-Bus bewerben. Das 8 Bit Prioritätsfeld in einer COSMIC-CAN-Nachricht entscheidet maßgeblich, welche Nachricht auf dem CAN-Bus die Arbitrierung gewinnt. HRT-Nachrichten haben demnach die höchste Priorität und greifen bevorzugt auf den Bus zu. Sie besitzen ein eigenes Sendezeitfenster auf dem CAN-Bus in dem sie gesendet werden. SRT- und NRT-Nachrichten können den Bus belegen, wenn keine HRT-Nachricht an der Arbitrierung teilnimmt. SRT-Nachrichten gewinnen dabei immer gegen NRT-Nachrichten. Die Relation 3.1 zeigt das Prioritätsverhältnis der drei Echtzeitklassen.

Die dynamische Prioritätsberechnung der SRT-Nachrichten wird in COSMIC (siehe Unterabschnitt 3.3.3) nach dem in [LKJ98] beschriebenen Verfahren zur Planung von weichen Echtzeitnachrichten nach dem EDF-Algorithmus durchgeführt. Für die Zuordnung der SRT-Prioritäten wird im CAN- und im TTP/C-Netz dieselbe Berechnungsvorschrift verwendet. Es wird für die Berechnung eine lineare Abbildung der Prioritäten zwischen dem höchsten Prioritätswert (P_{Max}) und dem niedrigsten (P_{Min}) durchgeführt[6]. Die Parameter dieser Abbildung sind das Prioritätszeitintervall ($\Delta t_{Zeitintervall}$), welches die Gültigkeitsdauer für einen Prioritätswert angibt, und der Zeithorizont ($\Delta t_{Horizont}$), welcher den durch die Prioritäten abgedeckten Zeitbereich angibt. Neben den zeitlichen Parametern beeinflusst auch die Anzahl der unterschiedlichen Prioritätswerte die Abbildung. Für die Definition des Zeithorizontes ($\Delta t_{Horizont}$) und der Beschreibung des Zusammenhanges zu den anderen Parametern sei auf Gleichung 3.2 verwiesen. Der höchste Prioritätswert einer SRT-Nachricht wird erreicht, wenn die Nachricht weniger als ein Prioritätsintervall von seiner Deadline ($t_{Deadline}$) entfernt ist. Nachrichten, die sich außerhalb des Zeithorizontes befinden, werden auf die niedrigste Priorität abgebildet. Die Gleichung 3.3 zeigt die Berechnungsvorschrift für die SRT-Prioritäten. Die Zuordnung der Prioritätswerte hat innerhalb des Prioritätszeitintervalls ($\Delta t_{Zeitintervall}$) eine Ungenauigkeit, bei der alle Nachrichten innerhalb des Intervalls auf einen Prioritätswert abge-

[5] Es wird davon ausgegangen, dass eine Warteschlange eine definierte Länge hat.

[6] Es ist wiederum zu beachten, dass den höheren Prioritäten geringere Zahlenwerte zugeordnet sind.

bildet werden. Die Abbildung aller Werte innerhalb eines Intervalls wird durch Abrunden der reellen Zahlenwerte auf den niedrigeren Zahlenwert und somit höheren Prioritätswert erreicht.

$$P_{SRT} = \begin{cases} \left\lfloor \frac{t_{Deadline} - t_{jetzt}}{\Delta t_{Zeitintervall}} \right\rfloor + P_{Max} & für \quad (t_{Deadline} - t_{jetzt}) < \Delta t_{Horizont} \\ P_{Min} & sonst \end{cases} \tag{3.3}$$

Die Unterstützung der Echtzeitklassen im TTP/C ist für die eigentliche Nachrichtenübertragung nicht entscheidend. Die Zuordnung der Klassen zu Ereigniskanälen ermöglicht es aber dem Gateway wichtigere Nachrichten bevorzugt zu behandeln und somit deren Verzögerungen zu minimieren.

Nachrichtenformat

Da die TTP/C-Nachrichten über Ereigniskanäle gesendet werden, müssen sie diesen zugeordnet werden. Diese Zuordnung wird im Weiteren als Bindung bezeichnet. Die Bindung von Ereigniskanal zu Zeitschlitz kann zur Entwicklungszeit erfolgen, da die Kommunikation im TTP/C-Netz statisch ist. Damit sind alle Nachrichten, die in einem bestimmten Zeitschlitz übertragen werden, direkt einem Ereigniskanal zugeordnet. Es werden dabei keine Adressinformationen in den Nachrichten übertragen. Die Bindung von Nachricht zu Ereigniskanal ist im TTP/C-Netz auch dynamisch zur Laufzeit möglich. Damit wird innerhalb jeder Nachricht eine Zuordnung zu einem Ereigniskanal benötigt. In Abschnitt 3.4.2 werden beide Bindungsmöglichkeiten untersucht und verglichen.

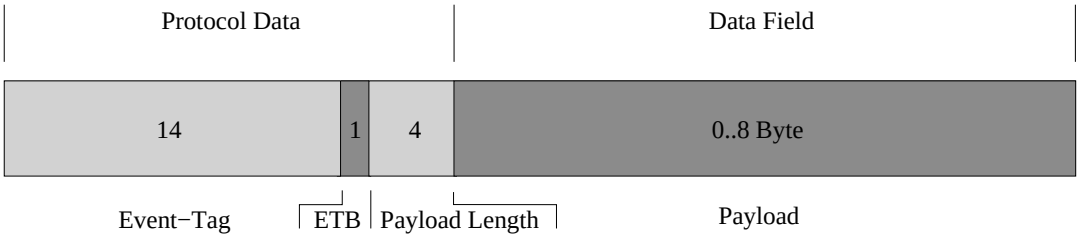


Abbildung 3.5: COSMIC-TTP/C-Nachrichtenformat

Wie in Abbildung 3.5 zu erkennen ist, bestehen das COSMIC-TTP/C-Nachrichtenformat aus vier Teilen. Das **Event-Tag** ist die lokale Netzadresse des Ereigniskanals und ermöglicht damit eine Zuordnung von Nachricht zu Ereigniskanal. Das Event-Tag ist wie bei der Implementierung von COSMIC für das CAN-Protokoll 14 Bit lang und adressiert damit bis zu 16384 Kanäle in einem TTP/C-Netz. Bei der statischen Bindung wird dieses Feld nicht benötigt, da die Zuordnung von Nachricht zu Ereigniskanal bereits fest in der Anwendung enthalten ist.

Das zweite Feld des COSMIC-TTP/C-Nachrichtenformates ist das **Event-Toggle-Bit** (*ETB*). Dieses Bit gibt an, ob der Wert im Anwendungsdatenfeld neu ist oder ob es sich um einen bereits bekannten Wert handelt. Mit der Einführung des ETB wird die korrekte Unterstützung von Ereignisnachrichten im TTP/C-Protokoll erreicht. Damit können Ereignisnachrichten aus dem CAN- ins TTP/C-Netz eingeleitet werden. Eine neue Nachricht wird erkannt, wenn sich das ETB im Vergleich zur zuvor empfangenen Nachricht verändert hat. Hat

sich der Wert nicht geändert, ist die empfangene Nachricht zum wiederholten Male gesendet worden und muss von der Anwendung bzw. dem Gateway nicht beachtet werden.

Das 4 Bit lange **Payload-Length** Feld gibt die Länge der Nutzlast in Byte an. In dem Feld sind Werte von 0 bis 8 erlaubt, damit umfasst eine Nachricht maximal 8 Byte Nutzlast. Diese maximale Nutzlastlänge wurde aufgrund der maximalen CAN-Nutzlastgröße gewählt, so dass das Gateway die Nachrichten bei der Konvertierung nicht fragmentieren muss. Die Beschränkung der Nutzlastlänge ist zwar von TTP/C-Seite aus nicht notwendig, erlaubt aber eine einfache Weiterleitung jedes TTP/C-Ereigniskanals in das CAN-Netz. Je nach Länge der Nutzlast wird der zugehörige Zeitschlitz im TTP/C-Kommunikationsplan angepasst. Sollen mehrere verschiedene Nachrichten nacheinander über einen Zeitschlitz gesendet werden, muss der Zeitschlitz die Länge der größten Nachricht haben.

Der Protokolloverhead durch diese Anpassungen der TTP/C-Nachrichten beträgt 19 Bit bei maximal 64 Bit Nutzlast. Daraus ergibt sich ein Protokolloverhead von mindestens 30%. Der Vorteil der sich mit diesem hohen Anteil an Protokolldaten verbindet, ist die Möglichkeit Informationen netzübergreifend zu adressieren und die TTP/C-Zeitschlitze mit Nachrichten unterschiedlicher Ereigniskanäle dynamisch zu belegen. Zu Beachten ist, dass die COSMIC-TTP/C-Nachricht im Datenfeld des TTP/C-Rahmens (siehe Abbildung 2.5 auf Seite 31) eingebettet ist. Die Nachricht ist damit durch die 16 Bit CRC-Summe des TTP/C-Rahmens gesichert.

Nachrichtentransportkanal

Ein TTP/C-Zeitschlitz in dem n COSMIC-TTP/C-Nachrichten gleichzeitig gesendet werden, wird als Nachrichtentransportkanal (*Message Transport Channel, MTC*) bezeichnet. Der Begriff Zeitschlitz richtet sich hierbei auf die Abbildung im TTP/C-Kommunikationsplan, während der Begriff des MTCs einen Sende- bzw. Empfangspuffer beschreibt. Die Sonderform eines MTC ist der einfache Zeitschlitz, er hat eine Breite von eins. Ein MTC kann entweder statisch oder dynamisch befüllt werden.

- **Exklusiver MTC** *(Exclusive MTC)*:
  Ein exklusiver MTC ordnet den Ereigniskanälen einen festen Platz in diesem Zeitschlitz zu. Damit wird in jeder Periode eine Nachricht aus den zugeordneten Ereigniskanälen gesendet. Wenn der Ereigniskanal keine neuen Nachrichten bereitstellt, wird solange die zuletzt gesendete Nachricht wiederholt bis eine neue Nachricht verfügbar wird. Für einen exklusiven Zeitschlitz gibt es keinen Nachrichtenpuffer, da die Breite des MTC gleich der Anzahl der zugeordneten Kanäle ist. Neue Nachrichten werden an die zugeordnete Position des Ereigniskanals im Zeitschlitz geschrieben. Dabei wird die alte Nachricht von der neuen überschrieben. Wird ein einfacher Zeitschlitz genutzt, kann nur ein Ereigniskanal einem exklusiven Zeitschlitz zugeordnet werden. Dies entspricht der normalen Nutzung der Zeitschlitze im TTP/C-Protokoll.

- **Dynamischer MTC** *(Arbitrary MTC)*:
  Ein dynamischer MTC sendet in jeder Periode die n wichtigsten Nachrichten aus der zugeordneten Warteschlange. Zu übermittelnde Nachrichten werden in die Warteschlange nach ihrer Priorität eingefügt. Die Länge der Warteschlange muss dabei so dimensioniert werden, dass sie bei hohem Nachrichtenaufkommen nicht überläuft. Ist die Warteschlange

leer, wird im dynamischen Zeitschlitz keine Nachricht gesendet. Eine Wiederholung von Nachrichten findet also nicht statt. Im Gegensatz zum exklusiven Zeitschlitz, können einem dynamischer Zeitschlitz mehr Ereigniskanäle zugeordnet werden, als mit einem Mal gesendet werden können. Die Breite des Zeitschlitzes ist daher von der Anzahl der zugeordneten Ereigniskanäle unabhängig.

Ein exklusiver MTC garantiert den zugeordneten Ereigniskanälen eine Übermittlung in jeder Periode. Damit können über die Eigenschaften des exklusiven MTC die zeitlichen Bedingungen des Ereigniskanals durchgesetzt werden. Er wird daher für HRT-Nachrichten eingesetzt. Durch die Wiederholung der Nachrichten im exklusiven MTC können wichtige Ereignisnachrichten zeitredundant gesendet werden. Wird der exklusive MTC von mehreren Ereigniskanälen genutzt, haben die Nachrichten jedes Kanals einen festen zugewiesenen Platz im MTC. Der exklusive MTC benötigt daher einen Zeitschlitz, der genau so groß wie die Länge Nachrichten der zugewiesenen Ereigniskanäle ist.

Der dynamische MTC bildet einen Kommunikationskanal mit fester Bandbreite, in dem sich verschiedene Nachrichten prioritätsbasiert, um die Übertragung bewerben. Sporadische Nachrichten werden durch diesen Kanal effizient unterstützt, da sie nur dann Bandbreite im Kanal belegen, wenn sie übertragen werden. Der dynamische MTC kann damit eine dynamische Kommunikation im TTP/C-Netz ermöglichen. Durch die prioritätsbasierte Warteschlange wird garantiert, dass die wichtigsten Nachrichten bevorzugt übermittelt werden. Die Warteschlange speichert in Phasen mit hohem Nachrichtenaufkommen die eingehenden Nachrichten zwischen, um sie bei geringerer Buslast zu übermitteln. Bei dem dynamischen MTC werden die Nachrichten der Ereigniskanäle nach ihrer Priorität in den Zeitschlitz gelegt. Es gibt dabei keine feste Platzierung der Nachrichten im dynamischen MTC. Da zur Entwicklungszeit nicht bekannt ist, wie der MTC zur Laufzeit befüllt wird, muss dieser Zeitschlitz n mal die maximale Länge einer COSMIC-TTP/C-Nachricht (83 Bit) haben. Der dynamisch befüllte MTC ist von der Nutzung mit den dynamischen Kommunikationsphasen, welche in Abschnitt 3.1 beschrieben wurden, vergleichbar. Der Unterschied zwischen beiden Konzepten zur dynamischen Kommunikation ist, dass der dynamische MTC durch die im TTP/C-Protokoll definierte eindeutige Zugehörigkeit von Zeitschlitz und Task nur durch einen Task befüllt werden kann. Ein dynamischer MTC wird in der Regel nur vom Gateway mit aus dem CAN-Netz eingeleiteten Ereigniskanälen beschrieben, da in der zeitgesteuerten Domäne keine sporadischen Nachrichten und keine dynamische Kommunikation auftritt.

Die TTP/C-Knoten lesen aus den Nachrichtentransportkanälen indem sie über den Ereigniskanal die Nachrichten filtern. Dazu wird über ein Information Pull (`fetch`) auf dem Ereigniskanal die zugehörige Nachricht aus dem MTC gelesen. Enthält der MTC eine Nachricht für den entsprechenden Kanal, wird sie zurückgegeben.

Unterstützung von Ereignis- und Statusnachrichten

Ereignisnachrichten haben die Anforderung, dass sie nur einmal empfangen (*exactly-once*) werden dürfen. Dies bedeutet, dass wenn eine Ereignisnachricht durch einen Fehler zerstört wird, sie erneut gesendet werden muss. Das TTP/C-Protokoll bietet keinen Mechanismus zur Signalisierung eines Übertragungsfehlers an den Sender und auch keine Möglichkeit eine fehlerhafte Nachricht erneut zu übertragen. Es muss also auf einer höheren Ebene sichergestellt werden, das eine Ereignisnachricht nur einmal gelesen wird. Dazu kann eine Fehlersignalisierung und

das wiederholte Senden oberhalb des TTP/C-Protokolls implementiert werden. Diese Signalisierung hat den Nachteil, dass jeder lesende TTP/C-Knoten einen Zeitschlitz mit ausreichend Bandbreite zugewiesen bekommen muss, auf dem er empfangene Nachrichten quittiert oder Fehler mitteilt. Dies kostet Kommunikationsbandbreite und verkompliziert die Anwendung. Des Weiteren kann der Sender, bei einem signalisierten Fehler, die Nachricht nur in dem für diese Nachricht bereitgestellten Zeitschlitz[7] erneut senden. Eine wiederholte Übermittlung einer fehlerhaften Nachricht, greift damit in die Kommunikation des Zeitschlitzes ein und verzögert andere Nachrichten, welche in diesem Zeitschlitz gesendet werden.

Eine andere Möglichkeit Übertragungsfehler zu verhindern ist es die Nachricht redundant zu übermitteln. TTP/C bietet einen solchen Ansatz über redundante Kanäle und Zeitschlitze. Wird eine Ereignisnachricht sooft (n-mal) gesendet, dass sie unter Berücksichtigung der Fehlerannahme ohne Fehler empfangen wird, ist eine Fehlersignalisierung und eine erneute Übermittlung im Fehlerfall nicht notwendig. Die Zahl n beschreibt die Fehlerannahme, dass bei n-maligen Senden die Wahrscheinlichkeit eines Empfangsfehlers in allen Nachrichten (${p_f}^n$) so gering ist, dass sie in der Fehlerannahme akzeptiert wird. Dies lässt sich durch eine mehrmalige Übermittlung in einem Zeitschlitz, redundante Kommunikationskanäle oder eine wiederholte Übertragung über mehrere Perioden im exklusiven MTC erreichen. Anstatt wie bei Fehlersignalisierung die Nachricht erneut zu senden, kann sie immer n-mal gesendet werden. In diesem Ansatz entstehen keine Interferenzen bei der Kommunikation im Zeitschlitz zwischen wiederholten und normalen Nachrichten.

Es muss dabei darauf geachtet werden, dass zwischen zwei Ereignisnachrichten soviel Zeit zur Verfügung steht, um die Nachricht n-mal zu senden. Für einen Ereigniskanal wird dazu ein Platz in einem exklusiven MTC reserviert, der immer von der aktuellen Ereignisnachricht belegt und wiederholt gesendet wird. Tritt eine neue Ereignisnachricht auf, überschreibt sie die alte. Es können zwei Fehler unterschieden werden, die bei der Übermittlung von Ereignisnachrichen auftreten können:

1. Zwei Ereignisse passieren so schnell, dass die neue Ereignisnachricht die alte überschreibt, bevor diese in den Zeitschlitz gelegt und gesendet werden kann. Das frühere Ereignis wird dabei von keinem Abonnenten des Ereigniskanals empfangen, da eine entsprechende Nachricht nie gesendet wird.

2. Zwei Ereignisnachrichten werden in so kurzem Abstand gesendet, dass durch fehlerhafte Übertragung die alte Ereignisnachricht nicht oft genug wiederholt werden kann und somit nicht gelesen wird bevor die neue Nachricht diese überschreibt. Es kann dazu kommen, dass verschiedene Knoten die alte Ereignisnachricht erhalten und andere nicht.

Es muss, um den ersten Fall auszuschließen, sichergestellt werden, dass der minimale Abstand zwischen zwei Ereignissen ($\Delta T_{MinEreignis}$) größer der Periode des dem Ereigniskanal zugehörigen Zeitschlitzes im TTP/C ($\Delta T_{Periode}$) ist. Zur Verhinderung des zweiten Falles muss eine ausreichend häufige redundante Übermittlung (n-mal) der Nachricht zugesichert werden. Dies kann über FTUs (Redundante Kanäle, mehrmaliges Senden pro Zeitschlitz) in TTP/C oder über die mehrmalige Übermittlung im selben exklusiven MTC über mehrere Perioden hinweg erreicht werden. Damit muss eine von der Periode des sendenden Zeitschlitzes abhängende

[7] Es gibt auch die Möglichkeit einen weiteren Zeitschlitz für die wiederholte Übermittlung bei Fehlern anzulegen, dies würde Anwendung sender- wie empfängerseitig jedoch aufwändiger gestalten.

minimale Zeit zwischen zwei Ereignissen eines Ereigniskanals verstreichen, um eine gewisse Fehlersicherheit zu erreichen. Die Formeln 3.4 und 3.5 beschreiben diesen Zusammenhang.

$$n = \left\lfloor \frac{\Delta T_{MinEreignis}}{\Delta T_{Periode}} \right\rfloor \cdot n_K \cdot n_R \tag{3.4}$$

$$p_e = 1 - {p_f}^n \tag{3.5}$$

Dabei sind die Variablen wie folgt definiert: p_e gibt die Wahrscheinlichkeit an, dass die Ereignisnachricht beim Empfänger ohne Fehler ankommt. p_f ist die Wahrscheinlichkeit mit der ein Übertragungsfehler beim Senden auftritt. $\Delta T_{MinEreignis}$ gibt den minimalen Abstand zwischen zwei Ereignisnachrichten desselben Ereigniskanals an. Damit können aperiodische Ereignisnachrichten nicht unterstützt werden. $\Delta T_{Periode}$ beschreibt die Periode des Zeitschlitzes, welcher die Ereignisnachrichten des Ereigniskanals sendet. n_K ist die Anzahl der voneinander unabhängigen[8] physischen Übertragungskanäle. n_R gibt die Anzahl der Nachrichtenwiederholungen innerhalb eines Zeitschlitzes an. Um die gewünschte Fehlersicherheit zu erreichen, müssen die Parameter $\Delta T_{Periode}$, n_K und n_R so gewählt werden, dass n ausreichend groß wird. Wird die Anzahl der Wiederholungen innerhalb des kürzesten Abstand zweier Ereignisse des selben Ereigniskanals (n) groß genug gewählt, entspricht die Wahrscheinlichkeit (p_e), dass die Ereignisnachricht von jedem Empfänger korrekt gelesen wird, der getroffenen Fehlerannahme.

Das in der COSMIC-TTP/C-Nachricht enthaltene ETB sichert zu, dass neue und wiederholte Ereignisnachrichten unterschieden werden können. Denn nur durch die Unterscheidung kann sichergestellt werden, dass eine Ereignisnachricht nur einmal verarbeitet wird. Wird eine neue Ereignisnachricht gesendet, erhält das zugehörige ETB den inversen Wert der vorhergehenden Nachricht. Dadurch kann der Empfänger erkennen, dass eine neue Ereignisnachricht gesendet wurde und sie verarbeiten. Wird hingegen eine Ereignisnachricht empfangen, die den selben ETB Wert, wie die Vorherige besitzt, wird sie ignoriert.

Wird eine Ereignisnachricht, inklusive alle Wiederholungen der Nachricht, aufgrund von Übertragungsstörungen nicht empfangen, trägt die nächste neue Nachricht, den selben ETB-Wert wie die zuletzt empfangene, obwohl es sich um eine neue Nachricht handelt. Damit wird diese Ereignisnachricht nicht als neu angesehen und nicht verarbeitet. Für Ereignisnachrichten gilt daher, dass wenn eine ungerade Anzahl von aufeinanderfolgenden Ereignisnachrichten durch einen Fehler nicht empfangen wird, kann die darauffolgende korrekt empfangene Ereignisnachricht auch nicht gelesen werden. Die Anzahl der verpassten Ereignisnachrichten ($n_{Verpasst}$) ist damit maximal um eins höher als die Anzahl der durch einen Fehler nicht empfangenden Ereignisnachrichten (n_{Fehler}). Mit der Anzahl der fehlerhaften Ereignisnachrichten wird eine ununterbrochene Sequenz von nicht empfangenden Nachrichten, inklusive deren Wiederholungen, bezeichnet. Die Formel 3.6 zeigt diese Abhängigkeit.

$$n_{Verpasst} = \begin{cases} n_{Fehler} + 1 & für \quad n_{Fehler} \ mod \ 2 = 1 \\ n_{Fehler} & für \quad n_{Fehler} \ mod \ 2 = 0 \end{cases} \tag{3.6}$$

Innerhalb der Fehlerannahme ist eine durch Fehler nicht empfangene Ereignisnachricht nicht möglich. Daher kann das Verpassen einer korrekt empfangenen Nachricht durch die Verwendung des ETBs in der Praxis vernachlässigt werden.

[8] Eine Störung des einen Kanals ist unabhängig von der Störung des anderen Kanals

Die Betrachtungen zur Sicherung des Empfanges einer Nachricht durch redundantes Senden und das ETB werden für Statusnachrichten nicht benötigt, da Statusnachrichten in TTP/C-Protokoll bereits unterstützt werden. Die Nachrichten werden in Zeitschlitzen periodisch gesendet und wenn einige Nachrichten nacheinander fehlerhaft empfangen werden, kann dies durch die nächste korrekt empfangene Statusnachricht kompensiert werden.

Kontrollnachrichten

Kontrollnachrichten werden in der Publish/Subscribe-Umgebung für die Bekanntgabe, Abonnement und Kündigung von Ereigniskanälen verwendet. COSMIC kennt innerhalb des CAN-Netzes vier Kontrollnachrichten[9]: zwei davon dienen der Zuweisung einer netzeindeutigen Knotenidentifizierung und zwei sind für die Bindung der UID des Ereigniskanals an das netzeindeutige Event-Tag (Netzadresse des Ereigniskanals) bestimmt. Für das genaue Nachrichtenformat der Kontrollnachrichten sei auf die Implementierungsdokumentation von COSMIC [Bru03] verwiesen. Im Folgenden sind diese vier Kontrollnachrichten für COSMIC-CAN kurz beschrieben.

- **Knotenidentifier anfordern** *(Request Short ID)***:**
  Mit dieser Nachricht sendet ein Knoten umgehend nach dem Start seine 64 Bit lange global eindeutige Knoten-ID an den Event-Channel-Broker (ECB). Die 64 Bit ID wird in acht Teilen mit acht Nachrichten übermittelt. Damit fordert der Knoten eine kurze netzeindeutige ID (TxNode) zur Identifizierung des Knotens an.

- **Bereitstellen des Knotenidentifiers** *(Supply Short ID)***:**
  Zur Bestätigung der acht Teile der *Request Short ID* Nachrichten, beantwortet der ECB jede Nachricht mit einer *Supply Short ID* Nachricht. Der ECB bindet beim kompletten Erhalt aller acht Nachrichten die Knoten-ID an eine 7 Bit lange lokal eindeutige ID (TxNode). Die gebundene TxNode-ID wird mit der letzten Antwortnachricht an den anfordernden Knoten bereitgestellt. Hat der anfragende Knoten den TxNode-Wert erhalten, kann er mit der Kommunikation beginnen und Ereigniskanäle bereitstellen oder abonnieren. Die TxNode-ID wird dabei in das TxNode-Feld jeder ausgehenden Nachricht geschrieben und verhindert durch die eindeutige Zuweisung zum Knoten Arbitrierungsfehler durch gleiche CAN-Identifier.

- **Event-Tag anfordern** *(Request Etag)***:**
  Die Kommunikation über die Ereigniskanäle erfordert die Zuweisung des Event-Tags zur globalen UID des Ereigniskanals. Ist das Event-Tag bekannt, können Nachrichten in den Ereigniskanal publiziert oder daraus empfangen werden. Die Anforderungsnachricht für ein Event-Tag wird vor der Nutzung eines Ereigniskanals an den ECB übermittelt. In der Anforderungsnachricht wird im Datenfeld die 64 Bit lange globale UID des Ereigniskanals zur Bindung angegeben.

- **Event-Tag bereitstellen** *(Supply Etag)***:**
  Hat der ECB eine Anforderungsnachricht nach einem Event-Tag erhalten, prüft er, ob dieser Kanal bereits ein Event-Tag besitzt. Ist der Ereigniskanal noch nicht an ein Event-Tag gebunden, wird ein neues Event-Tag zugewiesen. Die Antwortnachricht mit dem

[9] Es werden in [Bru03] noch zwei weitere Kontrollnachrichten für TCP/IP in COSMIC definiert, welche hier nicht berücksichtigt werden.

zugehörigen Event-Tag wird dann an den aufrufenden Knoten übermittelt. Dieser kann nach dem Erhalt des Event-Tags Nachrichten für diesen Ereigniskanal empfangen oder publizieren.

Wie anhand der oben dargestellten Nachrichten erkennbar ist, wird in COSMIC-CAN auf Netzwerkebene kein direktes `announce` oder `subscribe` realisiert[10]. Dies wird indirekt umgesetzt, indem jeder Knoten der einen Ereigniskanal bedient oder sich dafür interessiert ein Event-Tag für diesen Kanal anfordert. Hat der Knoten dieses Tag erhalten, kann er in diesem Kanal Nachrichten publizieren oder daraus lesen.

Die TTP/C-Kontrollnachrichten für das Management der Ereigniskanäle haben das in Abbildung 3.6 dargestellte Format. Das erste Feld (*Type*) gibt in 3 Bit den **Typ** der Nachricht an. Die Tabelle 3.1 zeigt die verschiedenen Nachrichtentypen und erläutert sie kurz. Das **Betreff-Feld** (*Subject*) enthält die 64 Bit globale UID eines Ereigniskanals. Das **Event-Tag-Feld** beinhaltet die zu dem Ereigniskanal gehörende 14 Bit Netzwerkadresse. Insgesamt ist damit eine Kontrollnachricht 81 Bit breit.

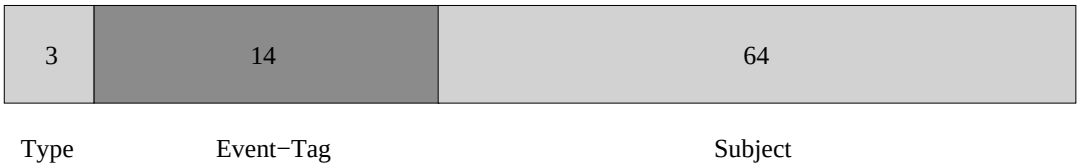


Abbildung 3.6: COSMIC-TTP/C-Kontrollnachrichtenformat

Typ	Typname	Beschreibung
0	*TTPRQ_NONE*	Kennzeichnet einen leeren Zeitschlitz.
1	*TTPRQ_ANNOUNCE*	Ankündigung eines Ereigniskanals.
2	*TTPRQ_SUBSCRIBE*	Abonnement eines Ereigniskanals.
3	*TTPRQ_ETAG_BOUND*	Bestätigt Anforderung Typ 1 & 2.
4	*TTPRQ_CANCEL_SUBSCRIBE*	Zurückziehen eines Abonnements.
5	*TTPRQ_CANCEL_ANNOUNCE*	Abmelden eines Ereigniskanals.
6	*TTPRQ_ETAG_UNBOUND*	Freigabe eines Event-Tags.
7	*TTPRQ_INIT_CLIENTS*	Rekonfiguration der Ereigniskanäle.

Tabelle 3.1: COSMIC-TTP/C-Kontrollnachrichten

Die Kontrollnachrichten werden nur bei der dynamischen Bindung der Ereigniskanäle benötigt. Die statische Bindung verwendet, aufgrund der Bindung zur Entwicklungszeit, diese Nachrichten nicht.

3.4.2 Adressierung der Ereigniskanäle in TTP/C

Die Einführung der Ereigniskanäle in die TTP/C-Anwendung ermöglicht es, die Adressierung von Informationen auf Basis dieser Kanäle zu realisieren. Die Anwendung kann damit einen

[10] Die `announce` oder `subscribe` Anfragen werden auf dem lokalen Knoten von der Anwendung an den ECH gerichtet, welcher mit dem ECB kommuniziert und so die Bindungsinformationen bereitstellt.

Ereigniskanal abonnieren und dazugehörige Nachrichten empfangen oder einen Ereigniskanal bereitstellen und Informationen publizieren. Dazu wird die UID des Ereigniskanals auf eine netzinterne Adresse gelegt. Über die netzinterne Adresse kann der Ereigniskanal auf die zugehörigen Nachrichten zugreifen. Die Zuweisung der netzinternen Adresse kann entweder statisch oder dynamisch erfolgen.

Statische Bindung der Ereigniskanäle

Die statische Bindung weist den Ereigniskanälen die Adressierungsinformationen implizit zu. Einem Ereigniskanal ist dabei fest an ein Zeitschlitz gebunden, in dem nur die zugehörigen Nachrichten übertragen werden. Die dafür notwendigen Informationen finden sich in dem Nachrichtenplan (MEDL), der bereits zur Entwicklungszeit feststeht. Aufgrund der vorher bekannten Kommunikation, werden die in Tabelle 3.1 beschriebenen Kontrollnachrichten zum Abonnement und zur Bereitstellung von Kanälen sowie das Event-Tag und die Nutzlastlänge des COSMIC-TTP/C-Nachrichtenformates (vgl. Abbildung 3.5) bei der statischen Bindung nicht benötigt.

Für Ereigniskanäle, die in das TTP/C-Netz herein- oder herausgeleitet werden, muss das Gateway die Zeitschlitze zu den jeweiligen Kanälen zuordnen. Ereigniskanäle, die in das CAN-Netz geleitet werden, werden dazu aus vorher von der Anwendung festgelegten Zeitschlitzen gelesen, umgewandelt und in das CAN-Netz überführt. Nachrichten aus dem CAN-Netz müssen vom Gateway zu einem Ereigniskanal zugeordnet und im zugehörigen Zeitschlitz weitergeleitet werden. Diese Ereigniskanal-Zeitschlitz-Zuordnung erfolgt über den bei der Anwendungsentwicklung erstellten Nachrichtenplan.

Der Vorteil des statischen Bindens ist, dass Nachrichten nur einen geringen Protokolloverhead (nur das ETB) durch die Einführung der Ereigniskanäle besitzen. Die Informationen zur Adresse und der Länge[11] der Nachricht sind hierbei implizit über den Zeitschlitz bekannt. Die Ereigniskanäle nutzen die native Adressierung des TTP/C-Protokolls über die MEDL. Die Begrenzung dieser Bindung der Ereigniskanäle liegt bei der festen Nutzung eines Zeitschlitzes. Dabei muss ein Zeitschlitz immer gleich verwendet werden, da sonst die Zuordnung der Nachrichten zu den Ereigniskanälen nicht möglich ist. Die statische Bindung unterstützt keine dynamischen MTCs und damit keine effiziente Übermittlung von sporadischen Nachrichten. Dies wird durch erst die dynamische Bindung ermöglicht.

Dynamische Bindung der Ereigniskanäle

Die dynamische Bindung der Ereigniskanäle verwendet zur Zuordnung von Nachrichten zu Ereigniskanälen netzinterne Adressen, die zuvor beschriebenen **Event-Tags**. Das Event-Tag wird in jeder Nachricht (siehe Abbildung 3.5) übertragen und erlaubt deren Zuordnung zum Ereigniskanal. Die Bindung der Event-Tags an die Ereigniskanäle erfolgt mit Hilfe der in Tabelle 3.1 gezeigten Nachrichten. Für die Bindung wird ein Dienst im TTP/C-Netz benötigt, welcher die Zuweisung der Event-Tags übernimmt. Dieser Dienst wird in Anlehnung an COSMIC im CAN-Netz als **Event-Channel-Broker** (*ECB*) bezeichnet. Dieser ECB-Dienst kann im Gateway oder in einem anderen TTP/C-Knoten lokalisiert sein. Im weiteren Verlauf der Arbeit wird der ECB-Dienst für das TTP/C-Netz im Gateway integriert.

[11] Unter der Annahme, dass in einem Ereigniskanal nur gleichlange Nachrichten gesendet werden.

Für die Kommunikation mit dem ECB werden Zeitschlitze für Kontrollnachrichten (*Control Message Slot, CMS*) in allen Knoten des TTP/C-Netzes benötigt. Jeder Knoten muss seine Anfragen an die ECB-Komponente im Gateway in diesem CMS senden und auf Antworten vom Gateway reagieren. Das Gateway besitzt einen CMS zur Mitteilung von Bindungsinformationen und es liest alle Kontrollnachrichten der Netzteilnehmer aus deren CMS. Wie häufig die CMS geplant werden und wieviele Kontrollnachrichten sie aufnehmen können, muss anhand der Anzahl der Ereigniskanäle im TTP/C-Netz definiert werden. Dabei muss zwischen der Schnelligkeit der Übermittlung der Kontrollnachrichten und der durch die CMS genutzte Bandbreite abgewogen werden. In der Regel werden Kontrollnachrichten hauptsächlich beim Starten des Systems oder beim Wechsel eines Zustandes der Anwendung gesendet. Da zu diesen Zeitpunkten ein hohes Kontrollnachrichtenaufkommen herrscht, sollten die CMS ausreichend Bandbreite zugewiesen bekommen. Hat ein CMS wenig Bandbreite entsteht eine Warteschlange für diesen Zeitschlitz und die Kontrollnachrichten werden verzögert gesendet. Um eine Verzögerung der Kontrollnachrichten zu verhindern, kann die Bandbreite der CMS erhöht werden. Dies verringert die Auslastung der CMS und verschwendet in den Phasen, in denen keine Kontrollnachrichten gesendet werden, unnötig Bandbreite des globalen Kommunikationsplans. Eine Lösung dieses Problems ist es, während der normalen Kommunikation den CMS relativ kleine Bandbreiten zuzuweisen und in Phasen, in denen viele Kontrollnachrichten auftreten, den Kommunikationsplan zu wechseln (via *TTP/C Mode Change*). Der andere Kommunikationsplan weist dann den CMS eine hohe Bandbreite zu und die Kontrollnachrichten werden innerhalb von kurzer Zeit gesendet. Stehen keine Kontrollnachrichten mehr an, wird wieder zum normalen Nachrichtenplan zurückgewechselt. Eine andere einfachere Lösung des Problems, ist es die Größe und die Periode der CMS genau abzuwägen, um das beste Verhältnis aus Auslastung der CMS und Bandbreite zu erreichen. Diese Lösung ist für Systeme mit wenigen Ereigniskanälen ausreichend.

Die Verwendung der dynamischen Bindung der Ereigniskanäle erfordert die Definition eines **Bindungsprotokolls**. Dafür werden, die in Tabelle 3.1 beschriebenen, Kontrollnachrichten benötigt. Der Ablauf dieses Protokolls ist wie folgt definiert. Eine Anfrage nach einem Event-Tag für einen Ereigniskanal kann je nach Verwendung des Kanals mit einer Ankündigung (Typ 1, *Announce*) oder einem Abonnement (Typ 2, *Subscribe*) gestellt werden. Dazu wird die entsprechende Kontrollnachricht mit einem leeren Event-Tag-Feld und dem Betreff (UID) des Kanals an das Gateway gesendet. Das Gateway prüft, ob der Kanal bereits gebunden ist. Ist der Kanal noch ungebunden, wird der globalen UID des Ereigniskanals ein Event-Tag zugeordnet. Die Zuordnung von Ereigniskanal zum Event-Tag wird allen Knoten mit einer Kontrollnachricht (Typ 3, *Event-Tag Bound*) mitgeteilt. Beim Empfang weisen die Knoten dem Ereigniskanal das gebundene Event-Tag zu. Damit können sie nun Nachrichten über diesen Kanal publizieren oder empfangen. Kündigt ein Knoten sein Abonnement (Typ 4, *Cancel Subscribe*) wird eine entsprechende Nachricht mit Event-Tag und UID des Kanals an das Gateway gesendet. Damit ist dem Gateway bekannt, wieviele TTP/C-Knoten einen Ereigniskanal abonniert haben. Zieht ein Knoten eine Ankündigung eines Ereigniskanals (Typ 5, *Cancel Announce*) zurück, löscht das Gateway diesen Kanal aus der Menge der gebundenen Kanäle und sendet eine entsprechende Nachricht (Typ 6, *Event-Tag Unbound*) an alle Knoten. Dadurch können die Informationskonsumenten feststellen, wenn der Produzent den Ereigniskanal nicht mehr bedient und keine neuen Nachrichten zu erwarten sind. Eine (Re-)Initialisierungsnachricht (Typ 7, *Init*) vom Gateway veranlasst alle TTP/C-Knoten die Bindungsinformationen zu verwerfen und die Ereigniskanäle erneut zu binden.

Neben der Zuordnung der Ereigniskanäle zu dynamisch vergebenen Event-Tags müssen die Ereigniskanäle durch das TTP/C-Protokoll auch einem Zeitschlitz zugeordnet werden. Dies bedeutet, dass Nachrichten über die Verbindung aus Zeitschlitz und Event-Tag einem Ereigniskanal zugeordnet werden. Zwar ist das Event-Tag im TTP/C-Netz eindeutig, aber wenn eine Nachricht in einem Zeitschlitz gesendet wird, in dem der Ereigniskanal die Nachricht nicht erwartet, kann sie vom Ereigniskanal nicht gelesen und somit nicht zugeordnet werden. Die feste Zuordnung der Zeitschlitze zu den Nachrichten ist durch das TTP/C-Protokoll[12] bedingt.

Der Vorteil der dynamischen Bindung ist, dass mehrere Ereigniskanäle in einem Zeitschlitz bedient werden können, ohne dass bekannt ist, wann welche Nachricht in dem Zeitschlitz gesendet wird. Ein Knoten kann damit auf Anfrage eines anderen Knotens zur Laufzeit weitere Ereigniskanäle in einen dynamischen MTC zur Verfügung stellen, z.B. wenn ein Knoten weitere Debugmeldungen oder zusätzliche Sensorinformationen anfordert. Diese Nutzung eines Zeitschlitzes erhöht dessen Auslastung und schränkt so die verfügbare Bandbreite für die bereits zugeordneten Ereigniskanäle ein. Eine andere Möglichkeit verschiedene Ereigniskanäle dynamisch in einen Zeitschlitz unterzubringen ist der komplette Wechsel der Nutzung eines Zeitschlitzes. Dabei wird das Publizieren eines Ereigniskanals eingestellt, um einen anderen Kanal in diesem Zeitschlitz bereitzustellen. Die Abonnenten des eingestellten Kanals werden vom ECB über die Einstellung des Kanals mit einer Kontrollnachricht (Typ 6, *Event-Tag Unbound*) informiert. Sie können dann bei Interesse den neuen Ereigniskanal abonnieren, der in dem gleichen Zeitschlitz bereitgestellt wird. Zwei Beispiele für eine solche Anwendung sind, dass ein Knoten zur Laufzeit auf Anfrage verschiedene Genauigkeiten von Messwerten bereitstellen kann oder unterschiedliche Informationsquellen wie IR-Sensor und Laserscanner für eine Messgröße anbietet. Dabei müssen die Knoten, welche den Ereigniskanal abonniert haben und den neu bereitgestellten auch abonnieren wollen mit Verzögerungen durch die Neubindung der Ereigniskanäle rechnen.

Die dynamische Bindung erlaubt die Nutzung sporadischer Nachrichten und veränderlicher Kommunikation im TTP/C-Protokoll. Dies ist besonders bei der Einleitung von CAN-Ereigniskanälen nützlich und spart durch die Mehrfachnutzung eines MTCs Bandbreite. Zu beachten ist dabei, dass durch diese Nutzung die Vorhersehbarkeit der Nachrichten in diesen Zeitschlitzen beeinträchtigt wird. Die dynamische Bindung erleichtert auch die Verwendung der Ereigniskanäle in der gesamten Anwendung. Zur Identifikation der Kanäle werden in der TTP/C-Anwendung die UIDs[13] der Ereigniskanäle verwendet. Da diese UIDs auch in der CAN-Anwendung eingesetzt werden, um die Ereigniskanäle zu adressieren, wird eine konsistente Identifizierung der Ereigniskanäle in der gesamten verteilten Anwendung erreicht. Dies erleichtert es dem Entwickler die Informationen in den kommunizierenden Teilanwendungen zuzuordnen und vereinfacht so die Verfolgung des Informationsflusses.

Für die Nachrichten der Ereigniskanäle wird das in Abschnitt 3.4.1 beschriebene Format genutzt. Durch das Nachrichtenformat ergibt sich der Nachteil der dynamischen Bindung. Das Event-Tag und die Übermittlung der Nutzlastlänge erhöhen den Protokolloverhead der Nachrichten auf ein Verhältnis von Protokollinformationen zu Nutzlast von mindestens 30%, siehe dazu Abbildung 3.5 auf Seite 69. Ein weiterer Nachteil der dynamischen Bindung ist die Not-

[12] Andere zeitgesteuerte Protokolle haben eine ähnliche Bindung von Informationsmenge an den Zeitschlitz.

[13] Die Ereigniskanal-UIDs können auch bei der statischen Bindung von der TTP/C-Anwendung genutzt werden. Sie sind aber nicht notwendig und daher nicht direkt mit den Nachrichten verbunden. Es kann daher zu Fehlern in der Zuordnung kommen.

wendigkeit eines Bindungsprotokolls und eines ECBs. Die Versendung der Kontrollnachrichten für die Bindung kostet Bandbreite, da jeder Knoten einen CMS für das Bindungsprotokoll benötigt. Weiterhin nimmt die Verarbeiteung des Bindungsprotokolles in jedem Knoten Prozessorleistung in Anspruch. Auch die Einführung einer ausfallkritischen ECB-Komponente ist als Nachteil der dynamischen Bindung anzusehen. Aufrund der Vorteile der dynamischen Bindung in der Unterstützung von sporadischen Nachrichten und der komfortablen Verwendung der Ereigniskanäle in der Anwendung wird dieser Bindungmechanismus bevorzugt. Im weiteren Verlauf der Arbeit wird die dynamische Bindung für die Beschreibung und die Umsetzung des Konzeptes verwendet.

3.4.3 Komponenten des Gateways

Das Gateway hat die Aufgabe Ereigniskanäle zwischen Netzen aufzuspannen und damit eine Interoperabilität zwischen Knoten aus verschiedenen Netzen zu ermöglichen. Dazu benötigt das Gateway Teilkomponenten, die diese Aufgabe in kleinere Teile aufbrechen. Das Gateway besteht aus zwei funktionalen Teilen, welche die Teilkomponenten gruppieren:

1. **Verwaltung:**
   - Ereigniskanalbindung
   - Filterung
   - Synchronisation
   - Abonnementverwaltung
   - Adressauflösung
2. **Umsetzung:**
   - Adressumsetzung
   - Nachrichtenkonvertierung

Die Verwaltung hat die Aufgabe die Ereigniskanalbindung vorzunehmen, die Filterung der Ereigniskanäle durchzusetzen, die globale Zeit bereitzustellen, die Abonnements in jedem Netz zu organisieren und die Adressen in beiden Netzen aufzulösen. Die Umsetzung teilt sich in die Adressumsetzung und Nachrichtenkonvertierung auf. Bevor auf die einzelnen Komponenten eingegangen wird, soll zunächst das Taskkonzept des Gateways eingeführt werden.

Taskkonzept

Das Gateway muss die zeitgesteuerte Kommunikation des TTP/C-Protokolls wie auch die ereignisgesteuerte Kommunikation des CAN-Protokolls unterstützen. Um die korrekte TTP/C-Kommunikation zu ermöglichen, wird das Gateway als zeitgesteuerte Anwendung definiert. Es ist damit ein Teil des TTP/C-Clusters und alle Funktionen, die im Gateway ablaufen, werden innerhalb von zeitgesteuerten Tasks durchgeführt. Ein zeitgesteuerter Task hat die Eigenschaft, dass er eine feste Start- und Endzeit hat und periodisch aufgerufen wird. Die Periode des Tasks richtet sich nach der Periode der zu empfangenden und zu sendenden Zeitschlitze. Jeder Task wird in eine TDMA-Runde eingeplant, welche analog zum TTP/C-Kommunikationplan die Ausführung der Tasks strukturiert. Durch die Positionierung von Tasks und Zeitschlitzen in einem globalen Zeitplan entstehen Zeitdifferenzen zwischen Zeitschlitzen und Tasks. Besteht eine Zeitdifferenz zwischen einem Zeitschlitz und einem von diesem Zeitschlitz abhängigen Task, kommt es zu einer Verzögerung beim Verarbeiten oder Senden. Diese Zeitdifferenzen können auch als **Phasenverschiebung** von Zeitschlitz zu Tasks bezeichnet werden. Die Phasenver-

schiebung zwischen einem Zeitschlitz und dem Task, der diesen Zeitschlitz liest wird mit φ_1 angegeben. Die Phasenverschiebung zwischen einem Task und einem Zeitschlitz, in dem der Task sendet ist im Folgenden mit der Variable φ_2 belegt. Beide Verzögerungen können aufgrund der Periodizität von Task und Zeitschlitz maximal die Länge des Clusterzyklus annehmen. Durch den statischen Plan sind diese Phasenverschiebungen fest und bereits zur Entwicklungszeit bekannt. Eine optimale Planung kann einen Task so in die TDMA-Runde einordnen, dass er direkt nach den zu lesenden und direkt vor den zu schreibenden Zeitschlitzen geplant wird. Die Zeitdifferenzen werden dann minimiert und sind im Optimum null.

Ereigniskanalbindung

Die Bindung der Ereigniskanäle an eine netzlokale Adresse wird im TTP/C-Netz vom ECB realisiert. Diese Komponente weist neue Event-Tags zu und hält die gebundenen. Das dafür verwendete Bindungsprotokoll ist bereits in Unterabschnitt 3.4.2 beschrieben worden. Die Ereigniskanalbindung ist eigentlich keine Funktionalität des Gateways und daher kann der TTP/C-ECB auch außerhalb des Gateways angeordnet werden. Ist er ECB nicht teil des Gateways, muss es bei der Adressumsetzung mit dem ECB kommunizieren, um fehlendende Adressinformationen für TTP/C-Ereigniskanäle zu erhalten. Die Integration des ECB in das Gateway spart Rechenleistung, Kommunikationsbandbreite und vereinfacht die notwendige Infrastruktur. Daher ist der ECB für das TTP/C-Netz im Gateway angesiedelt.

Synchronisation

Für die Bereitstellung der Echtzeitkommunikation wird eine gemeinsame Zeitbasis (*global time base*) zwischen beiden Netzen benötigt. Die Synchronisation im TTP/C-Protokoll wird in der Spezifikation [TTA03] vorgeschrieben und mit den Fault-Tolerant Average (FTA) Algorithmus nach [KO87] realisiert. In COSMIC werden die CAN-Knoten nach dem in [KBM05, GS94] beschriebenen Verfahren synchron gehalten. Für die Bereitstellung einer globalen gemeinsamen Zeit in beiden Netzen, kann die in COSMIC verwendete Synchronisation mit der TTP/C-Zeit als Referenzzeit (*master clock*) genutzt werden. Es erfolgt dabei kein Eingriff in die Synchronisation der TTP/C-Knoten. Das Gateway wird für die Bereitstellung der globalen Zeit als Zeitmaster des CAN-Netzes eingerichtet und propagiert die TTP/C-Zeit. Dazu muss es periodisch Synchronisationsnachrichten senden. Diese Nachrichten müssen nicht die höchste Priorität haben und können auch verzögert ankommen. Es muss jedoch sichergestellt werden, dass der zeitliche Abstand zwischen den Synchronisationsnachrichten nicht zu groß wird. Der maximal erlaubte Abstand ist von den Uhrendrifts der CAN-Knoten und des Drifts der TTP/C-Zeit abhängig und muss daher je nach Art der verwendeten CAN-Knoten angepasst werden. Das Synchronisationsintervall beschreibt diesen Abstand zwischen den Synchronisationsnachrichten. Je kleiner das Synchronisationsintervall ausgelegt wird, desto genauer folgen die CAN-Knoten der TTP/C-Referenzzeit.

Die Synchronisation der CAN-Knoten mit der TTP/C-Zeit nach [GS94] kann wie folgt erreicht werden. Das Gateway sendet eine Synchronisationsnachricht und speichert direkt nach dem erfolgreichen Senden dieser Nachricht die TTP/C-Zeit. Diese Zeit wird bei der nächsten Synchronisationsnachricht im Datenfeld gesendet. Beim Empfang einer Synchronisationsnachricht speichert der CAN-Knoten die lokale Zeit ab. Dann liest er, die in der Synchronisationsnachricht enthaltende, TTP/C-Sendezeit der letzten Synchronisationsnachricht. Anhand

des Zeitstempels der TTP/C-Sendezeit und des letzten Zeitstempels des CAN-Knotens, sowie den bekannten Verzögerungen durch die Zeitnahme von Gateway und CAN-Knoten, kann der CAN-Knoten einen Korrekturterm für seinen lokalen Zeitgeber berechnen. In COSMIC wird durch dieses Verfahren im CAN-Netz bei einem Synchronisationsintervall von $1s$ ein maximales Offset von $\pm 2,5\mu s$ erreicht [KBM05]. Die Genauigkeit der TTP/C-Uhrensynchronisation wird in [PNK00] mit kleiner als $500ns$ angegeben, dabei hängt die Genauigkeit vom Synchronisationsintervall und der Genauigkeit der Zeitquelle ab. Die unterschiedliche Genauigkeit der Synchronisation in beiden Netzen, muss durch das Einfügen einer zusätzlichen Verzögerung (ε) in der zeitlichen Kopplung von netzübergreifenden Ereigniskanälen kompensiert werden. Das Verfahren aus [GS94] stellt das Gateway mit der TTP/C-Zeit als einen genauen Zeitgeber für die COSMIC-CAN-Knoten bereit. Durch die beschriebene Angleichung der CAN-Knoten an die Zeit im TTP/C-Netz, wird eine hinreichend genaue globale Zeit in beiden Netzen erreicht.

Wird ein CAN-Netz mit mehreren TTP/C-Netzen über verschiedene Gateways verbunden, muss eine globale Synchronisation zwischen den TTP/C-Netzen über das CAN-Netz durchgeführt werden. Dies erfordert eine Anpassung der TTP/C-Zeiten in den TTP/C-Knoten der unterschiedlichen Netze. Es muss daher eine Synchronisation verwendet werden, welche in die Synchronisation des TTP/C-Protokolls der einzelnen Netze eingreift. Die hier beschriebene Synchronisation genügt diesen Ansprüchen nicht, da das Gateway nur als Referenzzeitgeber für die CAN-Knoten agiert und keine direkten Zeitanpassungen im TTP/C-Netz vornehmen kann. Wenn mehrere TTP/C-Netze über ein CAN-Netz verbunden werden, muss das Gateway aber weitere Möglichkeiten der Synchronisierung bereitstellen oder ein komplett anderes Verfahren zur Bereitstellung einer globalen Zeit verwenden. In [KHK+97] wird ein Ansatz zur externen Synchronisation von mehreren TTP/C-Clustern dargestellt. Dieses Konzept bezieht sich auf TTP/C-Netze, welche durch Zeit-Gateways gekoppelt sind. Eine Verbindung von mehreren TTP/C-Netzen durch andere (nicht TTP/C-) Netze wird nicht spezifiziert. Das in [KHK+97] beschriebene Synchronisationskonzept muss, daher für diese Anforderungen angepasst werden. Im Rahmen dieser Arbeit wird kein Konzept für die Synchronisation von mehreren TTP/C-Netzen entwickelt.

Filterung

Eine Filterung der Ereigniskanäle begrenzt den Informationsaustausch zwischen dem TTP/C- und dem CAN-Netz. Die Einschränkung der Weiterleitung der Ereigniskanäle hat den Hintergrund, dass die Weiterleitung aller Ereigniskanäle das Gateway und die Bandbreite der daran angeschlossenen Netze belastet. Zu dem werden nicht alle Ereigniskanäle aus dem einen Netz auch im anderen Netz benötigt. Daher legt die Filterung die **Reichweite** (*Scope*) eines Ereigniskanals fest und entscheidet somit, welche Ereigniskanäle nur im lokalen Netz verfügbar sind und welche durch das Gateway geleitet werden. Aufgrund dessen, dass die gesamte Kommunikation im TTP/C-Netz zur Entwicklungszeit bereits bekannt ist, kann bestimmt werden, welche Informationen im TTP/C-Netz von außen benötigt werden und welche herausgeleitet werden sollen. Es wird dann eine Liste von Ereigniskanälen definiert, welche das Gateway passieren. Ereigniskanäle, die nicht in der Liste stehen, können die Netzgrenze nicht überschreiten. Die Definition der Reichweite der Ereigniskanäle gibt daher dem Entwickler die Möglichkeit den Informationsfluss durch das Gateway bereits zur Entwicklungszeit zu kontrollieren.

Abonnementverwaltung

Die Abonnementverwaltung überwacht in welchem Netz welcher Ereigniskanal abonniert bzw. bereitgestellt wurde. Dadurch kann das Gateway verhindern, dass Ereigniskanäle, die zwar durch das Gateway geleitet werden können, aber nicht abonniert wurden, Bandbreite im Empfängernetz beanspruchen. Eine Weiterleitung von Ereigniskanälen durch das Gateway muss damit erst durch die Abonnementverwaltung freigegeben werden. Die Abonnementverwaltung erlaubt die Weiterleitung nur, wenn in einen Netz ein Knoten diesen Kanal bereitgestellt und im anderen Netz ein Knoten den Ereigniskanal abonniert hat. Dies spart Netzbandbreite, denn ohne die Überwachung der Abonnements würden alle erlaubten Ereigniskanäle durch das Gateway weitergeleitet werden. Die alleinige Weiterleitung von abonnierten Ereigniskanälen ist sinnvoll, wenn z.B. für spätere CAN-Anwendungen bereits Ereigniskanäle aus dem TTP/C-Netz bereitgestellt werden. Diese Kanäle können in die Liste des Filters eingetragen werden. Sie werden aber nur dann durch das Gateway weitergeleitet, wenn sie in der Anwendung wirklich benötigt werden. Das CAN-Netz kann zum späteren Zeitpunkt so erweitert werden, dass es diese Kanäle nutzt. Dies geschieht ohne Anpassung der Kommunikation in der TTP/C-Anwendung oder des Gateways. Diese Erweiterung kann daher zur Laufzeit im CAN-Netz realisiert werden. Durch die Abonnementverwaltung kann die Anwendung im CAN-Netz modular und erweiterbar gestaltet werden. Die Abonnementverwaltung kann aufgrund des ähnlichen Datenbestandes eng mit der Adressumsetzung zusammenarbeiten.

Adressumsetzung

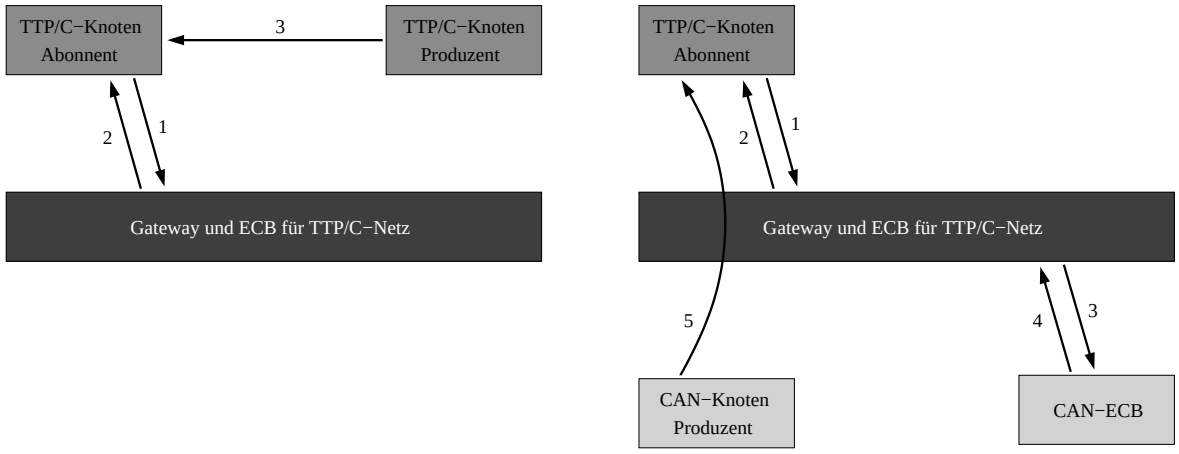
Für die Konvertierung der Nachrichten benötigt das Gateway zur Netzadresse der Eingangsnachricht die Netzadresse der Ausgangsnachricht. Die Adressumsetzung löst genau diese Aufgabe. Sie hat dazu eine Adressumsetzungstabelle (*Lookup Table*) in der die Event-Tags des CAN- und des TTP/C-Netzes, sowie die dazu gehörenden UIDs der Ereigniskanäle, einander zugeordnet sind. Sind für einen Ereigniskanal alle drei Einträge vorhanden, kann die Konvertierung der Nachrichten zu diesem Ereigniskanal durchgeführt werden. Fehlt ein Eintrag in der Adressumsetzungstabelle für die Weiterleitung eines Ereigniskanals, muss die Adressauflösung diesen bereitstellen.

Adressauflösung

Die Adressauflösung hat die Aufgabe die Adressumsetzungstabelle zu komplettieren. Bei einem Abonnement aus einem Netz muss die Adressauflösung, der Adressumsetzung das Event-Tag des Ereigniskanals im anderen Netz zur Verfügung stellen. Dazu muss diese Komponente unter anderem mit dem ECB für das CAN-Netz kommunizieren. Es können für einen Abonnent in TTP/C- und CAN-Netz je zwei Fälle bei der Adressumsetzung unterschieden werden.

Die Abbildung 3.7 beschreibt die Abläufe des Ereigniskanalabonnements eines TTP/C-Knotens. Abonniert ein TTP/C-Knoten einen Ereigniskanal, wird dazu eine Anforderung nach einem Abonnement (***1***, `subscribe`) an die ECB-Komponente im Gateway gesendet. Das Gateway bindet den Ereigniskanal, sofern dieser noch kein Event-Tag hat, und stellt das gebundene Event-Tag mit einer Kontrollnachricht (***2***, `ETag bound`) bereit. Der TTP/C-Konten beginnt, ab dem Erhalt des Event-Tags damit in dem dafür vorgesehenen Zeitschlitz, auf Nachrichten für den abonnierten Ereigniskanal zu warten. (Es wird dazu die `fetch` Funktion des Ereig-

niskanals ausgeführt.) Je nachdem wo der Informationsproduzent des Ereigniskanals zu finden ist, verhält sich das Gateway unterschiedlich. Die Abbildung 3.7 (a) zeigt den Fall, dass das Gateway nicht für die Weiterleitung dieses Ereigniskanals aus dem CAN-Netz zuständig ist, da der Ereigniskanal von einem Knoten im TTP/C-Netz bereitgestellt wird. Das Gateway verhält sich deshalb passiv und vermerkt lediglich, dass der betreffende Ereigniskanal aus dem TTP/C-Netz abonniert wurde. Hat der Produzent aus dem TTP/C-Netz den Ereigniskanals bereits angekündigt und sendet entsprechende Nachrichten, erhält der Abonnent sie auf direktem Wege (*3*). Ansonsten muss der Abonnent solange warten bis der Ereigniskanal vom Produzent bereitgestellt wird.



(a) Bereitsteller im TTP/C-Netz

(b) Bereitsteller im CAN-Netz

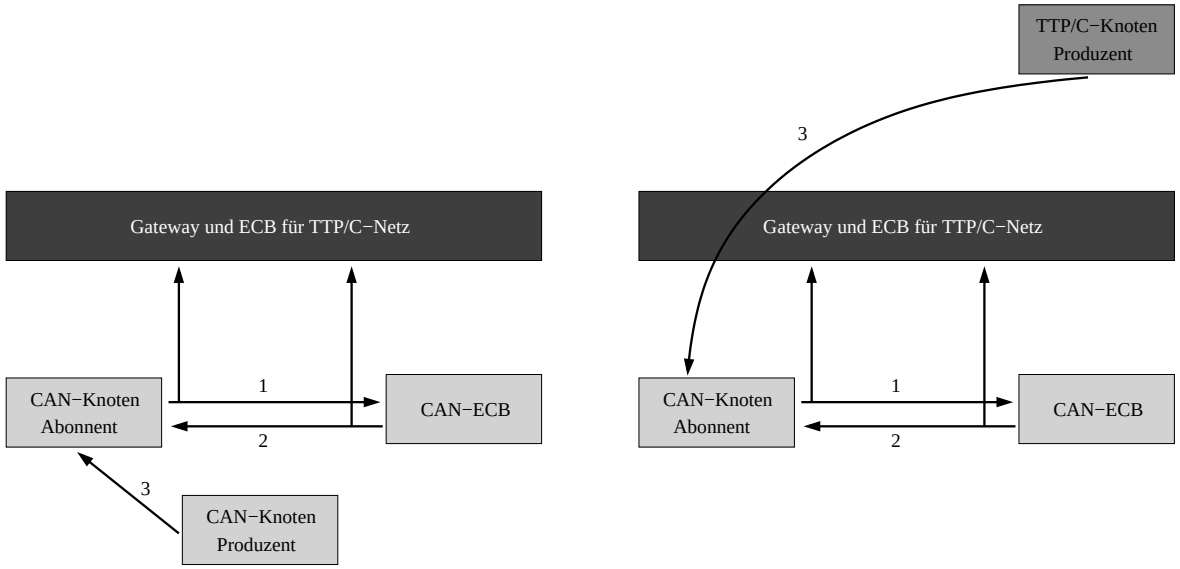
Abbildung 3.7: Abläufe des Ereigniskanalabonnements eines TTP-Knotens

In Abbildung 3.7 (b) wird dargestellt, dass das Gateway für einen Ereigniskanal, der in CAN-Netz produziert wird, die Nachrichtenkonvertierung übernimmt. Handelt es sich bei dem Abonnement des Kanals durch den TTP/C-Knoten, um das erste Abonnement dieses Kanals aus dem TTP/C-Netz, kann es sein, dass das CAN-Event-Tag des Ereigniskanals dem Gateway noch nicht bekannt ist. In diesem Fall sendet das Gateway eine Anforderung nach dem entsprechenden Event-Tag aus (*3*, `request ETag`) und erhält darauf vom CAN-ECB das zugeordnete Event-Tag für das CAN-Netz (*4*, `supply ETag`). Damit hat das Gateway alle notwendigen Adressinformationen und kann die Konvertierung der Nachrichten übernehmen. Ist das CAN-Event-Tag zum Zeitpunkt des Abonnements dem Gateway bereits bekannt, erfolgt die Anfrage an den CAN-ECB (*3* und *4*) nicht. Wenn die Anfrage des Abonnements durch den TTP/C-Knoten die erste Anfrage für diesen Ereigniskanal ist und somit der Ereigniskanal noch nicht ins TTP/C-Netz weitergeleitet wird, beginnt das Gateway mit der Einleitung der Nachrichten des Ereigniskanals (*5*). Wurde der Kanal bereits abonniert, werden die entsprechenden Nachrichten bereits ins TTP/C-Netz geleitet und das Gateway vermerkt lediglich das zusätzliche Abonnement. Dem TTP/C-Knoten, der den Ereigniskanal abonniert hat, ist dabei nicht bekannt woher die Nachrichten kommen, da er in beiden Fällen ab der Zuweisung des Event-Tags im entsprechenden Zeitschlitz auf Nachrichten zu diesem Ereigniskanal wartet. Der einzige Unterschied für den Abonnenten ist, dass wenn das Gateway das CAN-Event-Tag nicht

kennt oder der Kanal noch nicht weitergeleitet wird, der Empfang der Nachrichten um die Zeit für die Bindung des CAN-Event-Tags und den Start der Weiterleitung verzögert wird.

Das Abonnement eines Ereigniskanal von einem CAN-Knoten wird in Abbildung 3.8 graphisch dargestellt. Dem CAN-Knoten soll dabei, unabhängig davon in welchem Netz der Ereigniskanal bereitgestellt wird, ein Abonnement ermöglicht werden. Ein CAN-Knoten, welcher einen Ereigniskanal abonniert, richtet dazu eine Anforderung nach einem Event-Tag an den ECB im CAN-Netz (***1***, ***`request ETag`***). Dazu wird die UID des Kanals in der entsprechenden Kontrollnachricht gesendet. Mit dem Senden einer Anforderung verhält sich ein Abonnent ebenso wie der Bereitsteller des Kanals, der ein Event-Tag anfordert, um dann in diesen Kanal zu publizieren. Es kann also vom Nachrichtenverkehr nicht erkannt werden, ob ein Abonnement oder eine Ankündigung eines Ereigniskanals angestrebt wird. Der CAN-ECB nimmt die Nachricht entgegen, bindet das Event-Tag, sofern es ungebunden ist, und sendet das gebundene Event-Tag an den anfordernden Knoten zurück (***2***, ***`supply ETag`***). Das Gateway hört die Anforderung des Event-Tags mit und entscheidet durch die Filtertabelle anhand der dort enthaltenen UID des Ereigniskanals, wie es mit der Anforderung des CAN-Knotens umgeht. Dabei können drei Fälle unterschieden werden: Der Ereigniskanal wird im TTP/C bereitgestellt, der Ereigniskanal wird in TTP/C abonniert oder der Ereigniskanal ist nicht in der Filtertabelle enthalten. Beim ersten Fall wird ein TTP/C-Ereigniskanal durch die Anforderungsnachricht des CAN-Knotens angefordert. Der zweite Fall beschreibt, dass ein CAN-Knoten einen Ereigniskanal, der auch im TTP/C-Netz abonniert werden kann, bereitstellt oder abonniert. Das Gateway kann bei der empfangenen Anforderung nicht entscheiden, ob ein Abonnement oder eine Bereitstellung des Kanals vorgenommen wird. Unabhängig von der Bedeutung der Nachricht wird das vom CAN-ECB bereitgestellte Event-Tag in der Adressumsetzungstabelle gespeichert, da es beim Abonnement des TTP/C-Knotens benötigt wird. Der dritte Fall tritt ein, wenn das Gateway keinen Eintrag in der Filtertabelle besitzt. Der Ereigniskanal darf dann vom Gateway nicht weitergeleitet werden. Die Abbildung 3.8 (a) stellt die Fälle dar, dass entweder keine Information über den Ereigniskanal im Gateway bekannt ist (dabei entfällt das Abhören der Nachricht (***2***)) oder, dass das Gateway den Ereigniskanal einleiten kann, dabei werden die Bindungsinformationen aus (***2***) in der Adressumsetzungstabelle gespeichert. Das Gateway verhält sich in beiden Fällen passiv. Mit dem Erhalt des Event-Tags vom CAN-ECB, kann der anfordernde CAN-Knoten, die Nachrichten für diesen Ereigniskanal aus dem CAN-Netz herausfiltern und empfangen (***3***).

Die Abbildung 3.8 (b) stellt den Fall dar, dass das Gateway den angeforderten Ereigniskanal für das CAN-Netz bereitstellt. Sofern es noch keinen Abonnenten für diesen Ereigniskanal aus dem CAN-Netz gibt, d.h. die Anforderung des CAN-Knotens ist die erste vom Gateway empfangene Anforderung für diesen Ereigniskanal, muss das Gateway für diesen Kanal das CAN Event-Tag erhalten. Dazu wartet das Gateway auf die Antwort des CAN-ECBs auf die Anforderung des Abonnenten, welche die UID des Ereigniskanals einem Event-Tag zuordnet. Beim Erhalt der dieser Nachricht (***2***, ***`supply ETag`***), werden beide Informationen in die Adressumsetzungstabelle und die Abonnentenverwaltung eingetragen. Für eine Zustellung der Nachrichten an den Abonnenten, benötigt das Gateway nur noch die Adressinformationen des Ereigniskanals im TTP/C-Netz. Da die Kommunikation im TTP/C-Netz statisch ist und alle genutzten Ereigniskanäle zuvor bei der ECB-Komponente im TTP/C-Netz registriert (d.h. bereitgestellt oder abonniert) werden müssen, kann davon ausgegangen werden, dass das Gateway alle Event-Tags von den im TTP/C-Netz genutzten Ereigniskanälen bereits kennt. Liegt das TTP/C-Event-Tag



(a) Bereitsteller im CAN-Netz

(b) Bereitsteller im TTP/C-Netz

Abbildung 3.8: Abläufe des Ereigniskanalabonnements eines CAN-Knotens

dennoch nicht vor, kann es direkt im Gateway von der ECB-Komponente gebunden werden. Daher wird keine weitere Kommunikation zur Bindung des TTP/C-Event-Tags benötigt. Es ist auch möglich, dass wenn das Gateway kein TTP/C-Event-Tag für diesen Kanal besitzt und somit der Kanal nicht genutzt wird, die Anforderung des CAN-Knotens vom Gateway ignoriert wird. Die Weiterleitung der Nachrichten (*3*) durch das Gateway kann direkt nach dem Empfang der Zuweisung des CAN-Event-Tags (*2*, `supply ETag`) beginnen.

Die Adressauflösung für abonnierende TTP/C-Knoten ermöglicht diesen eine transparente Nutzung von Ereigniskanälen, die im TTP/C- und CAN-Netz bereitgestellt werden. Das dargestellte Abonnement eines Ereigniskanals durch einen CAN-Knoten entspricht der Vorgehensweise der COSMIC-Implementierung für das CAN-Protokoll. Das Gateway verhält sich dabei wie ein CAN-Knoten, welcher Ereigniskanäle bereitstellt, diese aber nur ins CAN-Netz leitet, wenn ein Knoten den Kanal abonniert. Es ist dabei für den abonnierenden CAN-Knoten nicht feststellbar, woher die Nachrichten des Ereigniskanals kommen. Die Einhaltung der COSMIC-CAN-Kommunikationskonventionen durch das Gateway, erlaubt es ein TTP/C-Netz mit angeschlossenem Gateway einfach mit einem CAN-Netz zu koppeln. Dies bietet die Möglichkeit eine bestehende CAN-Applikation komponentenweise in ein zeitgesteuertes deterministisches Umfeld zu verlegen und dabei die Konnektivität zwischen den Komponenten zu erhalten. Ein CAN-Teilnetz kann damit einfach durch ein TTP/C-Netz ersetzt werden, ohne dass die verbleibenden CAN-Knoten rekonfiguriert werden müssen. Dabei erlaubt die Adressauflösung zusammen mit dem Adressumsetzungmechanismus des Gateways die transparente Kommunikation zwischen den Knoten beider Netze.

Nachrichtenkonvertierung

Die Nachrichtenkonvertierung setzt die COSMIC-Nachrichten von TTP/C- ins CAN-Format und von CAN- ins TTP/C-Format um. Dazu muss die Nutzlast der Quellnachricht in die Zielnachricht kopiert werden. Die Adressumsetzung stellt unter Angabe des Event-Tags des Quellnetzes mit Hilfe der Adressumsetzungstabelle das Event-Tag des Zielnetzes bereit. Dieses Event-Tag wird in die Zielnachricht zur Identifizierung des Ereigniskanals im Zielnetz eingetragen.

Wird eine Ereignisnachricht vom CAN- in das TTP/C-Format konvertiert, muss das ETB in der TTP/C-Nachricht gesetzt werden. Dabei wird der Wert des ETB aus dem Ereigniskanal gelesen und in die Nachricht geschrieben. Daraufhin wird das ETB im Ereigniskanal gekippt. Ereignisnachrichten, die vom TTP/C- in das CAN-Format konvertiert werden sollen, können nur umgesetzt werden, wenn es sich um neue Nachrichten handelt. Wenn der Wert des ETBs in der TTP/C-Nachricht unterschiedlich zu dem Wert der vorherigen Nachricht des gleichen Ereigniskanals ist, wird die Nachricht konvertiert und gesendet. Ansonsten handelt es sich bei der Nachricht um eine Wiederholung, welche aufgrund der Eigenschaften der Ereignisnachrichten nicht gesendet werden darf. Das ETB ist für Statusnachrichten nicht entscheidend und wird daher ignoriert. Alle Nachrichten müssen priorisiert versendet werden, dabei hängt die Priorität von der Echtzeitklasse des Ereigniskanals ab. Die Priorisierung der Nachrichten wird im folgenden Unterabschnitt dargestellt.

3.4.4 Konvertierung der Ereigniskanäle

Für die Kommunikation zwischen den durch das Gateway verbundenen Netzen, müssen die Ereigniskanäle konvertiert werden. Das Gateway kann Ereigniskanäle in beide Richtungen weiterleiten. Dabei wird die Umsetzung der Kanäle vom Quellnetz ins Zielnetz durch Konvertierungstasks realisiert. Pro Weiterleitungsrichtung sind mehrere Tasks möglich, die einen oder mehrere Ereigniskanäle behandeln. Die Aufgabe der Konvertierungstasks ist es die Quellnachrichten umzusetzen, sie unter Berücksichtigung der Echtzeitanforderungen zu priorisieren und abschließend an das Zielnetz zu übergeben. Die Filterung und die Abonnementverwaltung bestimmen dabei, welche Ereigniskanäle weitergeleitet werden. Für die Bereitstellung der Ereigniskanäle im Zielnetz, muss das Gateway, aufgrund der verschiedenen zeitlichen Anforderungen der Echtzeitklassen und der Berücksichtigung der Eigenschaften von Ereignis- bzw. Statusnachrichten, verschiedene Strategien der Weiterleitung verfolgen. Es bilden sich folgende Schwerpunkte, auf die in diesem Unterabschnitt speziell eingegangen wird:

- Priorisierung der Nachrichten.
- Platzierung der Nachrichten innerhalb der Zeitschlitze im TTP/C-Netz.
- Pufferung der Nachrichten im Gateway.
- Sendeplanung unter Beachtung der zeitlichen Eigenschaften.
- Ermittlung von minimalen und maximalen Verzögerungen durch das Gateway.

Die Überlegungen für die Umsetzung der Ereigniskanäle sind für die jeweilige Richtung der Kommunikation unterschiedlich. Daher werden die Darstellungen für beide Richtungen

getrennt vorgenommen. Zunächst wird die Weiterleitung von CAN nach TTP/C verdeutlicht. Die Konvertierung von Ereigniskanälen aus dem TTP/C-Netz nach CAN wird anschließend erläutert.

CAN nach TTP/C

Die Konvertierung von Ereigniskanälen von CAN nach TTP/C erfordert eine Zuordnung der Ereigniskanäle zu den TTP/C-Zeitschlitzen. Diese Zuordnung muss statisch erfolgen, da auch die Anwendung auf den TTP/C-Knoten statisch geplant ist. Die Anwendung liest dabei nur die Zeitschlitze, welche für sie relevante Informationen enthalten können. Sie muss daher die Zeitschlitze zur Entwicklungszeit kennen, in die das Gateway die entsprechenden Ereigniskanäle aus dem CAN-Netz einleitet. Bei der Konvertierung können mehrere Ereigniskanäle einem Zeitschlitz zugewiesen werden. Damit können z.B. Ereigniskanäle mit ähnlichen Informationen oder zeitlichen Eigenschaften in einem Zeitschlitz abgebildet werden. Die eingeleiteten Informationen lassen sich so gruppieren. Ein Zeitschlitz kann eine oder mehrere Nachrichten in einem MTC auf einmal versenden. Dabei werden zwei verschiedene Verfahren der Zeitschlitznutzung unterschieden. Die Konvertierung mit exklusivem MTC und die Konvertierung mit dynamischem MTC. Für die Begriffe des exklusiven und dynamischen MTC wird auf den Unterabschnitt 3.4.1 verwiesen. Welches Verfahren für die Konvertierung genutzt wird, entscheidet sich anhand der zeitlichen Anforderungen des Ereigniskanals und des Nachrichtentyps. Die Abbildung 3.9 zeigt die verschiedenen Möglichkeiten für die Konvertierung der Ereigniskanäle von CAN nach TTP/C. Weil eine Kombination von HRT-Ereigniskanal mit einem dynamischen MTC nicht die zeitlichen Eigenschaften des HRT-Kanals garantieren kann, wird diese Kombination nicht betrachtet.

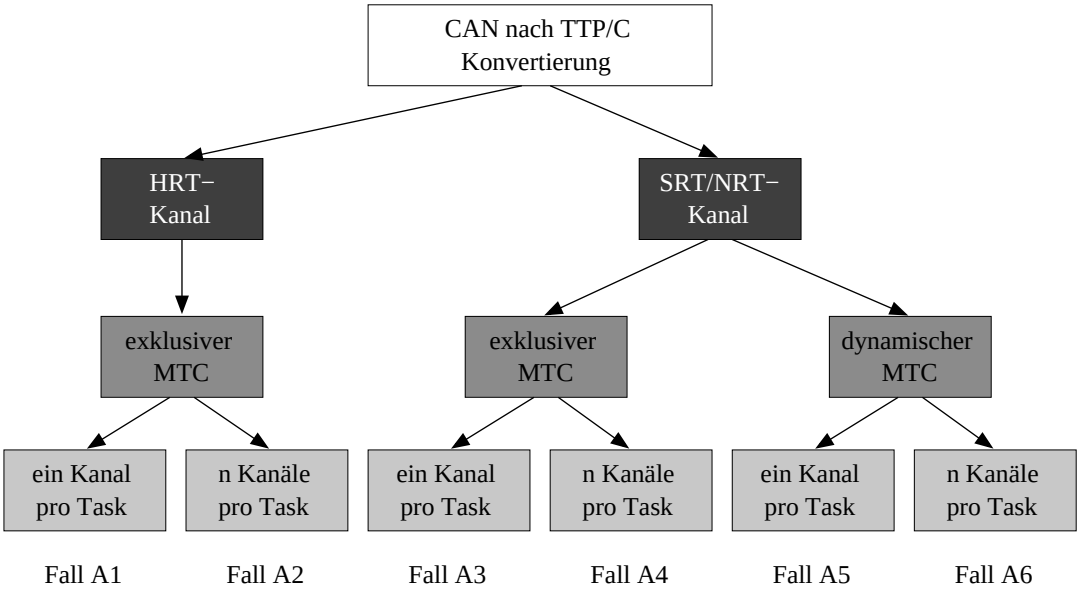


Abbildung 3.9: CAN nach TTP/C Konvertierungsfälle

Konvertierung mit exklusivem MTC

Es werden zuerst die **Fälle A1** und **A2** betrachtet. Die Konvertierung mit exklusivem MTC genügt, aufgrund der Zusicherung der Übermittlung im TTP/C-Netz, den Anforderungen von harten Echtzeitnachrichten. Diese Nachrichten werden im CAN-Netz zu festgelegten Zeiten gesendet und können durch genaue Planung der Kommunikation im TTP/C-Netz direkt nach dem Empfang in den zugehörigen Zeitschlitz abgelegt werden. Damit erhalten die TTP/C-Knoten die Nachricht mit minimaler Verzögerung. Im Allgemeinen ist die Periode der Ereigniskanäle gleich der Periode des Konvertierungstasks. Diese Abstimmung zwischen der Sendeperiode des HRT-Ereigniskanals und dem exklusiven Zeitschlitz in TTP/C erfordert eine Zeitsynchronisation (siehe Unterabschnitt 3.4.3) zwischen beiden Netzen. Sind beide Netze zeitlich synchron, kann eine HRT-Nachricht aus dem CAN-Netz mit minimaler Verzögerung die TTP/C-Knoten erreichen. Die Abbildungen 3.10 und 3.11 stellen die Ende-zu-Ende Verzögerung der Nachrichten dar.

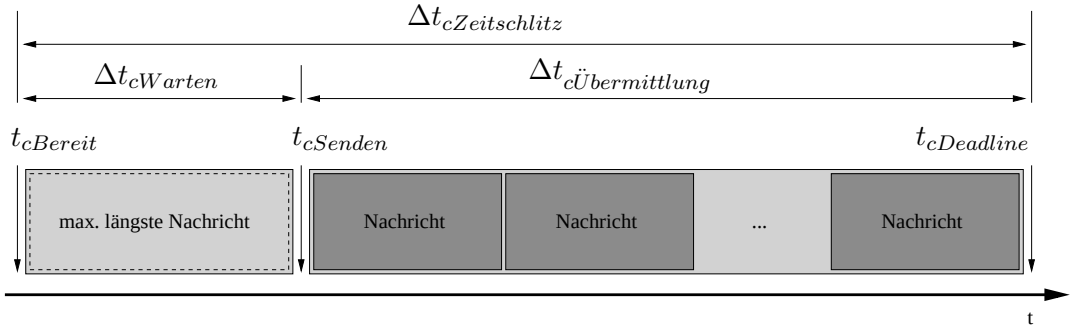


Abbildung 3.10: Aufbau eines CAN-HRT-Zeitschlitzes nach [KBM05]

Die Abbildung 3.10 zeigt den zeitlichen Aufbau des Zeitschlitzes einer CAN-HRT-Nachricht. Der Zeitschlitz ($\Delta t_{cZeitschlitz}$) setzt sich aus zwei Teilen zusammen, dem Warten darauf, dass der Bus frei wird und dem Senden der Nachricht. Die HRT-Nachricht, welche in dem Zeitschlitz gesendet werden soll, muss spätestens zu Beginn des Zeitschlitzes ($t_{cBereit}$) verfügbar sein. Da beim Beginn des Zeitschlitzes, der Bus belegt sein kann und eine Unterbrechung von Nachrichten im CAN-Protokoll nicht zulässig ist, muss mit der Übermittlung eventuell gewartet werden. Wenn ein Knoten zu Beginn des Zeitschlitzes das SOF-Bit seiner Nachricht auf den Bus gelegt hat, muss die komplette Sendedauer der Nachricht abgewartet werden, bevor die HRT-Nachricht gesendet werden kann. Die maximale Wartezeit ($\Delta t_{cWarten}$) beträgt daher die Sendedauer der längst möglichen Nachricht. Die späteste Sendezeit setzt sich aus der spätesten Bereitzeit der Nachricht (also dem Beginn des Zeitschlitzes) und der maximalen Wartezeit zusammen ($t_{cSenden} = t_{cBereit} + \Delta t_{cWarten}$). Wie in Unterabschnitt 3.3.3 beschrieben, werden im Zeitfenster Annahmen über die maximale Anzahl der Nachrichtenwiederholungen getroffen. Daher ergibt sich für die maximale Übermittlungszeit ($\Delta t_{cÜbermittlung}$) die Dauer der mehrmaligen Nachrichtenübermittlung, bei der die Nachricht erst zum letztmöglichen Zeitpunkt korrekt gesendet wird. Der letztmögliche Empfangszeitpunkt der Nachricht ist damit das Ende des Zeitschlitzes ($t_{cDeadline}$). Dieser Zeitpunkt wird auch als Deadline bezeichnet, da ein späteres Empfangen der Nachricht in dieser Periode nicht möglich ist. Um Kommunikationjitter zu verhindern, gibt der ECH des empfangenen CAN-Knotens die HRT-Nachricht erst zu diesem Zeitpunkt an die Anwendung weiter. Auch das Gateway erhält, aufgrund der

Planung des Konvertierungstasks, eine HRT-Nachricht erst zum Ende des CAN-Zeitschlitzes ($t_{tEmpfang} \geq t_{cDeadline}$). Damit beträgt die Verzögerung einer HRT-Nachricht im CAN-Netz die Dauer des Sendezeitfensters ($\Delta t_{cZeitschlitz}$).

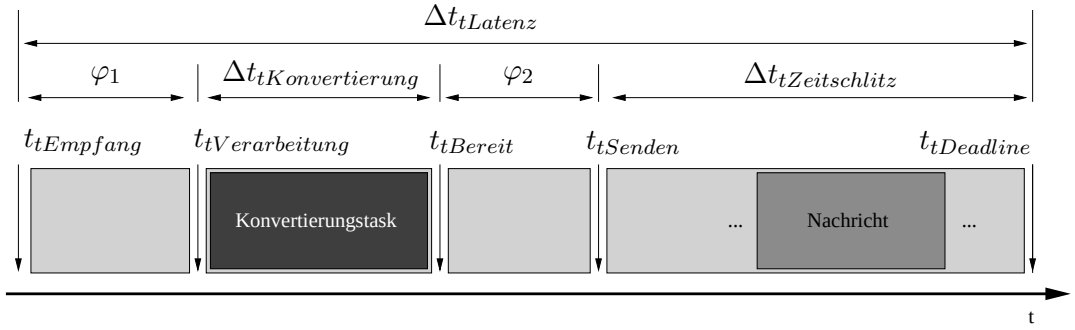


Abbildung 3.11: Latenz von Echtzeitnachrichten durch das Gateway

Die Abbildung 3.11 beschreibt die Verzögerung, welche sich durch die Einleitung einer Nachricht aus dem CAN- in das TTP/C-Netz ergibt. Das Gateway erhält die Nachricht zum Zeitpunkt ($t_{tEmpfang}$) und benötigt ab dann eine gewisse Zeit, um mit der Verarbeitung zu beginnen (φ_1). Diese Wartezeit ist dadurch bestimmt, dass der Konvertierungstask ein zeitgesteuerter Task ist und zu einem festgesetzten Zeitpunkt in der TDMA-Runde ausgeführt wird. Nach dem Eintreffen der Nachricht muss, abhängig von der Planung des Konvertierungstasks, auf den Beginn des Tasks gewartet werden. Die maximale Wartezeit beträgt dabei eine Periode des Konvertierungstasks. Wie im ECH eines CAN-Knotens sichert die Planung des Konvertierungstasks zu, dass die Nachricht erst am Ende des CAN-Zeitschlitzes vom Gateway verarbeitet wird.

Mit dem Beginn des Verarbeitung ($t_{tVerarbeitung}$) wird die Nachricht konvertiert. Am Ende der Konvertierungszeit ($\Delta t_{tKonvertierung}$) ist die Nachricht übersetzt und sendebereit ($t_{tBereit}$). Die Konvertierungszeit hat aufgrund der statischen Taskplanung unabhängig von der Dauer der Ausführung eine feste maximale Länge. Sie ist an die schlechteste Ausführungszeit des Konvertierungstasks (*worst case execution time*, *WCET*) angelehnt. Es ist möglich, dass zum Zeitpunkt, an dem die Nachricht sendebereit ($t_{tBereit}$) wird, der zugehörige Zeitschlitz im TDMA-Schema noch eine gewisse Wartezeit (φ_2) entfernt ist. Die maximale Wartezeit beträgt eine Periode des Zeitschlitzes. Dies tritt auf, wenn beim Bereitwerden der Nachricht der Zeitschlitz gerade angefangen hat und daher die Daten nicht mehr eingefügt werden dürfen. Die Wartezeit (φ_2) ist aufgrund des festen Anfanges und der konstanten Ausführungszeit des Konvertierungstasks fix und bereits zur Entwicklungszeit bekannt.

Beginnt der zugeordnete Zeitschlitz, wird die Nachricht gesendet ($t_{tSenden}$). Die TTP/C-Knoten empfangen die Nachrichten am Ende des Zeitschlitzes ($t_{tDeadline}$), unabhängig davon, wo sich die gewünschte Information im Zeitschlitz befindet. Enthält der Zeitschlitz nur eine Nachricht und ist somit nur genau solang wie die Nachricht, ist der Empfangszeitpunkt gleichzeitig das Nachrichtenende. Die Länge des Zeitschlitzes ($\Delta t_{tZeitschlitz}$) bestimmt damit auch die Latenz der Nachrichtenübermittlung. Ist im CAN die Deadline der HRT-Nachricht auf den Empfangszeitpunkt im CAN gelegt ($t_{cDeadline}$), kommt die Nachricht im TTP/C-Netz durch die zusätzliche Verzögerung erst nach der Deadline an und ist ungültig. Aus diesem Grund muss die Frist der HRT-Nachricht, um die Verzögerungszeit der TTP/C-Kommunikation ($\Delta t_{tLatenz}$),

verlängert werden. Die Latenz ($\Delta t_{tLatenz}$) setzt sich damit aus den Bestandteilen: Warten auf die Verarbeitung (φ_1), Verarbeitungszeit des Gateways ($\Delta t_{tKonvertierung}$), Wartezeit auf den Zeitschlitz (φ_2) und der Länge des Zeitschlitzes zusammen ($\Delta t_{tZeitschlitz}$), wobei sich nur die Wartezeit auf die Verarbeitung (φ_1) bei nicht synchronem Senden der CAN-Nachrichten dynamisch ändert. Alle anderen Größen sind statisch und abhängig von der Implementierung des Gateways oder des Kommunikationsplans und können somit a priori berechnet werden.

Die Wartezeit nach dem Empfang der Nachricht bis zur Verarbeitung durch das Gateway (φ_1) hängt davon ab, wann die CAN-Nachricht gesendet wird. SRT- und NRT-Nachrichten werden zu unbestimmten Zeitpunkten, also asynchron, gesendet. Es kann daher nicht bestimmt werden, wann eine solche Nachricht vom Gateway empfangen wird. Die Wartezeit bis zum Anfang des Konvertierungstasks ist damit veränderlich und zur Entwicklungszeit nicht bestimmbar. Es entsteht dadurch ein *Jitter* von der Größe der Periode des Konvertierungstasks. Für HRT-Nachrichten ist der Empfangszeitpunkt mit dem Ende des periodischen Zeitschlitzes bekannt. Eine Synchronisation der Zeiten im CAN- und TTP/C-Netz erreicht die Kopplung der Empfangszeit einer CAN-HRT-Nachricht ($t_{cDeadline}$) mit dem Beginn der Verarbeitung der eingehenden Nachrichten für den zugehörigen Kanal ($t_{tVerarbeitung}$). Der CAN-Zeitschlitz kann damit zeitlich wie ein TTP/C-Zeitschlitz betrachtet werden. Er besitzt eine feste Phasenverschiebung zum Konvertierungstask, damit ist die Wartezeit (φ_1) fest und im voraus bestimmbar. Der Konvertierungstask besitzt die gleiche Periode wie die Nachricht, da er jede HRT-Nachricht empfangen muss. Durch die effiziente Planung des Konvertierungstasks und des CAN-Zeitschlitzes können das Ende des CAN-Zeitschlitzes und der Beginn der Verarbeitung nahezu gleich gesetzt werden und die Verzögerung (φ_1) wird minimal. Die Gleichung 3.8 zeigt dies.

$$t_{tEmpfang} = t_{cDeadline} \tag{3.7}$$

$$t_{tVerarbeitung} = t_{cDeadline} + \varepsilon \tag{3.8}$$

Die minimale Differenz ε wird aufgrund der Uhrensynchronisation eingeführt, um Ungenauigkeiten in der Synchronität der Zeiten zu puffern. Damit wird verhindert, dass die CAN-Nachricht durch synchronisationsbedingte Jitter nach dem Beginn des Verarbeitungstasks empfangen wird. Durch eine effiziente Planung der Kommunikation und des Gateways kann direkt nach der maximalen Konvertierungszeit ($\Delta t_{tKonvertierung}$) der Sendezeitschlitz beginnen. Damit wird die Wartezeit auf den Zeitschlitz (φ_2) auf null verringert. Es ergibt sich für den **Fall A1** unter Annahme der Synchronisierung der Uhren und Abstimmung der Kommunikation mit dem Gateway die minimale Verzögerungszeit durch das Gateway (ΔT_{min}) nach Gleichung 3.9.

$$\Delta T_{min} = \Delta t_{cZeitschlitz} + \varepsilon + \Delta t_{tKonvertierung} + \Delta t_{tZeitschlitz} \tag{3.9}$$

Werden die Uhren nicht hinreichend synchronisiert oder wird ε zu klein angenommen, kann es sein, dass eine CAN-HRT-Nachricht nicht rechtzeitig vom Gateway empfangen wird und damit die Startzeit der Konvertierung ($t_{tVerarbeitung}$) knapp verpasst. Die Nachricht muss dann eine komplette Periode, bis zum nächsten Start des Konvertierungstasks ($t_{tVerarbeitung} + \Delta t_{Periode}$) warten. Daher muss die Dimensionierung von ε für die Synchronisation sorgfältig vorgenommen werden. Bei synchronisierter Zeit ist der Kommunikationsjitter der Ende-zu-Ende Verzögerung mit ε durch die Uhrensynchronisation gegeben und damit minimal.

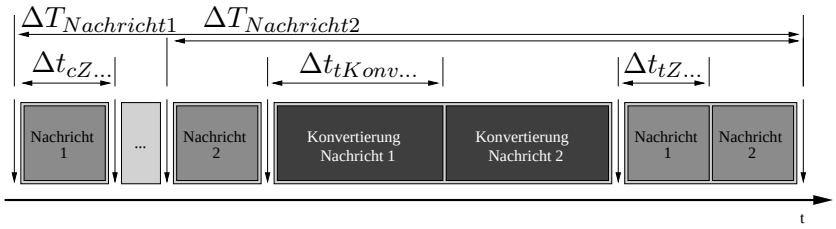
Kann die Kommunikation, z.B. aufgrund eines sehr vollen Plans, nicht optimal geplant werden, können die beiden Verzögerungszeiten, Warten auf Verarbeitung (φ_1) und Warten auf Übermittlung (φ_2), nicht auf null reduziert werden. Da beide Zeiten maximal die Periodenlänge des Konvertierungstasks bzw. des sendenden Zeitschlitzes annehmen können, ergibt sich für den schlechtesten Planungsfall die Gleichung 3.10. Es ist zu beachten, dass die Periode des Konvertierungstasks und die des sendenden Zeitschlitzes identisch sind, da beide voneinander abhängen. Die Gleichung 3.11 beschreibt die Ende-zu-Ende Verzögerung für den Fall, dass die Phasen φ_1 und φ_2 Werte zwischen dem Maximum und dem Minimum annehmen. Damit liegt die Ende-zu-Ende Verzögerung im Bereich der Schranken aus Gleichung 3.9 und 3.10. Die Ende-zu-Ende Übermittlungszeit von CAN-HRT-Nachrichten (**Fall A1**) kann damit im Voraus bestimmt werden.

$$\Delta T_{max} = \Delta t_{cZeitschlitz} + \varepsilon + \Delta t_{tKonvertierung} + \Delta t_{tZeitschlitz} + 2 \cdot \Delta t_{Periode} \quad (3.10)$$

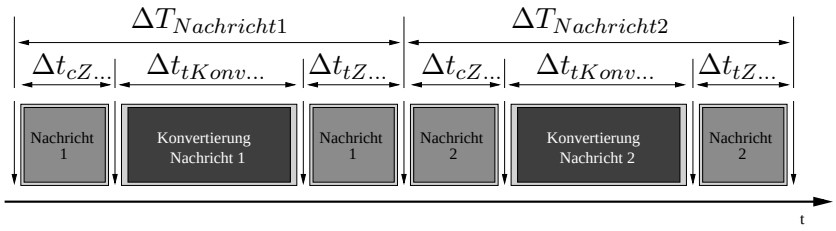
$$\Delta T = \Delta t_{cZeitschlitz} + \varepsilon + \varphi_1 + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz} \quad (3.11)$$

Wie in Abbildung 3.12 (a) dargestellt ist, steigt, durch die Abhängigkeit der Übermittlungszeit von der Länge des TTP/C-Zeitschlitzes ($\Delta t_{tZeitschlitz}$), die Verzögerung der Nachrichten an, wenn mehrere Ereigniskanäle in einem exklusivem MTC (**Fall A2**) gesendet werden. Auch die Verarbeitungszeit des Konvertierungstasks ($\Delta t_{tKonvertierung}$) steigt, da mehrere Nachrichten verarbeitet werden müssen. Die vom Konvertierungstask empfangenen CAN-HRT-Nachrichten werden im CAN-Netz nacheinander gesendet. Durch die sequentielle Übermittlung der Nachrichten erhöht sich die Wartezeit auf die Verarbeitung durch den Konvertierungstask (φ_1), welche abhängig von der Position der CAN-HRT-Nachricht in der Reihenfolge ist. Die Verzögerungszeiten für HRT-Ereigniskanäle, die gemeinsam einen Konvertierungstask verwenden, werden damit im Vergleich zur einfachen Übermittlung um die Übertragungszeiten der anderen Ereignisse des Konvertierungstasks (siehe Abbildung 3.12 (b)) und deren Verarbeitung größer. Eine Aufteilung der HRT-Ereigniskanäle auf einzelne Konvertierungstasks verringert die Übertragungsverzögerung der Nachrichten unter der Annahme, dass die TTP/C-TDMA-Runde genügend Platz für die Verringerung der Phasenverschiebung der Zeitschlitze und Tasks hat. Durch Optimierung der Phasenverschiebungen zwischen CAN-Zeitschlitz, Konvertierungstask und exklusiven Zeitschlitz der Breite eins kann die minimale Verzögerung von ΔT_{min} (aus Gleichung 3.9) erreicht werden. Die Abbildung 3.12 veranschaulicht diesen Zusammenhang der Nutzung von einem HRT-Ereigniskanal oder mehreren HRT-Ereigniskanälen pro Konvertierungstask. Die Variablen sind dabei wie folgt beschrieben: $\Delta t_{cZ...}$ gibt die Länge eine CAN-Zeitschlitzes für einen Ereigniskanal an, $\Delta t_{tZ...}$ definiert die Länge eines einfachen TTP/C-Zeitschlitzes für einen Ereigniskanal, $\Delta t_{tKonv...}$ bezeichnet die Dauer der Konvertierung eines Ereigniskanals.

In den Abbildungen 3.12 sind aufgrund der besseren Darstellung keine zusätzlichen Verzögerungen durch Phasenverschiebungen im globalen Plan dargestellt. Es sind daher die minimalen Verzögerungen der beiden Nachrichten ($\Delta T_{Nachricht1}$ und $\Delta T_{Nachricht2}$) gezeigt. Kann der Plan nicht, so wie es dargestellt wird, optimiert werden, treten die in Abbildung 3.11 beschriebenen Verzögerungen auf. Es ist festzuhalten, dass alle Verzögerungen bei HRT-Nachrichten, durch die Zeitsynchronisation fest und berechenbar sind. Lediglich eine synchronisationsbedingte Schwankung (ε) der Ende-zu-Ende Verzögerung tritt auf.



(a) Mehrere Ereigniskanäle pro Konvertierungstask (Fall A2)



(b) Ein Ereigniskanal pro Konvertierungstask (Fall A1)

Abbildung 3.12: Anordnungen zur Konvertierung von CAN-HRT-Ereigniskanälen

Das Verfahren mit exklusivem Zeitschlitz hat auch für SRT- oder NRT-Nachrichten Vorteile (**Fall A3** und **A4**). So können speziell Ereignisnachrichten, welche auf jeden Fall von den TTP-Knoten gelesen werden müssen, mit diesem Verfahren gesendet werden. Das automatische wiederholte Senden der Nachricht, wenn keine neue Nachricht für den Kanal vorliegt, verringert dabei die Wahrscheinlichkeit des Nichtempfangs einer Nachricht. Wie bereits beschrieben muss anhand der Gleichung 3.5 je nach geforderter maximaler Nachrichtenfehlerrate, das Verhältnis zwischen minimaler Periode der Ereignisnachrichten und Periode des exklusiven Zeitschlitzes gewählt werden. Durch die Einführung des ETB wird bei der Wiederholung der Nachrichten das mehrmalige Ausführen eines Ereignisses verhindert. Da SRT- und NRT-Nachrichten jederzeit auftreten können, ist die Zeit zwischen dem Lesen der CAN-Nachricht und dem Verarbeiten durch den Konvertierungstask unbestimmt und sie kann auch nicht im Voraus definiert werden. Die Gleichung 3.12 zeigt die Abhängigkeit der Verzögerung von der Phase der eintreffenden Nachricht (φ_1). Daher kann für die Ende-zu-Ende Latenz im **Fall A3** eine minimale und eine maximale Verzögerung angegeben werden. Die minimale Verzögerung (ΔT_{min}) ist in Gleichung 3.13 dargestellt und beschreibt den Fall, dass eine CAN-Nachricht genau zum Konvertierungszeitpunkt eintrifft. Die Gleichung 3.14 gibt die obere Schranke (ΔT_{max}) der Verzögerung an, bei der die Nachrichten gerade die Konvertierung verpassen. Die Wartezeit auf das Senden (φ_2) kann im Voraus bestimmt und durch Optimierung minimiert werden. Für die Wartezeit auf die Verarbeitung des Konvertierungstasks (φ_1) wird, durch die beliebige Ankunftszeit der Nachrichten in der Periode, für Gleichung 3.13 null ($\varphi_1 = 0$) und für Gleichung 3.14 der maximale Wert der Periodendauer ($\varphi_1 = \Delta t_{Periode}$) angenommen.

$$\Delta T = \Delta t_{CAN} + \varphi_1 + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz} \quad (3.12)$$

$$\Delta T_{min} = \Delta t_{CAN} + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz} \quad (3.13)$$

$$\Delta T_{max} = \Delta t_{CAN} + \Delta t_{Periode} + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz} \quad (3.14)$$

Es entsteht damit ein Jitter von der maximalen Größe der Periode ($\Delta t_{Periode}$) des Konvertierungstasks bzw. des sendenden Zeitschlitzes (beide Perioden sind aufgrund der Abhängigkeit von Task und Zeitschlitz identisch). Dieser Jitter wird durch die Abbildung der asynchronen CAN-Nachrichten auf den periodischen Kommunikationsplan in TTP/C verursacht.

Die Sendezeit der SRT- oder NRT-Nachrichten im CAN-Netz ($\Delta t_{CAN} = \Delta t_{cBlockierung} + \Delta t_{cArbitrierung} + \Delta t_{cNachricht}$) kann nicht bestimmt werden, da sie im CAN-Netz nicht die höchste Priorität haben und dadurch bei der Arbitrierung beliebig oft gegen Nachrichten mit höherer Priorität verlieren ($\Delta t_{cArbitrierung}$) können. Zusätzlich dazu kann zur Sendezeit der CAN-Bus durch eine Nachricht blockiert sein und die Nachricht wird vor der Arbitrierung verzögert ($\Delta t_{cBlockierung}$). Die Verzögerung durch die Blockierung geht in die Arbitrierungsverzögerung nicht mit ein, da sie vor der Arbitrierung auftritt. Beide nicht vorhersagbaren Verzögerungen müssen zum Jitter der Ende-zu-Ende Verzögerung hinzugefügt werden. Daher ist nur eine Aussage für die Gesamtverzögerung der Nachrichten möglich, wenn für die Sendezeit im CAN-Netz Aussagen getroffen werden können. Die Übermittlungszeit der CAN-Nachricht ($\Delta t_{cNachricht}$) ist durch die Bandbreite des CAN-Busses bestimmt und damit zur Entwicklungszeit bekannt.

Für den **Fall A4**, wenn mehrere Ereigniskanäle von einem Konvertierungstask bearbeitet werden, gelten auch die Gleichungen 3.12, 3.13 und 3.14 für die Ende-zu-Ende Verzögerung. Es ergeben sich keine weiteren Verzögerungen durch den Nachrichtenempfang, da die Nachrichten zu beliebigen Zeiten eintreffen können. Es ist zu beachten, dass die Verarbeitungszeit des Konvertierungstask sowie die Länge des sendenden MTCs mit der Anzahl der verarbeiteten Kanäle ansteigt.

Ereigniskanäle werden im exklusiven Zeitschlitz nicht gepuffert. Sie werden sofort gesendet, wenn sie ankommen. Es ist daher nicht möglich, dass eine Nachricht ihre Frist verletzt, wenn sie das erste Mal im exklusiven Slot gesendet wird. Es wird hierbei zu Grunde gelegt, dass HRT-Nachrichten immer rechtzeitig ankommen und verarbeitet werden. SRT-Nachrichten werden auch immer rechtzeitig vom Gateway empfangen, aber durch die Verzögerung durch die Wartezeiten, Verarbeitung und das Senden (φ_1, $\Delta t_{tKonvertierung}$, $\Delta t_{tZeitschlitz}$, φ_2), kann es dazu kommen, dass die Nachrichten die TTP/C-Knoten verspätet erreichen. Die Fristüberschreitung der SRT-Nachrichten wird aufgrund der weichen Echtzeitanforderung toleriert. Nicht tolerierbare Überschreitungen treten erst bei der Wiederholung der Übermittlung auf. Bei der Wiederholung der Nachrichten im exklusiven Zeitschlitz können die Nachrichten ihre Fristen überschreiten, aber dennoch an die TTP/C-Knoten gesendet werden. Die Fristen sollten so eingestellt sein, dass TTP/C-Knoten die Nachrichten vor ihrem Ablauf sicher empfangen haben. Es muss für Ereignisnachrichten durch die Dimensionierung der Fristen oder durch die Task- und Zeitschlitzplanung sichergestellt werden, dass sie innerhalb der Frist, sooft wiederholt werden, wie es die Fehlerannahme vorsieht. Die Anzahl der Wiederholungen in der Fehlerannahme wird nach Gleichung 3.5 eingestellt.

Konvertierung mit dynamischem MTC

Genügen für die eingeleiteten CAN-Ereigniskanäle die Sicherheiten, welche für das einmalige Senden im TTP/C gegeben sind, kann die Konvertierung mit dynamischen MTC (**Fall A5** und **A6**) verwendet werden. Das Verfahren nutzt für die Zwischenspeicherung der Nachrichten eine Warteschlange pro dynamischen MTC. Wie beim Verfahren mit exklusiven MTC werden einem MTC ein oder mehrere Ereigniskanäle zugeordnet. Dabei entfällt jedoch die Beschränkung, dass die Anzahl der Ereigniskanäle so groß sein muss, dass für jeden Ereigniskanal eine Nachricht auf einmal gesendet wird. Es können bei der dynamischen Verwendung des Zeitschlitzes viele verschiedene Ereigniskanäle unterschiedlichster Periode bzw. Häufigkeit einem MTC zugewiesen werden, da die Warteschlange kurzzeitiges erhöhtes Kommunikationsaufkommen abpuffert und die Nachrichten priorisiert.

Darf ein Ereigniskanal aus dem CAN-Netz das Gateway passieren und existiert ein Abonnement aus dem TTP/C-Netz, werden alle Nachrichten mit Hilfe der Adressumsetzung und der Nachrichtenkonvertierung umgewandelt und in die Warteschlange eingefügt. Dabei können neue Statusnachrichten beim Einsortieren in die Warteschlange die Statuswerte der bereits gepufferten Nachrichten des gleichen Ereigniskanals auf den neuen Wert aktualisieren. Ereignisnachrichen werden im Gegensatz dazu einfach an die Wartschlange angehängt. Ist die Warteschlange voll, können wichtigere Nachrichten, weniger wichtige aus der Liste verdrängen. Dabei haben HRT-Nachrichten[14] die höchste Priorität, gefolgt von der dynamischen Priorität der SRT-Nachrichten. Nicht-Echtzeitnachrichten haben die geringste Priorität. Durch die dynamische Priorisierung der SRT-Nachrichten nach dem EDF-Verfahren, müssen in jeder Periode bevor der dynamische Zeitschlitz gesendet wird, die Prioritäten in der Warteschlange berechnet werden.

Die Konvertierung von SRT-Nachrichten erfordert die Bestimmung der zeitlichen Eigenschaften der Nachricht. Dazu muss das Gateway anhand der Priorität und der Charakteristika des zugehörigen Ereigniskanals die Deadline der Nachricht ($t_{Deadline}$) berechnen. Für die Berechnung muss die Abbildung der Deadline auf die Priorität, nach Gleichung 3.3, rückgängig gemacht werden. Dazu wird die Anzahl der Prioritätswerte berechnet, welche noch nicht erreicht sind, und diese Zahl wird dann mit der Dauer des Prioritätszeitintervalls ($\Delta t_{Zeitintervall}$) multipliziert. Damit ergibt sich die Zeitdauer, welche vom Zeitpunkt des Empfanges durch das Gateway noch bis zur Deadline verfügbar ist. Wird die aktuelle Zeit addiert ergibt sich die Deadline der Nachricht. Es ist dabei zu beachten, dass die CAN-Nachricht durch die Übertragung verzögert ist. Die Übermittlungszeit ($\Delta t_{cNachricht}$) der Nachricht im CAN-Netz muss von der Deadline abgezogen werden. Die Verzögerungen, welche sich durch das Warten auf die erfolgreiche Arbitrierung ergeben, werden nicht mit eingerechnet, da sich die Priorität der Nachricht bis zum tatsächlichen Sendetermin ändert. Die Gleichung 3.15 beschreibt die Berechnung der Deadline.

$$t_{tDeadline} = t_{jetzt} + (P_{Nachricht} - P_{Max}) \cdot \Delta t_{Zeitintervall} - \Delta t_{cNachricht} \tag{3.15}$$

Durch die Rückrechnung der Abbildungsfunktion (siehe Gleichung 3.3) und die dabei erfolgende Abrundung der Prioritäten bei der Berechnung der Nachrichtenpriorität, kommt es zu

[14] HRT-Ereigniskanäle werden normalerweise nicht über die dynamischen MTCs weitergeleitet, da diese nicht die zeitlichen Eigenschaften des Ereigniskanals garantieren können.

einem Vorziehen der Deadline. Damit wird jeder Priorität die dicht möglichste Deadline zugeordnet, obwohl sich die tatsächliche Deadline im Intervall $[t_{tDeadline}, t_{tDeadline} + \Delta t_{Zeitintervall})$ befindet. Statistisch gesehen kann der Mittelwert dieses Intervalls als Deadline $(t_{tDeadline} + \frac{\Delta t_{Zeitintervall}}{2})$ angenommen werden, weil eine Gleichverteilung der Deadlines in diesem Intervall anzunehmen ist. Aufgrund dessen, dass es sich um Echtzeitnachrichten handelt, wird für die Deadline jeder Nachricht der schlechteste Fall angenommen und somit die Deadline vorgezogen. Das Vorziehen der Deadline stellt sicher, dass die Information vom Empfänger nicht fälschlicherweise als gültig anerkannt wird. Der Nachteil dieser Berechnung der Deadline ist, dass Nachrichten verfrüht für ungültig erklärt und somit verworfen werden. Diese Ungenauigkeiten umfassen dabei die Größe des Prioritätszeitintervalls. Eine Nachricht mit höchster SRT-Priorität erhält als Deadline den aktuellen Zeitpunkt, obwohl die tatsächliche Deadline der Nachricht zwischen $[t_{jetzt}, t_{jetzt} + \Delta t_{Zeitintervall})$ liegt. Eine SRT-Nachricht höchster Priorität muss also umgehend gesendet werden, um nicht ungültig zu werden. Bei der Berechnung der Deadlines der SRT-Nachrichten ist zu beachten, dass die Parameter ($\Delta t_{Zeitintervall}$ und $\Delta t_{Horizont}$) der Prioritätsberechnung im CAN-Netz und im Gateway identisch sind.

Wurde die Deadline errechnet, erfolgt die Übergabe der Nachricht an die Warteschlange, wo sie anhand der Restgültigkeit positioniert wird. Dabei verändert sich die Position der Nachrichten in der Warteschlange nach dem Einfügen nicht mehr. Um die dynamischen Prioritäten der zu sendenden Nachrichten zu bestimmen, muss vor dem Senden der Nachrichten in der Warteschlange die Berechnung der Prioritäten nach Gleichung 3.3 durchgeführt werden. Wird bei der Berechnung festgestellt, dass die Deadline einer Nachricht überschritten ist, muss die Nachricht verworfen werden.

Die Gleichung 3.16 beschreibt die Ende-zu-Ende Latenz für die **Fälle A5** und **A6**. Die Zeit für den den Aufenthalt in der Warteschlange ($\Delta t_{tWarteschlange}$) ist dabei für Nachrichten, welche nicht die höchste Priorität haben, unbestimmbar. Die minimale Latenz ist in Gleichung 3.17 beschrieben, dabei wird die betrachtete Nachricht nicht durch die Warteschlange verzögert ($\Delta t_{tWarteschlange} = 0$) und die Wartezeit auf die Verarbeitung ist minimal ($\varphi_1 = 0$). Diese Gleichung ist daher identisch mit der Gleichung 3.13. Die Beschränkung der Verzögerungszeiten beim Verfahren mit dem dynamischen Zeitschlitz, ist nur für die Nachricht mit der höchsten Priorität im dynamischen MTC möglich. Hat die Nachricht nicht die höchste Priorität, kann die Nachricht beliebig oft durch wichtigere Nachrichten in der Warteschlange zurückgestellt werden. Es ist dadurch möglich, dass eine Nachricht, deren Priorität nicht die höchste im dynamischen MTC ist, niemals gesendet wird oder, bevor sie gesendet werden kann, ungültig (bei SRT-Nachrichten) wird. Es ist daher für die **Fälle A5** und **A6** keine maximale Latenz (ΔT_{max}) angegeben.

$$\Delta T = \Delta t_{CAN} + \varphi_1 + \Delta t_{tKonvertierung} + \Delta t_{tWarteschlange} + \varphi_2 + \Delta t_{tZeitschlitz} \quad (3.16)$$

$$\Delta T_{min} = \Delta t_{CAN} + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz} \quad (3.17)$$

Der Kommunikationsjitter der Ende-zu-Ende Verzögerung basiert auf dem Jitter der **Fälle A3** und **A4**, wobei durch die Verwendung des dynamischen MTCs die nichtbestimmbare Zeit der Nachricht in der Warteschlange ($\Delta t_{tWarteschlange}$) hinzugefügt werden muss. Für den **Fall A6** ergibt sich im Vergleich zum **Fall A5** durch die mehrfache Nutzung eines Konvertierungstasks eine längere Verzögerung für die Verarbeitung ($\Delta t_{tKonvertierung}$) und, bei der Verwendung

von dynamischen MTCs der Länge größer eins, eine weitere Verzögerung durch die Dauer des Zeitschlitzes ($\Delta t_{tZeitschlitz}$).

Wenn die höchstpriore Nachricht für diesen dynamischen Zeitschlitz angenommen wird, wird sie am Anfang der Warteschlange einsortiert und umgehend gesendet. Damit gelten die Gleichungen 3.12, 3.13 und 3.14 für die Betrachtung der Verzögerungszeiten. Da die CAN-Nachrichten zu beliebigen Zeiten vom Gateway empfangen werden, entsteht auch bei der Konvertierung mit dynamischem MTC unabhängig von der Priorität der Nachricht mindestens ein Kommunikationsjitter von der Periode des Konvertierungstasks.

Die Konvertierung mit dynamischem Zeitschlitz kann, aufgrund der dynamischen Warteschlange, kein vorhersehbares Kommunikationsverhalten und somit keine Garantien zusichern. Die Kommunikation folgt innerhalb des Zeitschlitzes vielmehr der Kommunikation auf einem prioritätsbasierten Medium auf dem die n wichtigsten Nachrichten pro Periode gesendet werden. Aus diesem Grund eignet sich das Verfahren zur Konvertierung von CAN-Ereigniskanälen nach TTP/C mit dynamischem Zeitschlitz besonders für sporadische und nicht sicherheitskritische Nachrichten, welche nach *Best-Effort* übertragen werden sollen. Werden vom Gateway mehr CAN-Nachrichten für einen dynamischen MTC empfangen als gesendet werden können, speichert die Warteschlange die überzähligen Nachrichten zwischen. Wenn die Warteschlange voll ist, verdrängen eingehende Nachrichten die Nachrichten mit geringerer Priorität. Dies erlaubt es unter gegebenen Lastanforderungen die Größe des dynamischen MTCs und der Warteschlange effizient zu planen.

Der grundlegende Vorteil der Konvertierung mit exklusivem MTC gegenüber dem dynamischen ist die garantierte Latenz der Nachricht ab der Verarbeitung durch das Gateway. Mit den dynamischen MTCs kann aufgrund der priorisierten Übermittlung nur für die Nachricht höchster Priorität eine maximale Latenz garantiert werden. Für SRT- und NRT-Ereigniskanäle tritt bei beiden MTCs mindestens ein Jitter von der Größe der Periode des Konvertierungstasks auf. Dieser Jitter ist durch die Abbildung der asynchronen CAN-Nachrichten auf den periodischen Task- bzw. Kommunikationsplan zurückzuführen. Da HRT-Ereigniskanäle zum TTP/C-Netz synchron senden, geht der Kommunikationsjitter von HRT-Nachrichten gegen null. HRT-Ereigniskanäle werden nur in exklusive MTCs eingeplant, da sonst die Garantie der zeitlichen Eigenschaften nicht gewährleistet werden kann. Sonst könnten sich mehrere HRT-Ereigniskanäle in einem dynamischen MTC gegenseitig blockieren. Die Ende-zu-Ende Latenz von HRT-Nachrichten im exklusiven MTC lässt sich a priori berechnen und unterliegt nur einem synchronisationsbedingten Jitter (ε). Damit eignet sich diese Kombination (**Fall A1** und **A2**) für sicherheitskritische Informationen.

TTP/C nach CAN

Die Weiterleitung von TTP/C-Ereigniskanälen ins CAN-Netz erfordert neben der Nachrichtenkonvertierung und der Adressumsetzung auch eine Priorisierung der Nachrichten. Dabei unterscheiden sich die Anforderungen an die Weiterleitung für HRT-, SRT- und NRT-Ereigniskanäle stark. Daher wird die Konvertierung von TTP/C nach CAN, wie in Abbildung 3.13 dargestellt, in zwei verschiedene Verfahren untergliedert: Umsetzung von HRT- und Nicht-HRT-Ereigniskanäle. Nachrichten von HRT-Ereigniskanälen nutzen dafür fest vorgesehene CAN-Zeitfenster. Die Nachrichten von SRT- oder NRT-Ereigniskanälen müssen in einer globalen

CAN-Warteschlange abgelegt werden. Innerhalb dieser Warteschlange werden die Nachrichten nach ihrer Priorität sortiert und ins CAN-Netz gesendet.

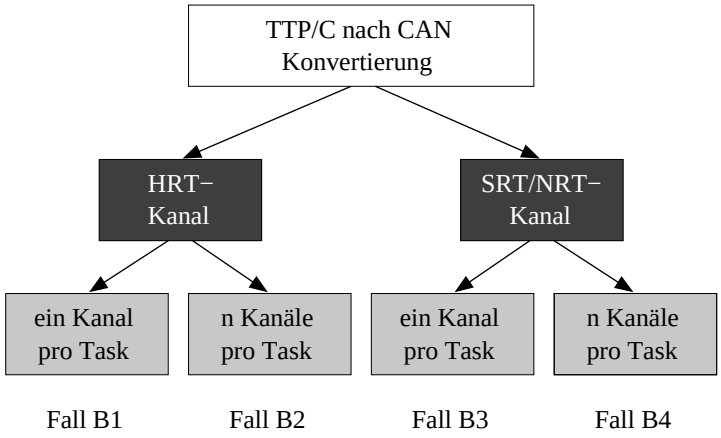


Abbildung 3.13: TTP/C nach CAN Konvertierungsfälle

Konvertierung von HRT-Ereigniskanälen

Für die Konvertierung von TTP/C-HRT-Ereigniskanälen nach CAN ist die Uhrensynchronisation beider Netze notwendig. Nur mit synchronen Zeiten kann der Zeitschlitz im CAN vom Gateway mit minimaler und konstanter Verzögerung genutzt werden. Voraussetzung hierfür ist, dass die Perioden der Zeitschlitze in CAN und TTP/C und des Konvertierungstasks identisch sind. Der Task zur Konvertierung eines HRT-Ereigniskanals wird so geplant, dass zum Ende seiner maximalen Konvertierungszeit der CAN-Zeitschlitz für die Übermittlung beginnt. Damit kann im Anschluss an die Konvertierung die HRT-Nachricht sofort gesendet werden. Dies erfordert jedoch eine genaue zeitliche Abstimmung des TTP/C-Zeitschlitzes, des Konvertierungstasks und des CAN-Zeitschlitzes für die HRT-Nachrichten. Werden die Zeitfenster und der Task optimal geplant, kann eine minimale konstante Verzögerung (ΔT_{min}) erreicht werden. Sie setzt sich aus der Länge des TTP/C-Zeitschlitzes ($\Delta t_{tZeitschlitz}$), der maximalen Konvertierungszeit ($\Delta t_{tKonvertierung}$) und der Länge des CAN-Zeitschlitzes ($\Delta t_{cZeitschlitz}$) zusammen. Die minimale Verzögerung für den **Fall B1** ist in Gleichung 3.18 dargestellt. Wie im Vergleich mit der Gleichung 3.9 zu erkennen ist, hängt diese minimale Ende-zu-Ende Verzögerung von den gleichen Parametern ab, wie die minimale Latenz bei der Konvertierung von CAN-HRT-Ereigniskanälen mit exklusivem MTC (**Fall A1** und **A2**).

$$\Delta T_{min} = \Delta t_{tZeitschlitz} + \Delta t_{tKonvertierung} + \varepsilon + \Delta t_{cZeitschlitz} \tag{3.18}$$

Die Variable ε stellt einen zeitlichen Puffer zwischen dem Konvertierungstask und dem CAN-Zeitschlitz dar. Dadurch können Ungenauigkeiten der Uhrensynchronisation zwischen den Netzen bis zur Größe von ε toleriert werden. Der Jitter der Ende-zu-Ende Verzögerung ist damit durch die Uhrensynchronisation begrenzt und hat die maximale Größe von ε. Wird der TTP/C-Zeitschlitz nicht direkt auf den Konvertierungstask abgestimmt, entsteht eine Lücke zwischen dem Zeitschlitz für die TTP/C-HRT-Ereignisnachricht und der Verarbeitung durch das Gateway (φ_1). Diese Lücke kann maximal den Wert einer Periodenlänge des Konvertierungstasks

annehmen. Muss nach dem Verarbeiten der Nachricht durch den Konvertierungstask noch eine Zeitspanne überbrückt werden, bis die CAN-HRT-Nachricht gesendet werden kann, entsteht eine zusätzliche Verzögerung (φ_2). Diese Verzögerung bis zum Beginn des CAN-Zeitschlitzes kann maximal den Wert der Periode des CAN-Zeitschlitzes annehmen. Da die Perioden der Zeitschlitze und des Tasks identisch sind, ergibt sich die maximale Verzögerung in Gleichung 3.19. Die Gleichung 3.20 beschreibt für den **Fall B1** die Ende-zu-Ende Verzögerungszeit zwischen dem minimalen und maximalen Wert. Diese Gleichung ist von den gleichen Parametern wie die Gleichung 3.11 für die **Fälle A1** und **A2** abhängig.

$$\Delta T_{max} = \Delta t_{tZeitschlitz} + 2 \cdot \Delta t_{Periode} + \Delta t_{tKonvertierung} + \varepsilon + \Delta t_{cZeitschlitz} \quad (3.19)$$

$$\Delta T = \Delta t_{tZeitschlitz} + \varphi_1 + \Delta t_{tKonvertierung} + \varphi_2 + \varepsilon + \Delta t_{cZeitschlitz} \quad (3.20)$$

Eine fehlende oder schlechte Synchronisation beider Netze führt zum Verpassen der CAN-Zeitschlitze. Daher muss für die Größe des zeitlichen Puffers (ε) die maximale Ungenauigkeit der Zeitsynchronisation angenommen werden. Es wird dann sichergestellt, dass der CAN-Zeitschlitz immer erreicht wird. Die Verzögerungen der HRT-Nachrichten lassen sich damit anhand der Eigenschaften der Zeitschlitze, der maximalen Ausführungszeit des Konvertierungstasks und des globalen Plans a priori berechnen. Mit den berechneten Verzögerungen werden die Fristen der Nachrichten festlegt.

Ein Konvertierungstask kann mehrere HRT-Ereigniskanäle (**Fall B2**) gleichzeitig bearbeiten. Dies ist jedoch aufgrund höherer Verzögerungszeiten der Nachrichten nicht sinnvoll. Werden mehrere HRT-Nachrichten in einem exklusiven MTC (siehe Abbildung 3.14 (a)) übertragen, vergrößert sich die Länge des TTP/C-Zeitschlitzes ($\Delta t_{tZeitschlitz}$) je nach Anzahl und Länge der Nachrichten der Ereigniskanäle. Zusätzlich dazu nimmt die Verarbeitung der verschiedenen Ereigniskanäle im Konvertierungstask ($\Delta t_{tKonvertierung}$) eine weitere Verzögerung in Anspruch. Dabei ist die Dauer der Verarbeitung von der Anzahl der zuvor zu verarbeitenden und zu versendenden Ereigniskanäle abhängig. Dieser Zusammenhang ist in Gleichung 3.21 verdeutlicht.

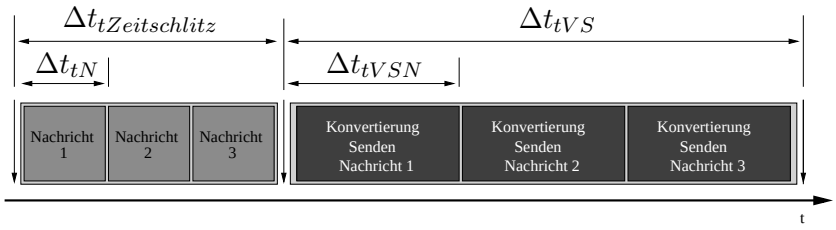
$$\Delta t_{tKonvertierung}(i) = \sum_{j=1}^{i-1} \Big(\Delta t_{tKonvertierung}(j) + \varphi_2(j) + \varepsilon + \Delta t_{cZeitschlitz}(j) \Big) + \Delta t_{tKonvertierung}(i) \quad (3.21)$$

$$mit \quad i = 1 \ldots n$$

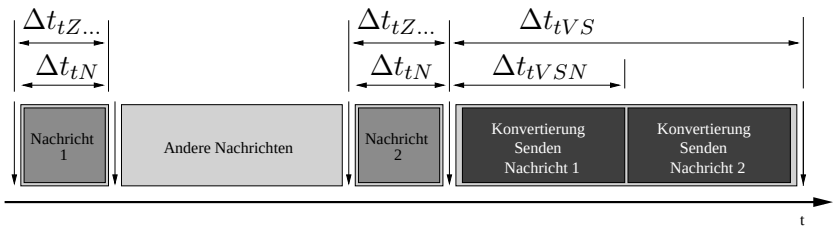
Durch das Einsetzen der Gleichung 3.21 in die Ende-zu-Ende Verzögerungen für den **Fall B1** (Gleichung 3.20) und unter Beachtung der Länge des Zeitschlitzes ($\Delta t_{tZeitschlitz}$) ergibt sich die Gesamtverzögerung für den **Fall B2**. Die Verarbeitung im Konvertierungstask ($\Delta t_{tKonvertierung(i)}$) unterliegt dabei keinem Jitter, da die Verarbeitungdauer jedes einzelnen Ereigniskanals im Konvertierungstask konstant ist. Ein Jitter der Ende-zu-Ende Verzögerung entsteht erst bei der Übergabe der Nachricht an das CAN-Netz durch die Ungenauigkeit der Synchronisation beider Netze. Daher ist der Jitter wie im **Fall B1** durch ε gegeben.

Die Abbildungen 3.14 (a) und (c) verdeutlichen diesen Unterschied zwischen der Übermittlung mehrerer HRT-Nachrichten in einem MTC und darauffolgende Verarbeitung (**Fall**

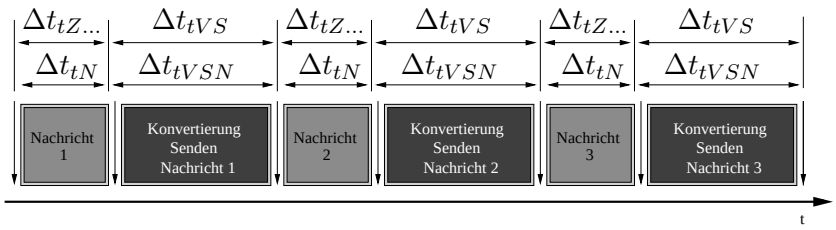
B2) und der sequentiellen Übermittlung und Verarbeitung (**Fall B1**) von HRT-Nachrichten. Bei der sequentiellen Übermittlung und Verarbeitung in Abbildung 3.14 (c) sind die CAN-Sendezeitschlitze der einzelnen Nachrichten, im Vergleich zur Übermittlung in einem MTC aus Abbildung 3.14 (a), verschoben. Dies hat jedoch keine Auswirkung auf die Qualität der Echtzeitdaten, da diese Verschiebung a priori vorgenommen wird und sich die Periodizität der Nachrichten nicht ändert. Auch die Erhebung der Echtzeitinformationen wird dabei in der Phase verschoben. Die Ende-zu-Ende Verzögerung kann in diesem Fall bei optimaler Planung der Wartezeiten (φ_1 und φ_2) bei der Übermittlung von jedem Ereigniskanal in je einen TTP/C-Zeitschlitz, minimiert werden.



(a) Mehrere Ereigniskanäle in einem MTC pro Konvertierungstask (Fall B2)



(b) Mehrere Ereigniskanäle in mehreren MTC pro Konvertierungstask (Fall B2)



(c) Ein Ereigniskanal pro Konvertierungstask (Fall B1)

Abbildung 3.14: Anordnungen zur Konvertierung von TTP/C-HRT-Ereigniskanälen

Werden mehrere HRT-Nachrichten in verschiedenen Zeitschlitzen von einem Konvertierungstask bearbeitet (**Fall B2**), kann der Task erst aktiviert werden, wenn der letzte Zeitschlitz gelesen wurde. Damit erhöht sich die Verzögerung, um die Zeit der Übermittlung der einzelnen Nachrichten und die Zeitdifferenz zwischen den relevanten Zeitschlitzen. Dieser Fall ist ähnlich dem, dass mehrere Nachrichten in einem MTC (siehe Abbildung 3.14 (a)) übermittelt werden, mit dem Unterschied, dass zwischen den Nachrichten Lücken durch andere Verwen-

dung des Kommunikationsplans auftreten. Für die Berechnung der Ende-zu-Ende Latenz muss als Länge des TTP/C-Zeitschlitzes ($\Delta t_{tZeitschlitz}$) dann die Zeitspanne zwischen dem Beginn des TTP/C-Zeitschlitzes, in dem die betreffende Nachricht gesendet wird, und dem Ende des letzten vom Konvertierungstask gelesenen Zeitschlitz eingesetzt werden. Die Abbildung 3.14 (b) stellt dieses dar. Eine Aufteilung der Konvertierungstasks für jeden Ereigniskanal, wie in Abbildung 3.14 (c) dargestellt, verringert auch hier die Verzögerung der Echtzeitnachrichten.

Die Verzögerung der Übermittlung der HRT-Nachrichten für die verschiedenen Anordnungen der Ereigniskanäle kann aufgrund des statischen Kommunikationsplans direkt berechnet werden. Daher wird je nach zeitlichen Eigenschaften des HRT-Ereigniskanals entschieden, ob der Ereigniskanal einen einzelnen Konvertierungstask erhält oder sich mit anderen Kanälen einen Konvertierungstask teilt. Zu beachten ist, dass die erreichbaren Verzögerungen für einen Ereigniskanal durch einen eigenen Konvertierungstask minimiert werden können.

Konvertierung von SRT- und NRT-Ereigniskanälen

Die SRT- und NRT-Ereigniskanäle verwenden im CAN-Netz keine Zeitschlitze. Sie müssen über die Arbitrierung in Konkurrenz zu anderen Nachrichten den Bus erhalten und können dann ihre Nachrichten versenden. Das Gateway übergibt daher die umgewandelten SRT- und NRT-Nachrichten (**Fall B3** und **B4**) schnellstmöglich an das CAN-Netz. Diese Nachrichten werden durch das Gateway periodisch gesendet, die Periode der Übermittlung hängt dabei vom Nachrichtenaufkommen ab. Werden SRT- oder NRT-Ereigniskanäle aus TTP/C nach CAN geleitet, muss sichergestellt werden, das wichtigere Nachrichten bevorzugt gesendet werden. Dazu wird eine globale CAN-Sendewarteschlange im Gateway benötigt. Es ist auch möglich, dass jeder Konvertierungstask eine eigene lokale Warteschlange besitzt, welche für die ihm zugehörigen Ereigniskanäle reserviert ist. Nutzen die Konvertierungstasks eigene Warteschlangen, kann aber nicht garantiert werden, dass immer die wichtigste Nachricht unter allen im Gateway wartenden Nachrichten gesendet wird. Hingegen sichert eine globale Warteschlange, welche von den Konvertierungstasks gemeinsam befüllt wird, über eine Sortierung nach Prioritäten, dass immer die wichtigste Nachricht zuerst gesendet wird. Es wird daher eine globale Warteschlange im Gateway verwendet. Die Sortierung der Nachrichten muss vor jedem Senden durchgeführt werden, da sich die Prioritäten der SRT-Nachrichten dynamisch ändern. Bei der Sortierung werden die Nachrichten entfernt, welche durch zu lange Wartezeiten ungültig geworden sind.

Die Berechnung der Fristen für SRT-Nachrichten kann das Gateway mit hoher Genauigkeit durchführen, da die TTP/C-Kommunikation deterministisch ist. Die Position der Zeitschlitze für die TTP/C-Nachrichten und die Position der verarbeitenden Konvertierungstasks im globalen TTP/C-Plan sind bekannt. Damit kann die feste Verzögerungszeit vom Senden durch den TTP/C-Knoten bis zum Empfangen durch den Konvertierungstask bestimmt werden. Verarbeitet jeder Konvertierungstask nur einen Ereigniskanal, kann die Verzögerung bis zur Bearbeitung auf nahezu null minimiert werden, sofern der Kommunikationsplan optimal mit dem Konvertierungstask abgestimmt ist. Sonst fällt eine Verzögerung bis zum Beginn des Konvertierungstasks (φ_1) an. Die Dauer der Übermittlung der Nachricht ist durch die Länge des Zeitschlitzes ($\Delta t_{tZeitschlitz}$) definiert. Wird der Beginn der Gültigkeit der Nachricht mit dem Zeitpunkt der Erfassung der Daten angegeben, muss die Zeitspanne bis zum Senden durch den Produzent bei der Gültigkeitsberechnung mit berücksichtigt werden. Es wird im Folgenden davon ausgegangen, dass die Gültigkeit mit dem Senden der Nachricht beginnt. Die Gleichungen 3.22 und 3.23 geben damit die Berechnung der Frist (t_{Frist}) für eine eingehende Nachricht an. Die

Restgültigkeit ($\Delta t_{Restgültigkeit}$) der Nachricht ist beim Empfang durch den Konvertierungstask im Voraus bestimmbar, da der Beginn der Gültigkeitsdauer mit dem Übermittlungszeitpunkt zusammenfällt und alle Parameter konstant sind.

$$\Delta t_{Restgültigkeit} = \Delta t_{Gültigkeitsdauer} - \varphi_1 - \Delta t_{tZeitschlitz} \tag{3.22}$$

$$t_{Frist} = t_{jetzt} + \Delta t_{Restgültigkeit} \tag{3.23}$$

Werden mehrere Ereigniskanäle innerhalb eines MTCs oder innerhalb von mehreren unabhängigen MTCs von einem Konvertierungstask (**Fall B4**) gehandhabt, muss die Verzögerung durch die sequentielle Übermittlung der Nachrichten bei der Fristberechnung mitberechnet werden. Dazu muss für die Berechnung der Restgültigkeit die Zeitspanne von Beginn der Übermittlung der Nachricht im MTC bis zum Beginn des Konvertierungstasks betrachtet werden. Daher wird anstelle der Länge des Zeitschlitzes ($\Delta t_{tZeitschlitz}$) die Verzögerung vom Beginn des sendenden MTCs bis zum Ende des spätesten MTCs verwendet. Dies ist damit begründet, dass der Konvertierungstask erst nach dem Empfang des spätesten MTCs die Verarbeitung beginnen kann. Der Konvertierungstask wird dann je nach Planung durch eine Wartezeit verzögert (φ_1).

Die Berechnung der Frist (t_{Frist}) muss zu Beginn des Konvertierungstasks durchgeführt werden, da sonst eine weitere Verzögerung innerhalb des Konvertierungstasks die Ermittlung der Frist verfälscht. Wurde die Frist als absoluter Zeitpunkt bestimmt, kann die Nachricht konvertiert und in die Warteschlange prioritätsbasiert eingeordnet werden. Für NRT-Nachrichten ist die Fristberechnung nicht notwendig, weil diese Nachrichten keine zeitlich begrenzte Gültigkeit haben. SRT-Nachrichten benötigen die Berechnung der Frist, da sie ihre Priorität dynamisch nach der Restgültigkeit anpassen und vom Gateway verworfen werden müssen, wenn sie ungültig werden. Die Prioritätsberechnung der SRT-Nachrichten erfolgt nach der Formel 3.3, welche auf Seite 69 beschrieben ist. Die Anpassung der Priorität ist, aufgrund der dynamischen Priorisierung, vor jedem Senden der CAN-Nachrichten erforderlich.

Die Konvertierung von SRT- und NRT-Nachrichten kann für Status- und Ereignisnachrichten unterschiedlich gehandhabt werden. Beide Nachrichtentypen werden nach Priorität in die Warteschlange eingefügt. Aufgrund der Eigenschaften der beiden Nachrichtentypen ergeben aber sich Unterschiede in der Behandlung der Nachrichten für den Fall, dass beim Einfügen einer Nachricht in der Warteschlange bereits eine Nachricht des gleichen Kanals vorhanden ist. Bei Ereignisnachrichten muss darauf geachtet werden, dass die neue Nachricht nicht vor der alten in der Warteschlange platziert wird. Dies wird dadurch realisiert, dass neue Ereignisnachrichten gleicher Priorität in der Warteschlange nach Nachrichten gleicher Priorität eingeordnet werden. Für Statusnachrichten können bei der Behandlung von mehrfach vorhandenen Nachrichten in der Warteschlange zwei Fälle unterschieden werden. Die einfachste Möglichkeit ist es, die neue Nachricht wie eine Ereignisnachricht einfach je nach Priorität in die Warteschlange einzufügen. Es wird dann zuerst die alte Nachricht und dann die neue den Empfänger erreichen. Dabei erhält der Empfänger mit der alten Nachricht ältere aber noch gültige Statusinformationen, obwohl bereits ein neuer Zustandswert im Gateway verfügbar ist. Dies kann dadurch verhindert werden, indem beim Einsortieren einer neuen Statusnachricht in die Warteschlange die Werte von Statusnachrichten des gleichen Ereigniskanals auf den Wert der neuen Statusnachricht aktualisiert werden. Damit gelangt mit jeder Statusnachricht der aktuelle Statuswert aus dem Gateway zum Empfänger.

Ist die Warteschlange zum Zeitpunkt des Einfügens einer neuen Nachricht bereits voll, kann die Nachricht eine Nachricht aus der Warteschlange verdrängen, wenn sie eine höhere Priorität besitzt. Damit ermöglicht es die Warteschlange in Phasen hoher CAN-Busbelastung wichtige Nachrichten über eine gewisse Zeit zu puffern. Sobald der Bus wieder arbitrierbar ist, werden die gepufferten Nachrichten gesendet. Neben dem Befüllen der Warteschlange muss das Gateway versuchen alle Nachrichten vor Ablauf ihrer Fristen zuzustellen. Dies ist durch eine periodische Sendefunktion realisiert. Die Periode dieser Funktion richtet sich nach dem Nachrichtenaufkommen und der Geschwindigkeit des CAN-Busses. Nachrichten mit abgelaufenen Fristen müssen vom Gateway aus der Warteschlange, je nach Toleranz der Fristverletzung, entfernt werden.

Die Ende-zu-Ende Verzögerung von Nachrichten aus SRT- und NRT-Ereigniskanälen ist variabel. Nachrichten aus SRT- oder NRT-Ereigniskanälen können durch wichtigere Nachrichten in der Warteschlange nach hinten verschoben oder aus dieser verdrängt werden. Durch die Nutzung des CAN-Busses von anderen Teilnehmern, kann vom Gateway nicht sichergestellt werden, dass die Nachricht an der ersten Position in der globalen Warteschlange ohne Verzögerung gesendet wird. Die Verzögerung einer SRT- oder NRT-Nachricht kann daher nicht bestimmt werden. Es ist lediglich eine Aussage über SRT- oder NRT-Nachrichten möglich, wenn sie die höchste Priorität im gesamten verteilten System haben. Eine solche Nachricht wird in die Warteschlange an vorderster Position eingefügt und beim nächsten Sendetermin gesendet. Diese Nachricht erhält bei der Busarbitrierung definitiv den CAN-Bus und wird ohne Verzögerung gesendet, sofern zum Sendezeitpunkt keine andere Nachricht den Bus blockiert. Unter dieser Annahme ergibt sich eine berechenbare Verzögerung, welche sich aus der Dauer der Übermittlung der Nachricht im TTP/C-Netz, der Wartezeit auf den Konvertierungstask (φ_1), der Konvertierungszeit ($\Delta t_{tKonvertierung}$), der Wartezeit auf den nächsten Sendetermin der globalen Warteschlange (φ_3), der Übermittlungszeit der CAN-Nachricht ($\Delta t_{cNachricht}$) und der Wartezeit bei blockiertem CAN-Bus ($\Delta t_{cBlockierung}$) zusammensetzt. Die Dauer der Übermittlung der Nachricht im TTP/C-Netz richtet sich auch danach, ob ein Konvertierungstask nur einen Ereigniskanal handhabt oder mehrere. Die Gleichung 3.24 beschreibt die Ende-zu-Ende Verzögerung für einen Konvertierungstask, welcher nur einen Ereigniskanal verwaltet (**Fall B3**). Es ergibt sich dabei als TTP/C-Übermittlungszeit die Länge des Zeitschlitzes ($\Delta t_{tZeitschlitz}$).

$$\begin{aligned} \Delta T_{\text{höchste Priorität}} &= \Delta t_{tZeitschlitz} + \varphi_1 + \Delta t_{tKonvertierung} + \\ &\quad \varphi_3 + \Delta t_{cBlockierung} + \Delta t_{cNachricht} \end{aligned} \tag{3.24}$$

Es ist zu beachten, dass es nicht möglich ist die Blockierungszverzögerung im CAN-Netz ($\Delta t_{cBlockierung}$) direkt zu bestimmen, da der CAN-Bus zum Sendezeitpunkt belegt oder frei sein kann. Die Übermittlungszeit der CAN-Nachricht ($\Delta t_{cNachricht}$) hängt von der Länge der Nachricht ab. Alle weiteren angegebenen Parameter sind durch den globalen Plan des TTP/C-Netzes im Voraus bekannt. Die Verzögerung durch das TTP/C-Netz kann damit statisch bestimmt werden. Bei optimaler Planung der Zeitschlitze und Tasks, kann die Nachrichtenverzögerungszeit, durch Reduzierung der Wartezeiten (φ_1, φ_3), minimiert werden. Wird nicht von der höchsten Priorität der betrachteten Nachricht ausgegangen, müssen die Verzögerungen für die Warteschlange ($\Delta t_{tWarteschlange}$) und die Arbitrierung des CAN-Busses ($\Delta t_{CAN} = \Delta t_{cBlockierung} + \Delta t_{cArbitrierung} + \Delta t_{cNachricht}$) mit einberechnet werden. Die Gleichung 3.25 gibt damit die nicht bestimmbare Ende-zu-Ende Verzögerung an.

$$\begin{aligned} \Delta T &= \Delta t_{tZeitschlitz} + \varphi_1 + \Delta t_{tKonvertierung} + \\ &\quad \Delta t_{tWarteschlange} + \varphi_3 + \Delta t_{CAN} \end{aligned} \tag{3.25}$$

Die Schwankung der Ende-zu-Ende Latenz setzt sich für Gleichung 3.25 aus der Verzögerung der Nachricht in der Warteschlange ($\Delta t_{tWarteschlange}$), der Wartezeit durch einen blockierten CAN-Bus ($\Delta t_{cBlockierung}$) und der Zeitspanne für die Arbitrierung des CAN-Busses ($\Delta t_{cArbitrierung}$) zusammen. Dadurch, dass die Verzögerungen durch die Warteschlange und die Arbitrierung des CAN-Busses nicht angegeben werden können, existiert keine obere Schranke für die Ende-zu-Ende Verzögerung und den Jitter. Für den Fall, dass die höchstpriore Nachricht betrachtet wird gilt die Gleichung 3.24 und der der Jitter lässt sich mit der Blockierungszeit des CAN-Busses ($\Delta t_{cBlockierung}$) beschreiben.

Werden mehrere Kanäle pro Konvertierungstask (**Fall B4**) genutzt, muss statt der Länge des Zeitschlitzes ($\Delta t_{tZeitschlitz}$) für die Übermittlungszeit die Verzögerung vom Beginn des sendenden MTCs der Nachricht bis zum Ende des letzten MTC betrachtet werden. Weiterhin muss eine größere Verarbeitungszeit ($\Delta t_{tKonvertierung}$) der Ereigniskanäle angenommen werden. Im Vergleich zum **Fall B2** ist die Latenz einer Nachricht nicht von vorher verarbeiteten Nachrichten abhängig, da die einzelnen Nachrichten nicht direkt nach ihrer Verarbeitung vom Konvertierungstask ins CAN-Netz übergeben werden. Die Übermittlung erfolgt für SRT- und NRT-Nachrichten mittels der globalen Warteschlange nach der Verarbeitung aller Ereigniskanäle außerhalb des Konvertierungstasks.

Der Unterschied zwischen der Handhabung von HRT-Ereigniskanälen zu anderen Nachrichten besteht bei der Konvertierung von TTP/C zu CAN darin, dass HRT-Nachrichten keine Pufferung und somit keine Warteschlange benötigen, da sie bei einer globalen synchronen Zeit direkt gesendet werden können. Dies kann nur durch eine unzureichende Synchronisation oder durch einen Fehler verhindert werden. Der Vorteil bei der Konvertierung von HRT-Ereigniskanälen ist damit die garantierte Ende-zu-Ende Verzögerung. Die Ende-zu-Ende Verzögerung kann für SRT- und NRT-Ereigniskanäle nicht berechnet werden. Zeitliche Aussagen über NRT- oder SRT-Ereigniskanäle sind nur für den höchstprioren Ereigniskanal in der Warteschlange im gesamten CAN-Netz möglich. Die SRT- und NRT-Nachrichten werden durch die globale Warteschlange nach ihrer Priorität versendet. Eine unterschiedliche Behandlung von Ereignis- und Statusnachrichten erlaubt eine effiziente Unterstützung beider Nachrichtentypen. Der Vorteil der Konvertierung von SRT- und NRT-Nachrichten ist es aber, dass Nachrichten über Perioden hoher Busbelastung zwischengespeichert und bei Freiwerden des Busses gesendet werden.

3.4.5 Zusammenfassung

Das in diesem Abschnitt vorgestellte Konzept eines Gateways zwischen CAN und TTP/C erfüllt die in Abschnitt 3.3 gestellten Anforderungen. Es stellt die netzübergreifende Adressierung der verwendeten Ereigniskanäle zur Verfügung und erlaubt damit eine transparente Kommunikation der Anwendung über die Netzgrenze hinweg. Die dynamische Bindung der Ereigniskanäle zu netzlokalen Adressen (Event-Tags) in beiden Netzen ermöglicht eine flexible Kommunikation. In TTP/C-Netz erlaubt die Bindung der Kanäle eine dynamische Nutzung der Zeitschlitze. Es können zur Laufzeit Ereigniskanäle, welche in einem TTP/C-Zeitschlitz

zugeordnet sind, ausgetauscht, hinzugefügt oder entfernt werden. Diese Bindung zur Laufzeit ist aufgrund von Verzögerungen bei der Neubindung nicht für sicherheitskritische Echtzeitnachrichten anwendbar. Es bieten sich aber verschiedene Applikationsszenarien wie z.B. die Anforderung von dynamischen Diagnoseinformationen oder weiteren Sensordaten zur Laufzeit an.

Die Verwendung der Publish/Subscribe-Middleware ermöglicht es, Produzent und Konsument voneinander zu entkoppeln. Das Gateway entscheidet anhand von Filterregeln und Abonnements der Ereigniskanäle, welchen Kanal es in das andere Netz weiterleitet. Damit stellt es die Informationskapselung innerhalb der einzelnen Netze sicher. Der Entwickler kann den Informationsfluss durch die explizite Freigabe von Ereigniskanälen a priori begrenzen und verhindern, dass in das andere Netz irrelevante Daten weitergeleitet werden. Dadurch wird eine Kontrolle des Informationsflusses zwischen den Netzen erreicht. Mit Hilfe der Abonnementverwaltung werden nicht abonnierte Ereigniskanäle vom Gateway nicht weitergeleitet. Nur das direkte Abonnement eines Kanals aus dem anderen Netz veranlasst das Gateway den Ereigniskanal bereitzustellen. Damit verhindert das Gateway die unnötige Nutzung von Bandbreite für nicht angeforderte Informationen. Das Gateway kann somit z.B. mehr Ereigniskanäle aus dem TTP/C-Netz anbieten als die CAN-Anwendung benötigt. Die CAN-Anwendung kann später so erweitert werden, dass sie mit den vom Gateway angebotenden Ereigniskanälen neue Funktionalitäten umsetzt.

Diese Informationskapselung kann z.B. in einer Brake-By-Wire Anwendung genutzt werden: Die sicherheitskritischen Funktionen der Anwendung sind im TTP/C-Netz angesiedelt, während das CAN-Teilsystem dieser Anwendung Bremskraftdaten zur Diagnose aufzeichnet. Bietet das Gateway neben diesen Bremskraftdaten auch Sensorinformationen über die Temperatur der Bremsscheiben, kann die CAN-Anwendung zu einem späteren Zeitpunkt so erweitert werden, dass sie anhand der Temperaturdaten eine Kühlung der Bremsscheiben realisiert. Mit diesem Konzept der Bereitstellung von umfassenden Informationen aus der statischen TTP/C-Anwendung können, durch unterschiedliche CAN-Applikationen bei identischer TTP/C-Anwendung, veränderte oder erweiterte Versionen der Gesamtanwendung erstellt werden. Die Modularität dieses Ansatzes unterstützt eine einfache Erweiterung oder Nachrüstung der CAN-Anwendung ohne Änderungen des TTP/C-Teilsystems.

Die globale Adressierung erlaubt eine für die Anwendung transparente Nutzung der Ereigniskanäle. Dabei ist es der Anwendung nicht bekannt, wo die abonnierten Ereigniskanäle produziert werden. Dies erlaubt eine einfache Migration von Anwendungsteilen in ein anderes Netz. So kann z.B. ein Teil einer CAN-Anwendung in eine TTP/C-Teilanwendung umgewandelt werden. Die TTP/C-Teilanwendung kommuniziert dann mit Hilfe des Gateways mit der nicht migrierten CAN-Anwendung über die gleichen Ereigniskanäle wie vor der Migration. Dies ist ohne die Anpassung der im CAN-Netz verbliebenen Teilanwendung möglich.

Das Gateway unterstützt die Weiterleitung von Ereigniskanälen mit verschiedenen Echtzeitklassen. Je nach Klasse und Richtung wird die Weiterleitung unterschiedlich realisiert. Dabei entstehen abhängig von der Konvertierungsart Kommunikationsjitter in der Ende-zu-Ende Latenz. Die Tabelle 3.2 stellt die Ende-zu-Ende Verzögerungen für die Fälle aus den Abbildungen 3.9 und 3.13 dar. Für die Weiterleitung von HRT-Ereigniskanälen ist der Jitter in beiden Richtungen mit maximal den Ungenauigkeiten durch die Uhrensynchronisation (ε) gegeben und daher minimal. Es kann für harte Echtzeitnachrichten so eine feste berechenbare Ende-zu-Ende

Verzögerung in beide Richtungen, welche von den gleichen Parametern abhängt, erreicht werden.

Fall	**Ende-zu-Ende Verzögerung ΔT**	**Jitter**	**Gl.**
A1, A2	$\Delta t_{cZeitschlitz} + \varepsilon + \varphi_1 + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz}$	ε	3.11
A3, A4	$\Delta t_{CAN} + \varphi_1 + \Delta t_{tKonvertierung} + \varphi_2 + \Delta t_{tZeitschlitz}$	$\Delta t_{cBlockierung} + \Delta t_{cArbitrierung} + \Delta t_{Periode}$	3.12
A5, A6	$\Delta t_{CAN} + \varphi_1 + \Delta t_{tKonvertierung} + \Delta t_{tWarteschlange} + \varphi_2 + \Delta t_{tZeitschlitz}$	$\Delta t_{cBlockierung} + \Delta t_{cArbitrierung} + \Delta t_{Periode} + \Delta t_{tWarteschlange}$	3.16
B1, B2	$\Delta t_{tZeitschlitz} + \varphi_1 + \Delta t_{tKonvertierung} + \varphi_2 + \varepsilon + \Delta t_{cZeitschlitz}$	ε	3.20
B3, B4	$\Delta t_{tZeitschlitz} + \varphi_1 + \Delta t_{tKonvertierung} + \Delta t_{tWarteschlange} + \varphi_3 + \Delta t_{CAN}$	$\Delta t_{tWarteschlange} + \Delta t_{cBlockierung} + \Delta t_{cArbitrierung}$	3.25

Tabelle 3.2: Übersicht der Ende-zu-Ende Verzögerungen

Bei der Einleitung von SRT- oder NRT-Nachrichten ins TTP/C-Netz entsteht durch die Abbildung der asynchronen Kommunikation auf die periodische TTP/C-Kommunikation ein Jitter von der Größe der Periode der Verarbeitung der Nachrichten ($\Delta t_{Periode}$). Wird die Verarbeitung der eingehenden CAN-Nachrichten nur einmal pro Clusterzyklus durchgeführt, ist der Jitter von der Größe des Clusterzyklus. Dieser Jitter kann nur durch die häufigerere Einplanung des Konvertierungstasks und des sendenden Zeitschlitzes verringert werden. In jedem Fall wird durch die Konvertierung von CAN nach TTP/C ein Kommunikationsjitter verursacht, welcher größer als die eigentliche Bearbeitungsverzögerung im TTP/C-Netz ist. Damit können SRT-Nachrichten mit kurzen Fristen nicht unterstützt werden. Ereigniskanäle, welche eine kurze Frist haben und daher einen kleinen Jitter benötigten, müssen die synchrone Übermittlung mit HRT-Eigenschaften nutzen. Zusätzlich zu diesem Jitter entstehen für SRT- und NRT-Nachrichten nicht planbare Verzögerungen durch die Kommunikation im CAN-Netz (Δt_{CAN}) und den Aufenthalt der Nachricht in der Warteschlange ($\Delta t_{tWarteschlange}$). Die Verzögerung innerhalb des CAN-Netzes ergibt sich aus der Zeit für die Arbitrierung ($\Delta t_{cArbitrierung}$), welche in der Regel unbestimmbar ist, der Wartezeit durch einen blockierten Bus ($\Delta t_{cBlockierung}$) und der eigentlichen Übermittlungszeit der CAN-Nachricht. Kann sichergestellt werden, dass die betrachtete Nachricht die Nachricht mit höchster Priorität auf dem Bus ist, entfällt der Jitter für die Arbitrierung ($\Delta t_{cArbitrierung}$). Durch die Möglichkeit der Blockierung des Busses zur Sendezeit durch eine andere Nachricht entsteht eine Schwankung der Latenz mit der Länge der längsten Nachricht. Ist die betrachtete Nachricht die Nachricht höchster Priorität in der Warteschlange, entfällt die dynamische Verzögerung durch den Aufenthalt in der Warteschlange ($\Delta t_{tWarteschlange}$), ansonsten ist diese Verzögerung nicht bestimmbar.

Werden SRT- oder NRT-Nachrichten vom TTP/C-Netz ins CAN-Netz geleitet entsteht ein Schwankung der Ende-zu-Ende Latenz durch die Warteschlange ($\Delta t_{tWarteschlange}$) und die Übertragung auf dem CAN-Bus ($\Delta t_{cArbitrierung}$, $\Delta t_{cBlockierung}$). Wird bei der betrachteten Nachricht von der Nachricht mit höchster Priorität in Warteschlange und CAN-Netz ausgegangen, minimiert sich der Jitter auf die veränderliche Verzögerung durch die Busblockierung ($\Delta t_{cBlockierung}$). Es besteht für SRT- und NRT-Nachrichten in beiden Richtungen eine Ab-

hängigkeit der Ende-zu-Ende Verzögerung und des Jitters von der Priorität der Nachricht. Dies wird dadurch verursacht, dass weiche Echtzeitnachrichten und Nachrichten ohne Echtzeitanforderungen nach Prioritäten geplant und so nach ihrer Wichtigkeit in das andere Netz weitergeleitet werden. Dabei sichert die dynamische Planung der weichen Echtzeitnachrichen nach EDF zu, dass Nachrichten, welche nur noch einen geringen Abstand zu ihrer Frist haben, bevorzugt gesendet werden. Bei dieser Planung ist es somit möglich, dass Nachrichten geringerer Priorität von hochprioren Nachrichten verdrängt und nicht (rechtzeitig) übermittelt werden.

Für alle Konvertierungsfälle gilt, dass der Jitter der Nachricht nicht abhängig von der Anzahl der Ereigniskanäle, die in einem Konvertierungstask bearbeitet werden, ist. Ereigniskanäle erreichen aber, wenn sie einen eigenen Konvertierungstasks zugeordnet sind, geringere Ende-zu-Ende Verzögerungen, als für den Fall, dass der Konvertierungstask noch weitere Ereigniskanäle bearbeitet. Dies ist durch die längere Bearbeitungszeit im Konvertierungstask und der größeren Verzögerungen durch Länge und Anordnung der Zeitschlitze begründet.

Die Konvertierung von mehreren Ereigniskanälen in einem Task ist besonders für sporadische SRT-Nachrichten mit langen Fristen und NRT-Nachrichten aus dem CAN-Netz geeignet, weil sich diese Nachrichten einen dynamischen MTC teilen können. Die Verwendung von dynamischen MTC ermöglicht es sporadische Nachrichten aus dem CAN-Netz im TTP/C effizient und bandbreitensparend zu repräsentieren. Dies wird durch eine Mehrfachnutzung des Zeitschlitzes durch verschiedene Ereigniskanäle aus dem CAN-Netz erlaubt. Wird z.B. ein Ereigniskanal für eine sporadische Alarmmeldung mit definierter maximaler Verzögerung von CAN nach TTP/C gesendet, kann der Ereigniskanal einem dynamischen MTC zugeordnet werden. Dieser MTC muss so eingestellt werden, dass maximale Ende-zu-Ende Latenz keiner als die maximal erlaubte Verzögerung der Alarmmeldung ist. Um die Auslastung des MTCs zu erhöhen, wenn keine Alarmmeldungen auftreten, werden weitere Ereigniskanäle für diesen MTC registriert. Es muss jedoch zugesichert werden, dass der Ereigniskanal der Alarmmeldung die höchste Priorität im dynamischen MTC hat. Empfängt das Gateway eine Alarmmeldung, wird sie allen anderen Nachrichten vorgezogen und innerhalb der Periode weitergeleitet. Die Alarmmeldung erreicht dann rechtzeitig den Empfänger im TTP/C-Netz. Damit kann die sporadische Alarmmeldung mit effizienter Bandbreitenauslastung unterstützt werden. Durch die Pufferung von Nachrichten, kann eine eventuell durch die Alarmmeldung verdrängte Nachricht zu einem späteren Zeitpunkt gesendet werden. Die Verwendung von Puffern für Ereigniskanäle ist in beiden Richtungen erlaubt und ermöglicht die Überbrückung von Hochlastphasen, so dass die Wahrscheinlichkeit, dass wichtige Nachrichten durch Überlastphasen verloren gehen, verringert wird.

Kapitel 4

Umsetzung des Gateways

Dieser Teil der Arbeit beschreibt die Implementierung des in Kapitel 3 dargestellten Konzeptes für ein Gateway zwischen CAN und TTP/C. Im Rahmen dieses Prototyps wurden nur Ereigniskanäle ohne Echtzeitanforderungen realisiert. Harte und weiche Echtzeitereigniskanäle werden daher vom implementierten Gateway nicht unterstützt. Zu Beginn dieses Kapitels wird die eingesetzte Entwicklungsumgebung beschrieben. Der Abschnitt 4.2 gibt einen Überblick über das Gateway bevor in Abschnitt 4.3 auf die prototypische Implementierung eingegangen wird. Der Fokus liegt dabei auf den Schnittstellen für die Konfiguration und Nutzung des Gateways. Der Abschnitt 4.4 stellt die Verwendung der Ereigniskanäle in der TTP/C-Anwendung dar. Den Abschluss des Kapitels bildet eine Beschreibung der Eigenschaften der Implementierung.

4.1 Entwicklungsumgebung

Für die Entwicklung des Gateways wird ein TTP/C-Cluster verwendet. Der TTP/C-Cluster besteht aus vier Knoten. Jeder Knoten ist ein TTTech PowerNode PN312 mit einem Motorola MPC555 Prozessor, 1 MByte RAM und 4 MByte Flash-Speicher. Ein PowerNode kann an ein TTP/C-, CAN- oder TTP/A-Netz angeschlossen werden. Dazu verfügt der Knoten über einen TTP/C-Kommunikationscontroller (AS8202NF) und einen CAN-Kommunikationscontroller (Freescale TouCAN) mit 16 einstellbaren Nachrichtenpuffern. Die Knoten sind über das TTP/C-Netz als Cluster verbunden. Auf den Knoten wird das Betriebssystem TTTech ^{TTP}OS 4.10 verwendet. Die Erstellung des TTP/C-Kommunikationsplans und die Planung der Tasks auf den TTP/C-Knoten erfolgt mit den TTTech Tools $^{TTP}Plan$ 5.4 und $^{TTP}Build$ 5.4. Der Zugriff auf Timer-Funktionen sowie Ein- und Ausgänge wird durch die Erweiterung TTTech I/O Toolbox 2.4 ermöglicht. Die entwickelte Anwendung ist in C++ realisiert. Als Compiler kommt der Diab C++ Compiler 5.3.1 für MPC555 von Wind River zum Einsatz. Um die Anwendung auf die TTP/C-Knoten zu übertragen und die Nachrichten auf den TTP/C-Bus sichtbar zu machen, wird ein TTTech MonitorNode MN322 eingesetzt. Der Verkehr auf dem TTP/C-Bus kann mit TTTech $^{TTP}View$ 6.4 dargestellt werden. Die Knoten werden mit der TTTech Anwendung $^{TTP}Load$ 6.4 über das TTP/C-Netz geflasht.

4.2 Architektur des Gateways

Das Gateway ist auf einem Knoten im TTP/C-Cluster implementiert. Es ist mit dem CAN-Netz verbunden und kann somit als Mittler zwischen beiden Protokollen agieren. Als Element des TTP/C-Clusters ist das Gateway als zeitgesteuerte periodische Anwendung realisiert. Dies ist dadurch begründet, dass auf den TTP/C-Knoten ^{TTP}OS als Betriebssystem läuft. Dieses Betriebssystem verwendet grundlegend zeitgesteuerte Tasks für die Anwendungen. Neben den Tasks steht auch noch ein Idle-Task zur Verfügung, welcher aufgerufen wird, wenn kein anderer Task den Prozessor nutzt. Dieser Idle-Task kann z.B. für Wartungsaufgaben oder andere nicht zeitkritische Anwendungen genutzt werden. Das Gateway wird, wie in Abbildung 4.1 zeigt, in verschiedene zeitgesteuerte Tasks aufgeteilt.

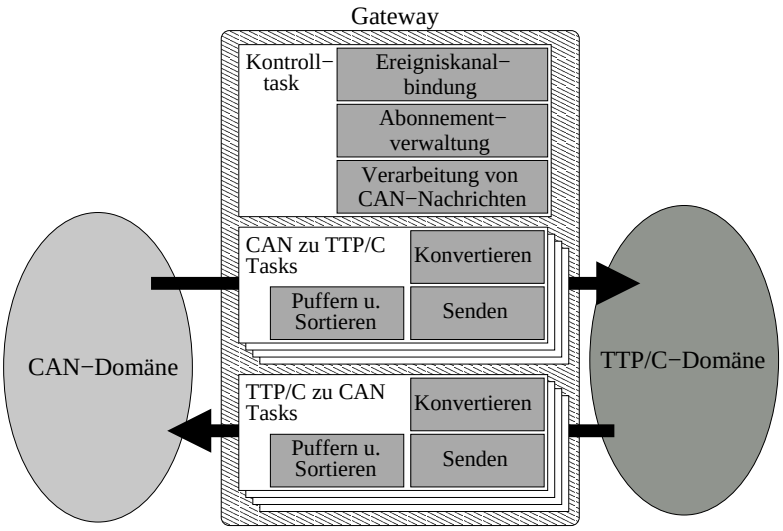


Abbildung 4.1: Gateway Task-Struktur

Das Gateway besteht grundlegend aus zwei Taskarten: Kontrolltasks und Konvertierungstasks. Ein Kontrolltask hat die Aufgaben die Abonnements in jedem Netz zu verwalten, eingehende CAN-Nachrichten zu empfangen, die CAN-Nachrichten den Konvertierungstasks zuzuordnen, den ECB im CAN-Netz zur Bindung der Ereigniskanäle zu kontaktieren und die ECB-Komponente im TTP/C-Netz zu realisieren. In einem Gateway genügt ein Kontrolltask zur Realisierung dieser Aufgaben. Es ist aber auch möglich die Aufgaben auf mehrere Kontrolltasks aufzuteilen, um z.B. die Aufgaben zeitlich besser anzuordnen. Die Konvertierungstasks gibt es in zwei Ausprägungen, wobei jeder Konvertierungsrichtung eine zugeordnet ist. Pro Richtung sind mehrere Tasks möglich, welche einen oder mehrere Ereigniskanäle zur Weiterleitung handhaben. Ihre Aufgaben sind es die Konvertierung der Nachrichten, deren Einordnung in die Ausgangswarteschlange unter Berücksichtigung der Nachrichtenprioritäten, die Wartung der Ausgangswarteschlange und das Senden der Nachrichten. Welche funktionalen Komponenten in beiden Taskarten verwendet werden, wurde bereits in Abschnitt 3.4.3 beschrieben.

Die Implementierung des Gateways besteht aus neun Hauptklassen. Dazu kommen weitere Klassen für die Datenhaltung, Warteschlangen, den Zugriff auf den CAN-Kommunikationscontroller und zur Verwendung des Timers. Diese Klassen werden im Rahmen der Arbeit nicht beschrieben. Die Abbildung 4.2 zeigt, dass Klassendiagramm der Hauptklassen.

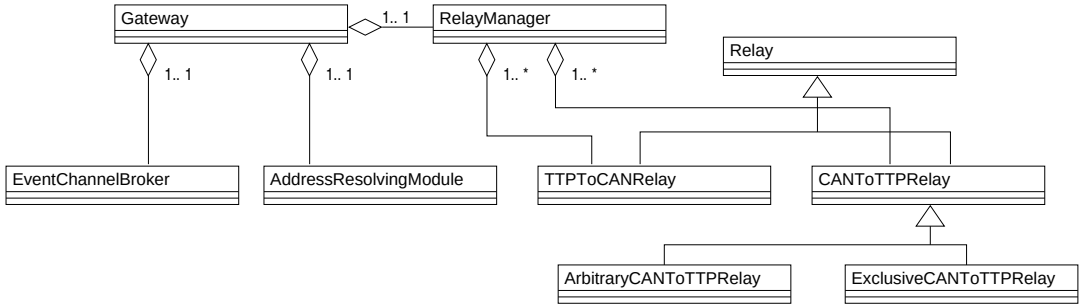


Abbildung 4.2: Schematische Klassenübersicht des Gateways

Die zentrale Klasse, welche alle Informationen hält oder zugänglich macht, ist die *Gateway*-Klasse. Sie handhabt das Senden und das Empfangen von CAN-Nachrichten und die Zuordnung von eingehenden CAN-Nachrichten zu den jeweiligen Konvertierungsobjekten. Damit übernimmt das *Gateway* die Filterung der eingehenden CAN-Ereigniskanäle. Die *Gateway*-Klasse verwaltet weiterhin den ECB (*EventChannelBroker*), die Komponente zur Auflösung der CAN-Event-Tags anhand der UID des Kanals (*AddressResolvingModule*) und die Verwaltung der Objekte zur Konvertierung der Ereigniskanäle (*RelayManager*).

Die *EventChannelBroker*-Klasse bietet die Methoden zum dynamischen Binden der Ereigniskanäle im TTP/C-Netz. Dazu verarbeitet sie die Kontrollnachrichten aller Knoten und sendet die Antworten über einen mit einer Warteschlange versehenen Kontrollzeitschlitz (CMS). Für die Kommunikation mit dem ECB des CAN-Netzes wird die *AddressResolvingModule*-Klasse verwendet. Sie fordert nach dem Start den TxNode-Knotenidentifier für das CAN-Netz an und stellt Bindungsanfragen an den CAN-ECB für Ereigniskanäle, die weitergeleitet werden sollen. Beide Module arbeiten mit der *RelayManager*-Klasse zusammen, welche die notwendigen Adressinformationen für die Adressumsetzung vermittelt und die Abonnements und Bereitstellungen für die Abonnentenverwaltung an die betreffenden Konvertierungsobjekte weitergibt. Eine Instanz der *Gateway*-Klasse hält eine Referenz zu je einer Instanz der drei anderen Verwaltungsklassen. Damit agiert ein *Gateway*-Objekt als Koordinator, welcher den Informationsaustausch zwischen dem *EventChannelBroker*, *AddressResolvingModule* und *RelayManager* organisiert. Es muss sichergestellt werden, dass für diese vier Verwaltungsklassen jeweils nur ein Objekt instantiiert wird, denn es erfolgt durch diese Klassen ein Zugriff auf Hardware (z.B. der CAN-Kommunikationscontroller oder der Timer) und auf global eindeutige Daten (z.B. die Bindungstabelle oder Nachrichtenwarteschlangen). Werden mehrere Objekte der Klassen instantiiert, kann es daher zum Fehlverhalten des Gateways oder zur Inkonsistenz der Daten kommen. Die vier Klassen implementieren daher das Singleton-Pattern [GHJV96].

Die *RelayManager*-Klasse hält für die Konvertierung der Nachrichten *CANToTTPRelay*- und *TTPToCANRelay*-Objekte. Sie übernehmen die Umsetzung der ihnen zugeordneten Ereigniskanäle, dazu gehört die Verwaltung der Ereigniskanäle, die Verwaltung der Abonnements, die Adressumsetzung, die Nachrichtenkonvertierung und -priorisierung sowie je nach Art der Konvertierung auch das Senden der Nachrichten. Die ausgehenden CAN-Nachrichten der *TTPToCANRelay*-Objekte werden an die globale CAN-Ausgangswarteschlange übergeben, welche von der *Gateway*-Klasse verwaltet wird. Die *CANToTTPRelay*-Objekte senden die Nachrichten autonom und halten für einen dynamischen Zeitschlitz (MTC) eine eigene Warteschlange. Die Vaterklasse (*Relay*) dieser Konvertierungsklassen lagert die Verwaltung der

zugeordneten Ereigniskanäle und die Adressumsetzung aus den Spezialisierungen aus. Die Abonnentenverwaltung kann nicht in die Vaterklasse ausgelagert werden, da die Zuweisung der Abonnements für beide Unterklassen unterschiedlich erfolgt. Es werden aber in der *Relay*-Klasse gemeinsame Schnittstellen definiert. Die *CANToTTPRelay*-Klasse verfügt über zwei Spezialisierungen, welche sich in der Nutzung des TTP/C-Ausgangszeitschlitzes unterscheiden. Die *ExclusiveCANToTTPRelay*-Klasse bietet für jeden registrierten Ereigniskanal einen festen Platz im Ausgangszeitschlitz und setzt damit eine Konvertierung mit exklusivem MTC (Fall A3 und A4) um. Eine Konvertierung mit dynamischem MTC (Fall A5 und A6) wird mit der *ArbitraryCANToTTPRelay*-Klasse, welche mittels einer Warteschlange den Ausgangszeitschlitz befüllt, ermöglicht. Die *TTPToCANRelay*-Klasse setzt die Konvertierung von TTP/C-NRT-Nachrichten mit globaler CAN-Warteschlange (Fall B3 und B4) um. Die verwendeten Konvertierungsverfahren sind im Unterabschnitt 3.4.4 erläutert.

Für die Implementierung des Kontrolltasks und der Konvertierungstasks sind unterschiedliche Schnittstellen für die beschriebenen Klassen definiert. Im Kontrolltask werden nur die Methoden des *Gateway*-Objekts für die Ausführung der Aufgaben benötigt. Innerhalb der Implementierung des Konvertierungstasks wird die Funktionalität des zugehörigen *Relay*-Objekt genutzt, um die Nachrichten weiterzuleiten.

4.3 Gateway-Implementierung

In diesem Abschnitt werden die Schnittstellen für die Nutzung des Gateways dargestellt. Es wird gezeigt, wie das Gateway konfiguriert und verwendet wird. Der Abschnitt gliedert sich in drei Teile. Der erste Unterabschnitt beschreibt die Initialisierung und Konfiguration des Gateway-Anwendung. Der zweite Unterabschnitt zeigt die Implementierung des Kontrolltasks. Der letzte Unterabschnitt stellt die Umsetzung der Konvertierungstasks dar.

4.3.1 Initialisierung und Konfiguration des Gateways

Beim Start des Knotens muss die Gateway-Anwendung zunächst initialisiert werden. Dazu wird das globale *Gateway*-Objekt mit den dazugehörigen drei Verwaltungsobjekten (siehe Abbildung 4.2) erstellt. Weiterhin werden *Relay*-Objekte erstellt, welche die Konvertierung der Ereigniskanäle realisieren. Im Beispiel aus Abbildung 4.3 wird ein *TTP2CANRelay*-(`ttp2canExpampleRelay`) und ein *CAN2TTPRelay*-Objekt (`can2ttpExpampleRelay`) erstellt. Für das Objekt der *CAN2TTPRelay*-Klasse muss entsprechend der Anwendung eine der beiden Spezialisierungen (*ArbitraryCAN2TTPRelay* oder *ExclusiveCAN2TTPRelay*) gewählt werden. Die Abbildung 4.3 zeigt die Initialisierung des Gateways und die Registrierung der beiden Konvertierungsobjekte.

Die Zeile ***5*** und ***6*** definieren die verwendeten UIDs und Attribute der weiterzuleitenden Ereigniskanäle. Als Attribute wird die Periode, der Echtzeitkanaltyp[1] und der Nachrichtentyp angegeben.

In der Zeile ***10*** wird für das *CAN2TTPRelay* der Sendezeitschlitz definiert. Der erste Parameter gibt die Adresse an, unter dem der Kommunikationscontroller die Nachrichten für die Übermittlung erwartet. Der zweite Parameter der *setSlot()*-Methode gibt die Breite des

[1] In dem implementierten Prototyp wird nur der NRT-Kanaltyp unterstützt.

```
 1: // Initialisierung des Knotens.
 2: void init (void)
 3: {
 4:         // Definition der UID der Ereigniskanäle. Setzen der Kanalattribute.
 5:         subject TTPSubject1, TTPSubject2, CANSubject1, CANSubject2;
 6:         EventChannelAttributeList evlist1, evlist2;
 7:         // [...]
 8:
 9:         // Initialisierung des CAN nach TTP/C Relays und des TTP/C nach CAN Relays.
10:         can2ttpExampleRelay.setSlot(&TTPSendSlot, 1);
11:         can2ttpExampleRelay.addManagedEventChannel(TTPSubject1, evlist1);
12:         can2ttpExampleRelay.addManagedEventChannel(TTPSubject2, evlist1);
13:         ttp2canExampleRelay.addManagedEventChannel(CANSubject1, evlist2);
14:         ttp2canExampleRelay.addManagedEventChannel(CANSubject2, evlist2);
15:
16:         // Initialisierung des Gateways: CAN-Modul, Timer, Sendezeitschlitz und Warteschlange.
17:         gw = Gateway::getInstance();
18:         //[...]
19:
20:         // Registrieren der Relays beim Gateway.
21:         gw->addTTPToCAN(&ttp2canExpampleRelay);
22:         gw->addCANToTTP(&can2ttpExampleRelay);
23: }
```

Abbildung 4.3: Gateway-Initialisierung

Zeitschlitzes an und bestimmt damit, wieviele Nachrichten auf einmal gesendet werden können. Soll eine Nachricht in dem Zeitschlitz gesendet werden, muss sie in diesen Speicherbereich gelegt werden, da der Kommunikationscontroller, immer wenn der Zeitschlitz beginnt, den Inhalt dieses Speicherbereichs sendet. Beim Lesen von Nachrichten aus Zeitschlitzen ist dieses Verfahren umgekehrt. Die Anwendung erhält die Nachrichten dann über den lesenden Zugriff auf einen Speicherbereich. Die Abbildung 2.4 aus Unterabschnitt 2.3.5 zeigt diesen Zusammenhang im Detail. Zur Vereinfachung der Zuweisung der Nachrichteninhalte und der Implementierung, wird in der Realisierung des Gateways jede TTP/C-Nachricht (siehe Abbildung 3.5) mit 8 Byte Nutzlast ausgestattet. Kürzere Nachrichten sind im Prototyp nicht vorgesehen, denn für die Unterstützung von Nachrichten mit unterschiedlichen Längen muss für jede Nachrichtenlänge ein eigener Datentyp in den Tools $^{TTP}Load$ und $^{TTP}Build$ vorgesehen werden. Diese Datentypen müssen, dann auch in der Gateway-Anwendung genutzt werden, da die Übergabe der Nachrichten an die Zeitschlitze an den Datentyp gebunden ist.

Die Zeilen ***11-14*** weisen den `Relay`-Objekten die UIDs der Ereigniskanäle und deren Attribute zu. Damit wird die Filterung der Kanäle definiert und festgelegt, welche Ereigniskanäle das Gateway passieren und welche `Relay`-Objekte dafür zuständig sind. Nur bei den `Relay`-Objekten registrierte Ereigniskanäle können weitergeleitet werden. Die `Relay`-Objekte werden in den Zeilen ***21*** und ***22*** beim `Gateway`-Objekt registriert. Es übergibt dabei die Referenzen der Objekte an das `RelayManager`-Objekt zur Verwaltung.

Die Initialisierung des `Gateway`-Objektes in Zeile ***17*** umfasst mehrere nicht näher dargestellte Schritte. So werden die Objekte für die Kommunikation über das CAN-Protokoll und den Zugriff auf der CAN-Kommunikationscontroller erzeugt und konfiguriert. Weiterhin sind Objekte für die globale CAN-Warteschlange und den Timer zur Messung von Zeiten zu erzeugen.

Auch die Referenz des Kontrollzeitschlitzes zum Senden von Antworten der ECB-Komponente im Gateway wird dem *Gateway*-Objekt zugewiesen.

Die Initialisierung der Objekte für die Gateway-Anwendung ermöglicht eine einfache und kompakte Formulierung des Kontrolltasks und der Konvertierungstasks.

4.3.2 Kontrolltask

Der Kontrolltask übernimmt die in Abbildung 4.1 dargestellten Aufgaben und verwaltet die Abonnements, führt die Bindung der Ereigniskanäle durch, empfängt CAN-Nachrichten und ordnet sie den *Relay*-Objekten zu. Für diese Funktionen wird in der Gateway-Anwendung nur die Instanz der *Gateway*-Klasse genutzt. Sie ermöglicht den Zugriff auf alle relevanten Daten zur Realisierung dieser Funktionen. Der Kontrolltask wird periodisch aufgerufen und erlaubt daher die Abarbeitung der Aufgaben mit einer festen Verzögerung. Das folgende Code-Beispiel in der Abbildung 4.4 verdeutlicht die Verwendung des *Gateway*-Objektes im Kontrolltask.

```
 1: tt_task (ttControlTask)
 2: {
 3:         // Verarbeitung der Kontrollnachrichten der TTP/C-Knoten.
 4:         gw->handleTTPRequestMessage(&NodeBRqSlot);
 5:         // [...]
 6:
 7:         // Empfang und Zuordnung neuer CAN-Nachrichten.
 8:         gw->fetchNewCANMessages();
 9:
10:         // Verarbeitung von CAN-Kontrollnachrichten.
11:         gw->handleCANRequestMessages();
12:
13:         // Senden von TTP/C-Kontrollnachrichten.
14:         gw->sendTTPRequestMessages();
15:
16:         // Senden der CAN-Kontrollnachrichten.
17:         gw->sendCANRequestMessages();
18: }
```

Abbildung 4.4: Gateway Kontrolltask

Die Zeile *4* zeigt die Verarbeitung von eingehenden Kontrollnachrichten (von *NodeB*). Der Übergabe-Parameter dieser Methode ist die Speicheradresse, in der die Nachrichten aus dem Kontrollzeitschlitz des Knotens vom Kommunikationscontroller abgelegt werden. Jeder Knoten im TTP/C-Cluster hat einen Kontrollzeitschlitz für Nachrichten zum Gateway. Das *Gateway*-Objekt muss daher für den Kontrollzeitschlitz jedes Knotens die Methode *handleTTPRequestMessage()* ausführen und dann die entsprechenden Anforderungen der TTP/C-Knoten verarbeiten. Die Verarbeitung der TTP/C-Kontrollnachrichten wird durch den *EventChannelBroker* realisiert. Bei erfolgreicher Bindung eines Ereigniskanals oder des Rückzugs eines Abonnements bzw. einer Ankündigung werden entsprechende Informationen über das *RelayManager*-Objekt an die betreffenden *Relay*-Objekte übergeben. Sie speichern die Informationen in der Abonnentenverwaltung und Adressumsetzungstabelle ab. Beim Empfang eines Abonnements für einen CAN-Ereigniskanal, wird das *AddressResolvingModule*-Objekt vom ECB beauftragt das Event-Tag dieses Kanals im CAN-Netz beim CAN-ECB zu erfragen.

Der Kontrolltask erhält mit dem Aufruf in Zeile ***8*** alle CAN-Nachrichten, welche nach dem letzten Leseaufruf vom CAN-Kommunikationscontroller empfangen wurden. Durch die periodische Ausführung des Kontrolltasks wird das Empfangen der CAN-Nachrichten im Gateway mittels periodischen Lesens der Puffer des CAN-Kommunikationscontrollers (`fetchNewCANMessages()`) realisiert. Dies kann dazu führen, dass CAN-Nachrichten, durch Überschreiben der Puffer im CAN-Kommunikationscontroller zwischen den Leseaufrufen, verpasst werden.

Im Normalfall werden CAN-Nachrichten über Empfangsinterrupts des Kommunikationscontrollers in den Speicher der Anwendung geholt. Doch aufgrund der zeitgesteuerten Anwendung, kann bei der Nutzung von Interrupts nicht sichergestellt werden, dass keine Fristen der Tasks verletzt werden. Insbesondere in Phasen in denen überdurchschnittlich viele CAN-Nachrichten empfangen werden, kann dies zu einer nicht vernachlässigbaren Verzögerung der Tasks führen und das Gateway kann dann z.B. seine TTP/C-Zeitschlitze nicht rechtzeitig bedienen. Verpasst das Gateway Zeitschlitze, in denen es senden muss, wird es von den anderen TTP/C-Knoten von der Kommunikation ausgeschlossen (Membership-Dienst). In diesem Fall passieren keine Nachrichten mehr das Gateway und wichtige Ereigniskanäle werden nicht zugestellt. Wenn die Interrupts auf eine maximalen Häufigkeit begrenzt werden, kann die maximale Häufigkeit zur Berechnung von Taskzeiten herangezogen werden. Die Tasks werden dann zwar unterbrochen, aber durch die Dimensionierung der Taskzeit unter Berücksichtigung der maximalen Interrupthäufigkeit, kann jeder Task seine Fristen garantiert einhalten. Die Möglichkeit die Interrupts in ihrer Auftrittsfrequenz durch das Betriebssystem zu begrenzen bietet ^{TTP}OS (siehe [TTT05b]) zur Zeit nicht an.

Für das Empfangen der CAN-Nachrichten werden daher keine Interrupts verwendet. Um die Wahrscheinlichkeit des Verlustes von Nachrichten zu minimieren, wird eine CAN-Pufferverwaltung implementiert. Sie weist den wichtigsten Ereigniskanälen eigene Lesepuffer zu, womit sich die Wahrscheinlichkeit verkleinert, dass Nachrichten ungelesen überschrieben werden. Da der CAN-Kommunikationscontroller eine begrenzte Zahl von Puffern besitzt, können nicht beliebig viele Ereigniskanäle einen eigenen Puffer erhalten. Werden mehr Ereigniskanäle vom Gateway benötigt als Puffer zur Verfügung stehen, müssen alle überzähligen Kanäle in einen allgemeinen Puffer abgelegt werden.

Bei hoher CAN-Buslast muss die Lesefrequenz der CAN-Nachrichten erhöht werden, um keine Nachricht zu verpassen. Zur optimalen Einstellung der Lesefrequenz kann der Aufruf der `fetchNewCANMessages()`-Methode auch in anderen Tasks auf dem Gateway ausgeführt werden. Aufgrund der zeitlichen Eigenschaften der Ereigniskanäle, kann die benötigte Lesefrequenz für die Nachrichten der Ereigniskanäle ermittelt werden. Die Kommunikation der CAN-Knoten mit dem CAN-ECB unterliegt aber keinen zeitlichen Beschränkungen. Es ist daher möglich, dass das Gateway, durch zu schnell aufeinanderfolgende Anfragen von CAN-Knoten an den CAN-ECB und dessen Antworten, Kontrollnachrichten nicht erhält. Eine Lösung dieses Problem ist es in die Kommunikation mit dem CAN-ECB künstliche Verzögerungszeiten einzuarbeiten. Bei diesem Ansatz bleibt es zu untersuchen, wie sich der CAN-Nachrichtenempfang bei höher Netzbelastung und einer Vielzahl Ereigniskanälen verhält und unter welchen Bedingungen CAN-Nachrichten im Gateway verloren gehen.

Mit dem Methodenaufruf in Zeile ***11*** verarbeitet das Gateway alle empfangenen Kontrollnachrichten aus dem CAN-Netz. Dazu werden die Nachrichten von dem `AddressResolvingModule`-Objekt überprüft. Beim erfolgreichen Abonnement eines TTP/C-Ereigniskanals gibt

das *AddressResolvingModule*-Objekt die Adress- und Abonnementinformationen über das *RelayManager*-Objekt an das betreffende *Relay*-Objekt weiter. Das *Relay*-Objekt kann dann den abonnierten Kanal ins CAN-Netz weiterleiten.

Für die Übermittlung ausstehender Kontrollnachrichten kontaktiert das *Gateway*-Objekt das *EventChannelBroker*-Objekt und das *AddressResolvingModule*-Objekt. Die TTP/C-Kontrollnachrichten werden in Zeile **14** vom *EventChannelBroker*-Objekt gesendet. Das *AddressResolvingModule*-Objekt übermittelt in Zeile **17** die Kontrollnachrichten an den CAN-ECB. In beiden Methoden wird pro Periode jeweils eine Kontrollnachricht gesendet. Dazu entnimmt die Übermittlungsmethode die erste Nachricht aus dem zugehörigen Kontrollnachrichtenpuffer.

4.3.3 Konvertierungstask

Als Konvertierungstask wird ein Task bezeichnet, welcher die Nachrichtenkonvertierung, die Priorisierung, die Pufferung und das Sendens der Nachrichten übernimmt. Dabei wird einem Konvertierungstask mindestens ein *Relay*-Objekt zugeordnet.

Die Verwendung der *Relay*-Objekte in den Konvertierungstasks ist aufgrund der vorherigen Konfiguration einfach gehalten. Bei der Nutzung der *Relay*-Objekte muss zwischen beiden Konvertierungsrichtungen unterschieden werden.

TTP/C-zu-CAN-Konvertierungstask

In einem TTP/C-zu-CAN-Konvertierungstask wird mit einem *TTPToCANRelay* die Weiterleitung von registrierten TTP/C-Ereigniskanälen in das CAN-Netz realisiert. Die Verwendung des *TTPToCANRelay*-Objektes zeigt das folgende Beispiel in Abbildung 4.5.

```
 1: // Konvertierungstask des TTP/C-zu-CAN-Relays.
 2: tt_task (ttTTP2CANConversionTask)
 3: {
 4:         // Hinzufügen und Verarbeiten der TTP/C-Nachrichten.
 5:         ttp2canExampleRelay.addIncommingMessages(&Slot1, 1);
 6:         ttp2canExampleRelay.addIncommingMessages(&Slot2, 1);
 7:
 8:         // Senden der konvertierten Nachrichten durch die globale CAN-Warteschlange.
 9:         ttp2canExampleRelay.sendOutput();
10: }
```

Abbildung 4.5: Gateway Konvertierungstask für TTP/C-zu-CAN-Relay

Zu konvertierende TTP/C-Nachrichten werden an das *TTPToCANRelay*-Objekt über die Adresse des Speicherbereichs, in dem der entsprechende Zeitschlitz zu finden ist, übergeben. Dabei prüft das Konvertierungsobjekt zunächst, ob Nachrichten, die zu einem registrierten Ereigniskanal passen, im Zeitschlitz enthalten sind. Gehört eine Nachricht zu einem vom *Relay*-Objekt zur Weiterleitung freigegebenen Ereigniskanal, wird sie übersetzt und in die globale CAN-Warteschlange eingefügt. Der dazu notwendige Aufruf ist in Zeile **5** und **6** dargestellt. Der Aufruf wird zweimal ausgeführt, weil das Konvertierungsobjekt zwei Ereigniskanäle, welche

sich in unterschiedlichen Zeitschlitzen befinden, weiterleitet. Die *addIncommingMessages()*-Methode muss für jeden relevanten Zeitschlitz des jeweiligen *Relay*-Objektes aufgerufen werden. Der erste Parameter der Methode bezeichnet die Adresse des Zeitschlitzes und der zweite dessen Breite.

Das Senden der Ausgangsnachrichten wird mit dem Aufruf in Zeile **9** durchgeführt. Dazu werden die Nachrichten in der globalen CAN-Warteschlange zuerst priorisiert und ggf. sortiert. Die wichtigste Nachricht wird mit dem Aufruf sofort gesendet. Der Methodenaufruf *sendOutput()* des *TTPToCANRelay*-Objektes wird dabei an den Sendeaufruf der globalen Warteschlange weitergereicht. D.h. es ist nicht sichergestellt, dass eine Nachricht, welche von diesem Konvertierungsobjekt übersetzt wurde, auch gesendet wird, da eventuell andere Nachrichten höher priorisiert sind. Die Häufigkeit, mit der der Sendebefehl aufgerufen wird, richtet sich nach dem Aufkommen der übersetzten Nachrichten. Werden in diesem Beispiel in jeder Periode zwei Nachrichten in der CAN-Warteschlange abgelegt aber nur eine gesendet, läuft der Puffer schnell über und Nachrichten gehen verloren. In diesem Fall muss der Aufruf aus Zeile **9** an einer anderen Stelle, in diesem oder in einem anderen Task, erneut ausgeführt werden. Treten jedoch die beiden Nachrichten aus dem Beispiel nur jede zweite Periode auf, so genügt ein Aufruf der *sendOutput()*-Methode. Anstatt den Sendeaufruf im Konvertierungstask aufzurufen ist es auch möglich die CAN-Nachrichten direkt über die globale CAN-Warteschlange zu senden. Dazu kann ein periodischer Sendetask, welcher nur die *sendOutput()*-Methode der Warteschlange (*GlobalCanOutqueue*) aufruft, eingeplant werden.

CAN-zu-TTP/C-Konvertierungstask

In einem CAN-zu-TTP/C-Konvertierungstask müssen die Nachrichten der Ereigniskanäle nicht hinzugefügt werden, da bereits durch den Kontrolltask beim Empfang von CAN-Nachrichten (*fetchNewCANMessages()*) eine Zuordnung zu den entsprechenden *CANToTTPRelay*-Objekten vorgenommen wird. Bei der Zuordnung werden die CAN-Nachrichten in der Eingangswarteschlange des *CANToTTPRelay*-Objektes platziert. Die Nachrichten der registrierten Ereigniskanäle müssen daher im CAN-zu-TTP/C-Konvertierungstask nur konvertiert und dann gesendet werden.

```
1: // Konvertierungstask des CAN-zu-TTP/C-Relays.
2: tt_task (ttCAN2TTPConversionTask)
3: {
4:         // Konvertieren und Platzieren der der Nachrichten in der globalen CAN-Warteschlange.
5:         can2ttpExampleRelay.convertIncommingEventMessages();
6:         // Sortieren, Priorisieren und Senden der Nachrichten im TTP/C-Zeitschlitz.
7:         can2ttpExampleRelay.sendOutput();
8: }
```

Abbildung 4.6: Gateway Konvertierungstask für CAN-zu-TTP/C-Relay

Der Aufruf im Beispiel aus Abbildung 4.6 in Zeile **5** konvertiert die Nachrichten aus der Eingangswarteschlange in TTP/C-Nachrichten und fügt sie nach ihrer Priorität in die Ausgangswarteschlange ein. Das Hinzufügen zu diesem Ausgangspuffer wird für die Spezialisierungen (*ArbitraryCANToTTPRelay* und *ExclusiveCANToTTPRelay*) der *CANToTTPRelay*-Klasse unterschiedlich realisiert. Für die Unterschiede beider Konvertierungsarten sei auf den Unter-

abschnitt 3.4.4 verwiesen. Die Nachrichten in dem Ausganspuffer werden mit dem Sendeaufruf in Zeile ***7*** mit aktuellen Prioritäten ausgestattet, sortiert und an den Sendezeitschlitz übergeben. Die Nachrichten, die nicht in den Sendezeitschlitz passen, verbleiben in der Ausgangswarteschlange.

4.4 Ereigniskanäle

Dieser Abschnitt veranschaulicht die Änderungen an einer beispielhaften TTP/C-Anwendung, welche sich durch die Nutzung der COSMIC-Middleware ergeben. Dazu wird speziell auf die Initialisierung und Nutzung der Ereigniskanäle zum Nachrichtenaustausch eingegangen.

Initialisierung der Ereigniskanäle

Für die Initialisierung eines Ereigniskanals (`initializeChannel()`) ist die UID des Kanals, die Attribute (Echtzeittyp, Periode und die Festlegung auf Ereignis- oder Statusnachrichten), die Richtung (Abonnement oder Publikation) und Informationen über die notwendigen TTP/C-Zeitschlitze anzugeben. Ein Ereigniskanal benötigt dabei drei Zeitschlitze: einen zum Senden bzw. Empfangen von Nachrichten (je nach Richtung des Ereigniskanals) und jeweils einen Zeitschlitz (CMS) zum Senden und Empfangen von Kontrollnachrichten für die Bindung des Ereigniskanals. Wird der Ereigniskanal für das Empfangen konfiguriert, muss für den Empfangszeitschlitz der Nachrichten die Breite definiert werden. Dadurch wird eingestellt, wieviele Nachrichten in einem Zeitschlitz enthalten sind. Im Gegensatz dazu kann ein Sendezeitschlitz einer Anwendung nur die Breite eins verwenden, da ein Kanal pro Periode nur eine Nachricht publizieren kann. Der Kontrollzeitschlitz für Anfragen an das Gateway kann von mehreren Ereigniskanälen genutzt werden. Dabei muss sichergestellt werden, dass nicht mehrere Ereigniskanäle gleichzeitig in diesen Zeitschlitz schreiben. In der Regel genügt es, wenn pro Knoten ein Kontrollzeitschlitz in Richtung Gateway verwendet wird. Das Gateway muss seinerseits mit der `handleTTPRequestMessage()`-Methode die von den Knoten verwendeten Kontzollzeitschlitze nach Anforderungen abfragen. Der Kontrollzeitschlitz, welcher vom Gateway gesendet wird, muss in jedem Knoten für jeden Ereigniskanal bekannt sein. Die Abbildung 4.7 zeigt die schematische Initialisierung der Ereigniskanäle.

Bindung der Ereigniskanäle

Für das Binden der Ereigniskanäle sollte ein eigener Task vorgesehen werden, da hierbei alle Ereigniskanäle den selben Kontrollzeitschlitz für Bindungsanfragen an das Gateway nutzen können. Die Bindung der Ereigniskanäle erfolgt je nach Richtung über die `announce()`- oder `subscribe()`-Methode des Ereigniskanals.

Das Beispiel in Abbildung 4.8 zeigt, dass die Bindung der Ereigniskanäle aus der Verarbeitung eingehender Kontrollnachrichten (Zeile ***5*** und ***6***), sowie aus dem Senden der Anforderung nach der Bindung des Ereigniskanals (Zeile ***11*** und ***13***) besteht. Die Übermittlung der Kontrollnachrichten ist dabei davon abhängig, ob der Kanal bereits ein Event-Tag hat und in welcher der Reihenfolge (Zeile ***10*** bis ***13***) die Bindunganfragen gestellt werden. Sind alle Ereigniskanäle gebunden, werden keine Kontrollnachrichten mehr gesendet (Zeile ***15***). Vom Gateway gesendete Kontrollnachrichten werden auch, wenn alle Ereigniskanäle gebunden sind, verarbeitet, da

```
 1: // Ereigniskanäle für Abonnement und Publikation.
 2: EventChannel eChannelSub, eChannelPub;
 3:
 4: // Initialisierung des Knotens.
 5: void init (void)
 6: {
 7:         // Definition der UID der Ereigniskanäle. Setzen der Kanalattribute.
 8:         subject ecs, ecp;
 9:         EventChannelAttributeList evlist;
10:         // [...]
11:
12:         // Initialisierung der Kanäle: Setzen der Richtung, der Zeitschlitze
13:         // für Ereigniskanalnachrichten und Kontrollnachrichten.
14:         eChannelSub.initializeChannel(eChannelSub, ecs, ... DIRECTION_SUBSCRIBE, ...);
15:         eChannelPub.initializeChannel(eChannelPub, ecp, ... DIRECTION_PUBLISH, ...);
16: }
```

Abbildung 4.7: Initialisierung eines TTP/C-Ereigniskanal

```
 1: // Task für die Bindung der Ereigniskanäle.
 2: tt_task (ttBindEC)
 3: {
 4:         // Verarbeitung der TTP/C-Kontrollnachrichten vom Gateway.
 5:         eChannelSub.handleRequestMessage();
 6:         eChannelPub.handleRequestMessage();
 7:
 8:         // Übermittlung der Kontrollnachrichten für die Bindung,
 9:         // wenn der Ereigniskanal noch nicht gebunden ist.
10:         if(!eChannelSub.hasEtag())
11:                 eChannelSub.subscribe();
12:         else if(!eChannelPub.hasEtag())
13:                 eChannelPub.announce();
14:         else
15:                 //... Nichts senden ...
16: }
```

Abbildung 4.8: Bindung eines TTP/C-Ereigniskanal

das Gateway die Bindung der Ereigniskanäle wieder aufheben oder die Neubindung aller Ereigniskanäle befehlen kann. Daher muss jeder Ereigniskanal die Kontrollnachrichten des Gateways durchgängig, mit den in Zeile ***5*** und ***6*** dargestellten Aufrufen, überwachen.

Lesen aus einem Ereigniskanal

In der Abbildung 4.9 wird die lesende Nutzung eines abonnierten Ereigniskanals beschrieben. Dabei befindet sich die eigentliche Anwendung des Tasks in Zeile ***11***. Der Rest des Beispieles zeigt die zusätzlichen Aufrufe, welche durch die Nutzung des Ereigniskanals hinzukommen.

In Zeile ***8*** wird geprüft, ob der Kanal gebunden ist und damit genutzt werden kann. Ist der Kanal gebunden, wird versucht eine Nachricht[2] aus dem Zeitschlitz zu lesen (Zeile ***10***). Der

[2]Der Bezeichner `EventMessage` bezeichnet eine getypte Nachricht und ist unabhängig davon, ob es sich um eine Ereignis- oder Statusnachricht handelt. Diese Klasse wird für beide Typen genutzt.

```
1:  // Task, welcher einen abonnierten Ereigniskanal nutzt.
2:  tt_task (ttConsumEC)
3:  {
4:          // Nachricht, die empfangen wird.
5:          EventMessage em;
6:
7:          // Überprüfung, ob der Kanal gebunden ist.
8:          if(eChannelSub.hasEtag())
9:                  // Holen der Nachricht aus dem Kanal.
10:                 if(eChannelSub.fetchEventMessage(&em))
11:                         // ... Verarbeitung ...
12:                 else
13:                         // ... Es ist keine Nachricht, die zu dem
14:                         //      Ereigniskanal passt, gelesen worden ...
15:         else
16:                 // ... Wenn der Kanal ungebunden ist, kann er nicht genutzt werden ...
17: }
```

Abbildung 4.9: Lesen aus einem TTP/C-Ereigniskanal

Aufruf schlägt fehl, wenn in dem zugeordneten Zeitschlitz des Ereigniskanals keine Nachricht enthalten ist, z.B. wenn Nachrichten nur sporadisch über einen Ereigniskanal aus dem CAN-Netz gesendet werden. Die Methode `fetchEventMessage()` liefert weiterhin nur eine Ereignisnachricht zurück, wenn das ETB (siehe Unterabschnitt 3.4.1) anzeigt, dass die Nachricht neu ist. Damit wird die Durchsetzung des einmaligen Empfanges (*exactly-once*) von Ereignisnachrichten transparent durch den Ereigniskanal zugesichert. Durch die Mitteilung des Ereigniskanals, ob eine neue Nachricht empfangen wurde, muss die Anwendung dies nicht mehr selbst überprüfen und wird dadurch vereinfacht. Das ETB wird bei Statusnachrichten nicht beachtet. Wurde eine Nachricht korrekt empfangen, kann sie verarbeitet (Zeile ***11***) werden, ansonsten kann die Anwendung ohne die Verarbeitung von empfangenen Nachrichten ihre Aufgabe erledigen (Zeile ***13***).

Handelt es sich bei dem zu lesenden Zeitschlitz, um einen Zeitschlitz in dem mehrere Nachrichten gleichzeitig gesendet werden können, liefert die `fetchEventMessage()`-Methode nur die erste passende Nachricht. Sollen alle zum Ereigniskanal passenden Nachrichten bereitgestellt werden, muss die `fetchAllEventMessages()`-Methode des Ereigniskanals genutzt werden.

Publizieren in einen Ereigniskanal

Die Übermittlung von Nachrichten durch einen Ereigniskanal wird über die Methode `publish()` realisiert. Die Abbildung 4.10 zeigt wieder einen Anwendungstask, bei dem der Anwendungscode in Zeile ***9*** steht.

Wie bei dem Empfang von Nachrichten über einen Ereigniskanal, muss zuerst geprüft werden, ob der Kanal einem Event-Tag zugeordnet ist. Nur in korrekt gebundene Ereigniskanäle kann eine Nachricht publiziert werden. Die Abfrage in Zeile ***8*** sichert dies zu. Ist der Ereigniskanal nicht gebunden, muss eine leere Nachricht gesendet werden. Der Aufruf in Zeile ***13*** schreibt daher eine leere Nachricht in die Speicheradresse, so dass der Kommunikationscontroller die leere Nachricht auf den TTP/C-Bus legt. Die `publish()`-Methode in Zeile ***10*** übergibt die Nachricht an den Ereigniskanal. Dieser konvertiert das Nachrichten-Objekt in das Format für

```
1: // Task, welcher in einen Ereigniskanal publiziert.
2: tt_task (ttProduceEC)
3: {
4:         // Zu sendende Nachricht.
5:         EventMessage em;
6:
7:         // Überprüfung, ob der Kanal gebunden ist.
8:         if(eChannelPub.hasEtag())
9:                 // ... Verarbeitung, um Nachrichteninhalt zu erzeugen ...
10:                eChannelPub.publish(&em, true);
11:        else
12:                // Kanal ist ungebunden, der Zeitschlitz bleibt leer.
13:                eChannelPub.publishEmptyMessage();
14: }
```

Abbildung 4.10: Publizieren in einen TTP/C-Ereigniskanal

TTP/C-Nachrichten. Der zweite Parameter der Methode gibt an, ob es sich bei der Nachricht um eine neue Nachricht oder um eine Wiederholung handelt. Neue Nachrichten werden durch den Wert `true` und Wiederholungen durch `false` gekennzeichnet.

4.5 Eigenschaften der Implementierung

Die vorgestellte Umsetzung eines Gateways zwischen CAN und TTP/C bietet neben der flexiblen Verwendung von Objekten zur Weiterleitung von Nachrichten auch eine Erweiterung der TTP/C-Kommunikation um die COSMIC-Middleware an. Die prototypische Implementierung des Gateways erlaubt die Weiterleitung von NRT-Ereigniskanälen in beide Netzwerke. Der Aufbau des Gateways ist dabei so realisiert, dass eine spätere Erweiterung der Implementierung um Echtzeitkanäle möglich ist.

Die Weiterleitung der Ereigniskanäle kann sehr flexibel gestaltet werden, so können z.B. mehrere Ereigniskanäle aus dem CAN-Netz mit einem Zeitschlitz ins TTP/C-Netz eingeleitet werden. Dabei sind je nach Anforderung an Dringlichkeit und Latenz der Nachrichten zwei verschiedene Strategien der Zeitschlitzbelegung möglich. Einem Ereigniskanal steht entweder bei Verwendung eines *`ExclusiveCANToTTPRelay`*-Objektes ein Platz in jeder Periode im Ausgangszeitschlitz zu oder es wird nach der Priorität entschieden, welche Nachrichten gesendet werden. Dies ist durch die *`ArbitraryCANToTTPRelay`*-Klasse realisiert. Für die Weiterleitung von TTP/C-Ereigniskanälen ins CAN-Netz wird die *`TTP2CANRelay`*-Klasse genutzt. Sie verwendet zur Übermittlung aller Ausgangsnachrichten in das CAN-Netz eine globale priorisierte CAN-Warteschlange. Die globale Warteschlange sichert zu, dass immer die wichtigsten Nachrichten aus dem TTP/C-Netz ins CAN-Netz gesendet werden.

Die Konfiguration des Gateways erfolgt zum Einen im Code und zum Anderen in der Planung der Zeitschlitze und Tasks. Der Code definiert die zeitlichen und semantischen Eigenschaften der Ereigniskanäle und konfiguriert die *`Relay`*-Objekte. Die Konfiguration der *`Relay`*-Objekte legt fest, welche Ereigniskanäle das Gateway passieren dürfen und welchen Zeitschlitzen diese Kanäle zugeordnet sind. Die eigentliche Planung der zeitgesteuerten Kommunikation und der Tasks wird außerhalb des Anwendungscodes vorgenommen. Die Erzeugung des globalen Kommunikationplans im TTP/C wird über das Tool $^{TTP}Plan$ ermöglicht. Die Konfiguratio-

nen der zeitgesteuerten Tasks für jeden Knoten werden daraufhin mit $^{TTP}Build$ erstellt und daraus Code-Stubs generiert. Die Planung der Kommunikation und der Tasks legt fest, mit welchen Frequenzen die Zeitschlitze gesendet und die Tasks aktiviert werden. Die Definition der Sendefrequenzen von Zeitschlitzen bestimmt maßgeblich die Eigenschaften des Ereigniskanals. So kann einem Ereigniskanal mit einer festgelegten Periode keinem Zeitschlitz mit einer größeren Periode zugeordnet werden, ohne dass Nachrichten durch einen Überlauf der Warteschlange verloren gehen. Weiterhin wird in der Planung festgelegt, wie groß die Phasenverschiebungen zwischen den Zeitschlitzen und den Tasks sind. Unter Beachtung der Eigenschaften der Planung kann für die höchstprioren Nachrichten eine feste minimale Ende-zu-Ende Latenz berechnet werden. Für die Bestimmung der Latenz wird auf die Betrachtungen aus Abschnitt 3.4 verwiesen.

Für den Aufbau eines Gateways muss die Konfiguration der Kommunikation und Tasks im globalen Plan und die Zuordnung der Eigenschaften der Ereigniskanäle und `Relay`-Objekte in der Gateway-Anwendung durchgeführt werden. Es ist dabei sinnvoll zuerst die Kommunikation und die zeitgesteuerten Tasks zu planen und dann die Konfiguration und Implementierung der Gateway-Anwendung im Code zu gestalten. Die Erstellung einer Gateway-Anwendung ist ohne größeren Aufwand realisierbar und über die dargestellten Schnittstellen einfach zu konfigurieren. Es werden damit die Anforderungen an eine einfache Integration des Gateways in eine TTP/C-Anwendung erfüllt.

Die Verwendung der COSMIC-Middleware in der TTP/C-Anwendung erfordert durch die Nutzung der Ereigniskanäle in der Applikation einen kleinen Überbau. Die TTP/C-Anwendung profitiert aber dafür von den Vorteilen des Ereigniskanalkonzeptes. In der gesamten TTP/C-Anwendung werden die UIDs der Ereigniskanäle verwendet. Dies ermöglicht eine globale Adressierung der Informationen. Durch die Transparenz ist der Anwendung nicht bekannt, woher die Nachrichten dieser Kanäle kommen. Ein Ereigniskanal kann dabei im TTP/C-Netz oder, durch das Gateway weitergeleitet, im CAN-Netz bereitgestellt werden. Mit Hilfe der dynamischen Bindung der Ereigniskanäle können die Zeitschlitze zur Laufzeit der TTP/C-Anwendung mit anderen Ereigniskanälen belegt werden. So kann z.B. der Bereitsteller eines Ereigniskanals die Bereitstellung zurückziehen und einen anderen Ereigniskanal in dem frei gewordenen Zeitschlitz[3] anbieten. Dies ist vor allem für Debug- oder andere unkritische Informationen sinnvoll. Die dynamische Bindung der Ereigniskanäle erlaubt den Knoten auch Nachrichten für bestimmte Ereigniskanäle aus dynamischen Zeitschlitzen herauszufiltern. Damit wird eine gemeinsame Nutzung eines Zeitschlitzes von verschiedenen Ereigniskanälen und somit eine höhere Auslastung des Zeitschlitzes erreicht. Dies macht eine effektive Unterstützung von sporadischen Nachrichten möglich. Da in der TTP/C-Anwendung im Allgemeinen nur periodische Nachrichten erzeugt werden, ist diese Unterstützung speziell für das Gateway von Bedeutung, um der Anwendung im TTP/C-Netz die sporadischen Nachrichten aus dem CAN-Netz priorisiert zuzustellen. Um Ereignisnachrichten in der TTP/C-Anwendung zu unterstützen, verwenden die Ereigniskanäle das ETB. Damit filtert der Ereigniskanal die Wiederholungen von Ereignisnachrichten heraus und liefert der Anwendung nur neue Ereignisnachrichten. Es wird so das einmalige Lesen einer Ereignisnachricht garantiert. Aufgrund der Kapselung des ETBs im Ereigniskanal sind Ereignisnachrichten in der TTP/C-Anwendung ohne Mehraufwand nutzbar.

[3] Ein Zeitschlitz kann im TTP/C-Protokoll nur von einem Task beschrieben werden. Daher kann nur der gleiche Task einen neuen Ereigniskanal in dem Zeitschlitz anbieten.

Kapitel 5

Evaluation des Gateways

Die Evaluation des Gateways wird auf zeitlicher und funktionaler Ebene vorgenommen. Das zeitliche Verhalten wird in durch Messungen der Round-Trip Zeiten von Nachrichten überprüft. Für die funktionale Evaluation wird ein Anwendungsszenario beschrieben, in dem innerhalb eines mobilen Roboters ein CAN- und ein TTP/C-Netz durch das Gateway verbunden werden.

5.1 Messung der Nachrichtenverzögerung

In diesem Abschnitt werden die in Unterabschnitt 3.4.4 berechneten Verzögerungszeiten Messungen der Gateway-Implementierung gegenübergestellt. Der Vergleich beider Werte erlaubt eine Beurteilung des Prototyps. Die direkte Messung der entstehenden Latenzzeiten von Nachrichten vom CAN- ins TTP/C-Netz sowie vom TTP/C- ins CAN-Netz ist aufgrund fehlender Uhrensynchronisation zwischen den Netzen nicht möglich. Aus diesem Grund wird die *Round-Trip* Zeit einer Nachricht gemessen. Dabei übermittelt ein Knoten über das Gateway zu einem anderen Knoten eine Nachricht und der Empfänger leitet bei Empfang der Nachricht eine Antwort über das Gateway wieder an den Sender zurück. Der sendende Knoten misst dabei die Zeit zwischen dem Senden der Nachricht und dem Empfang der Antwort. Die Messung der Round-Trip Zeit erlaubt Rückschlüsse auf die Kommunikationsverzögerung und den Jitter, welcher durch das Gateway, die TTP/C- oder die CAN-Anwendung hervorgerufen wird. Die Round-Trip Zeit kann entweder von einem TTP/C- oder von einem CAN-Knoten gemessen werden. In diesem Messaufbau werden die Round-Trip Zeiten von einem CAN-Knoten aufgenommen. Den Messaufbau bilden vier TTTech PowerNode PN312 Knoten für das TTP/C-Netz und ein Atmel AT90CAN128 als CAN-Knoten. Die Entwicklungsumgebung für die TTP/C-Knoten ist bereits in Abschnitt 4.1 beschrieben. Für die Erzeugung der Messanwendung auf dem CAN-Knoten wurde der `avr-gcc 4.1.1` Compiler verwendet.

Die Nachrichten für die Messung haben eine höhere Priorität als alle anderen Nachrichten und es treten nie zwei Nachrichten der Messanwendung zur selben Zeit im System auf. Sie konkurrieren daher nicht um das Kommunikationsmedium. Für das Messszenario werden zwei NRT-Ereigniskanäle im Gateway eingerichtet. Einer der beiden Ereigniskanäle wird in das TTP/C-Netz eingeleitet und der andere spannt sich vom TTP/C- ins CAN-Netz auf. Ein CAN-Knoten sendet Nachrichten in den einen Ereigniskanal und empfängt die zugehörigen Antwortnachrichten des anderen Ereigniskanals. Die Round-Trip Zeit wird dabei vom CAN-Knoten

bestimmt, indem er die Zeit zwischen der Übergabe der Nachricht an den CAN-Controller und dem Erhalt der Antwort in der CAN-Anwendung misst. Dabei müssen verschiedene Verzögerungszeiten ($\Delta t_{cAnwendung}$) innerhalb des CAN-Knotens berücksichtigt werden:

- Verzögerungen durch Nachrichtenübergabe und -bearbeitung des CAN-Controllers
- Behandlung des Empfangsinterrupts zur Bereitstellung der Nachricht in der CAN-Anwendung
- Prozesswechsel bei Multi-Prozessanwendung (Verzögerungen durch Prozesswechsel treten in der Messanwendung nicht auf.)

Die Verzögerungszeit im CAN-Knoten (vgl. Abbildung 5.1) teilt sich in den Teil für das Senden ($\Delta t_{cAnwendung1}$) und den Teil für den Empfang einer CAN-Nachricht ein ($\Delta t_{cAnwendung2}$). Nicht beeinflussbare dynamische Verzögerungszeiten durch eine fehlgeschlagene Arbitrierung oder durch einen belegten CAN-Bus zur Sendezeit ($\Delta t_{ci} = \Delta t_{cBlockierungi} + \Delta t_{cArbitrierungi}$) werden je nach Richtung durch die Verzögerungszeiten Δt_{c1} und Δt_{c2} dargestellt. Die vom CAN-Knoten zu messende Round-Trip Zeit lässt sich durch die Gleichung 5.1 beschreiben.

$$\Delta T_{RoundTrip} = \Delta t_{CAN2TTP} + \Delta t_{tAnwendung} + \Delta t_{TTP2CAN} + \Delta t_{cAnwendung} \tag{5.1}$$

Das Gateway verwendet für die CAN- nach TTP/C-Konvertierung der zu messenden Nachrichten einen eigenen Konvertierungstask und einen exklusiven Zeitschlitz der Breite eins. Die maximale Latenz in der Nachrichtenübermittlung von CAN nach TTP/C für NRT-Nachrichten ($\Delta t_{CAN2TTP}$) wird damit bereits durch die Gleichung 3.14 (Fall A3) auf Seite 93 dargestellt. In dieser Gleichung wird angenommen, dass die CAN-Nachrichten gerade den Task im Gateway verpassen, welcher für die Konvertierung zuständig ist. Diese Wartezeit (φ_1) ändert sich aufgrund der fehlenden Synchronisation zwischen dem Senden der CAN-Nachricht und dem Konvertierungstask im Gateway. In der Gleichung 5.2 wird daher der statistische Mittelwert für die Latenz der Übermittlung von CAN-NRT-Nachrichten ins TTP/C-Netz gegeben. Es ist dazu bemerkt, dass die Zeit für die Übermittlung der CAN-Nachricht ($\Delta t_{CAN} = \Delta t_{c1} + \Delta t_{cNachricht}$) zum Gateway nur bestimmbar ist, wenn die betrachtete Nachricht die Nachricht mit der höchsten Priorität auf dem CAN-Bus ist. Daher haben die gemessenen Nachrichten die höchste Priorität im Messszenario.

$$\Delta T_{avg} = \Delta t_{CAN} + \frac{\Delta t_{Periode}}{2} + \Delta t_{tKonvertierung1} + \varphi_2 + \Delta t'_{tZeitschlitz} \tag{5.2}$$

Die Konvertierung der Nachrichten von TTP/C nach CAN wird durch einen weiteren Konvertierungstask realisiert. Da der Ereigniskanal für die Antwortnachrichten die höchste Priorität hat, verarbeitet die Warteschlange die zugehörigen Nachrichten bevorzugt. Die maximale Latenz durch die Platzierung in der Ausgangswarteschlange und die Arbitrierung des CAN-Busses ist dadurch für diesen Ereigniskanal bestimmbar. Damit ist in Gleichung 3.24 (Fall B3) auf Seite 102 die Verzögerungszeit der NRT-Nachrichtenübermittlung von TTP/C zurück zum CAN-Knoten ($\Delta t_{TTP2CAN}$) bereits beschrieben. Die Ausführungszeit für den TTP/C-zu-CAN-Konvertierungstask ist mit $\Delta t_{tKonvertierung2}$ angegeben. Damit ist die Änderung der Nachrichtenlatenzzeit nur davon abhängig, ob der CAN-Bus zum Sendezeitpunkt blockiert ist.

In der TTP/C-Anwendung werden die Nachrichten des hereinkommenden Kanals vom RoundTrip-Task empfangen, transformiert und in den anderen Ereigniskanal gelegt. Die Verzögerung der Nachricht, welche durch diese Verarbeitung in der TTP/C-Anwendung entsteht, ist in Gleichung 5.3 dargestellt.

$$\Delta t_{tAnwendung} = \Delta t_{RoundTripTask} + \varphi_2' \tag{5.3}$$

Die Anwendungsverzögerung entsteht aus der eingeplanten Ausführungszeit des Tasks ($\Delta t_{RoundTripTask}$) und der fixen Phasenverschiebung zum sendenden Zeitschlitz (φ_2'). Damit kann die Round-Trip Zeit mit allen Teilverzögerungen durch die Abbildung 5.1 graphisch dargestellt werden.

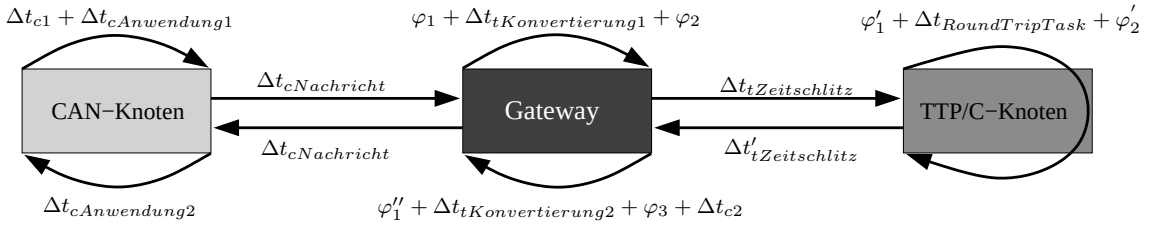


Abbildung 5.1: Messaufbau und Verzögerungszeiten

Die statischen Bestandteile der Round-Trip Zeit im TTP/C-Netz lassen sich durch die Gleichung 5.4 zusammenfassen. Der TTP/C-Round-Trip Zyklus umfasst die Verarbeitungsschritte vom Lesen der CAN-Nachricht bis zum nächsten Sendezeitpunkt der globalen Warteschlange nach Übergabe der Antwort durch das Gateway. Die Variable $\Delta t_{tZyklus}$ gibt die feste Länge des TTP/C-Round-Trip Zyklus an. Im TTP/C-Round-Trip Zyklus kommt es zu keinen Verzögerungen innerhalb der Warteschlange, da die Antwortnachrichten die höchste Priorität in der Warteschlange haben.

$$\begin{aligned} \Delta t_{tZyklus} \quad = \quad & \Delta t_{tKonvertierung1} + \varphi_2 + \Delta t_{tZeitschlitz} + \\ & \varphi_1' + \Delta t_{RoundTripTask} + \varphi_2' + \Delta t'_{tZeitschlitz} + \\ & \varphi_1'' + \Delta t_{tKonvertierung2} + \varphi_3 \end{aligned} \tag{5.4}$$

Die Gesamtverzögerungszeit ($\Delta T_{RoundTrip}$) setzt sich aus bekannten festen Länge des TTP/C-Round-Trip Zyklus ($\Delta t_{tZyklus}$), der CAN-Nachrichtenübertragung ($2 \cdot \Delta t_{cNachricht}$, für hin und zurück) und den veränderbaren Zeiten für die CAN-Übermittlung (Δt_{c1} und Δt_{c2}) sowie der Phase der eingehenden CAN-Nachricht (φ_1) zusammen. Zu dieser beschriebenen Gesamtverzögerungszeit kommen noch die Zeitdifferenzen ($\Delta t_{cAnwendung}$) hinzu, welche sich durch die messende CAN-Anwendung intern ergeben. Da für die Messungen nur die höchstprioren Nachrichten gemessen werden, treten keine weiteren Wartezeiten in Warteschlangen oder durch Nachrichten höherer Priorität im CAN-Netz auf. Die Gleichung 5.5 stellt die Round-Trip Zeit aus Gleichung 5.1 umgeformt dar:

$$\Delta T_{RoundTrip} = \Delta t_{c1} + \varphi_1 + \Delta t_{tZyklus} + \Delta t_{c2} + \Delta t_{cAnwendung} + 2 \cdot \Delta t_{cNachricht} \tag{5.5}$$

Es wird bei der Messung der Round-Trip Zeiten eine Gleichverteilung erwartet, welche maximal $\frac{\Delta t_{Periode}}{2}$ um den Mittelwert schwankt, wenn Verzögerungen durch andere Nachrichten ausgeschlossen sind. Dabei sind der minimale und maximale Wert durch die gegebenen Parameter bestimmbar. Sind Busblockierungen durch andere CAN-Nachrichten möglich, verändert sich die Verteilung der Verzögerungszeiten und die obere Schranke der Verzögerungszeit erhöht sich um die doppelte Sendezeit der längsten CAN-Nachricht. Beide Fälle sind in den folgenden Unterabschnitten erläutert. Zuvor werden jedoch die Messparameter für beide Szenarien dargestellt.

5.1.1 Messparameter

In der Tabelle 5.1 sind die wesentlichen Parameter des Messaufbaus dargestellt. Die beiden Konvertierungstasks sind auf dem Gateway lokalisiert und der RoundTrip-Task befindet sich auf einem anderen TTP/C-Knoten. Alle TTP/C-Tasks besitzen den Clusterzyklus als Periode.

Bezeichnung	**Formelzeichen**	**Wert**
Clusterzyklus	$\Delta t_{Periode}$	$5000\mu s$
TTP/C-Bitrate		$5000kBit/s$
CAN2TTP Taskzeit	$\Delta t_{tKonvertierung1}$	$700\mu s$
TTP2CAN Taskzeit	$\Delta t_{tKonvertierung2}$	$450\mu s$
RoundTrip Taskzeit	$\Delta t_{RoundTripTask}$	$200\mu s$
CAN-Bitrate		$250kBit/s$
CAN-Nachrichtenlänge		$147Bit$
CAN-Nachrichtenlatenz	$\Delta t_{cNachricht}$	$588\mu s$

Tabelle 5.1: Parameter des Messaufbaus

Das Messszenario verwendet erweiterte CAN-Nachrichten mit 8 Byte Payload. Die Payload-Bytes sind dabei mit 0 belegt. Die CAN-Nachrichten haben inklusive der aller Felder damit eine Länge von 131 Bit. Es werden bei der Übertragung aller verwendeten CAN-Nachrichten[1] 12 Stopfbits für das Payload und 4 Stopfbits im Identifier benötigt. Damit ergibt sich eine Gesamtlänge von 147 Bit, welche die Berechnungsgrundlage für die CAN-Nachrichtenlatenz ($\Delta t_{cNachricht}$) bildet.

Für die Zeitmessung wird der Atmel AT90CAN128 Mikrocontroller mit einer Taktfrequenz von 16 MHz verwendet. Der CAN-Knoten nutzt zur Zeitmessung einen Timer, welcher an die Taktfrequenz des Mikrocontrollers gekoppelt ist. Der Timer wird für die Messwertaufnahme vor der Übergabe der Nachricht an den CAN-Controller gestartet und bei Empfang der Antwortnachricht gestoppt. Zur Vereinfachung der Messanwendung im CAN-Knoten werden die CAN-Nachrichten sequentiell mit einer zufälligen Verzögerung gesendet.

Die Messanwendung im TTP/C-Netz besteht neben dem Kontrolltask für jeden Knoten aus den in Tabelle 5.1 dargestellten drei Tasks. Der CAN-zu-TTP/C-Konvertierungstask (CAN2TTP-Task) befindet sich im Gateway. Er nimmt die Nachrichten des eingehenden CAN-Ereigniskanals entgegen[2] und setzt sie in den zugeordneten TTP/C Zeitschlitz (*Slot 1*) um.

[1] Die Nachrichten wurden so ausgewählt, dass sie die gleiche Anzahl von Stopfbits benötigen.

[2] Zur Vereinfachung der Beschreibung des Messaufbaus wird die Methode `fetchNewCANMessages()` in diesem Task aufgerufen und nicht im Kontrolltask, wie in Unterabschnitt 4.3.2 beschrieben.

Diese TTP/C-Nachricht wird von einem anderen Knoten empfangen und dort von einem Task (RoundTrip-Task) über einen anderen Ereigniskanal in einen weiteren Zeitschlitz (*Slot 2*) gelegt. Damit wird die Nachricht wieder an das Gateway zurück gesendet. Die Antwortnachrichten werden vom TTP/C-zu-CAN-Konvertierungstask (TTP2CAN-Task) schließlich ins CAN-Netz überführt. In der Abbildung 5.2 werden Planungszeiten der Tasks[3] und der Zeitschlitze innerhalb des Clusterzyklus der TTP/C-Anwendung graphisch dargestellt.

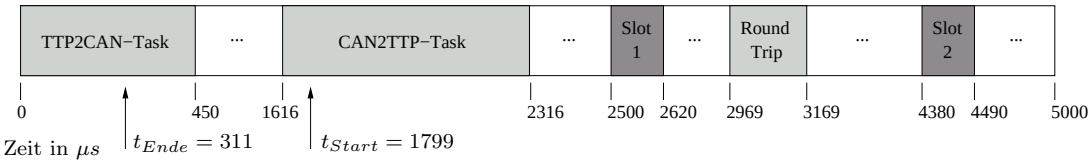


Abbildung 5.2: Task- und Zeitschlitzplanung des Messaufbaus im TTP/C-Netz

Die feste Zeitspanne $\Delta t_{tZyklus}$ ist kürzer als die Zeitspanne, welche sich durch die drei TTP/C-Tasks ergibt ($3834\mu s$), weil die CAN-Nachricht erst nach Beginn des CAN2TTP-Tasks gelesen und die Antwort bereits vor Ende des TTP2CAN-Tasks an den CAN-Controller übergeben wird. Die Startzeit (t_{Start}) des TTP/C-Round-Trip Zyklus ist damit nicht exakt zu Beginn des CAN2TTP-Tasks. Durch Messungen wurde ermittelt, dass die zu messende CAN-Nachricht erst mit einer Verzögerung von $183\mu s$ im Konvertierungstask verfügbar ist. Diese Verzögerung ist dadurch begründet, dass aufgrund des CAN-Puffermanagements vor dem Lesen des entsprechenden CAN-Puffers noch zwei andere CAN-Puffer für *Request* und *Supply Event-Tag* Nachrichten abgefragt werden. Der Zeitpunkt der CAN-Nachrichtenübermittlung (t_{Ende}) im TTP2CAN-Task markiert das Ende des TTP/C-Round-Trip Zyklus. Dieser Zeitpunkt fällt nicht mit der Deadline des Tasks zusammen, sondern tritt bereits zur Zeit $311\mu s$ im Clusterzyklus auf. Ist der CAN-Bus zu diesem Zeitpunkt nicht belegt, wird die CAN-Nachricht sofort gesendet. Damit kann die feste Länge des TTP/C-Round-Trip Zyklus von $\Delta t_{tZyklus} = 3512\mu s$ bestimmt werden. Daraus ergibt sich die minimale Round-Trip Zeit von $4688\mu s$ durch Addition nach Gleichung 5.5 mit den Nachrichtentransportzeiten im CAN-Netz ($2 \cdot \Delta t_{cNachricht} = 2 \cdot 588\mu s$). Die minimale Round-Trip Zeit tritt auf, wenn die Phase der eingehenden CAN-Nachricht minimal ist ($min(\varphi_1) = 0$). Die Verzögerungen in der CAN-Anwendung ($\Delta t_{cAnwendung}$) und bei der Übermittlung (Δt_{c1} und Δt_{c2}) sind dabei nicht mit berücksichtigt. Die Zeitpunkte (t_{Start}, t_{Ende}), welche die Länge es TTP/C-Round-Trip Zyklus ($\Delta t_{tZyklus}$) markieren, sind innerhalb der TTP/C-Anwendung ermittelt worden. Aufgrund der festen Planung der Tasks sind diese Werte hinreichend genau.

5.1.2 Ergebnisse ohne CAN-Buslast

In diesem Messszenario kann es zu keiner dynamischen Verzögerung durch den belegten CAN-Bus kommen, da nur die zu messenden CAN-Nachrichten auf dem CAN-Bus übermittelt werden und der CAN-Knoten die Messungen sequentiell durchführt. Die Tabelle 5.2 stellt die Ergebnisse der Messungen dar.

Es fällt auf, dass die berechnete minimale Verzögerung von $4688\mu s$ in der Messung um $11\mu s$ überschritten wird. Diese Differenz entsteht durch die CAN-Lese- und Schreibfunktionen im

[3] Die Planung der Tasks erfolgt in *TTP Build* über *Task Chains*, welche einen oder mehrere Tasks beinhalten können. Innerhalb dieser Task Chains werden die Tasks dann sequentiell abgearbeitet. Siehe dazu [TTT05a].

Bezeichnung	Wert
Anzahl der Messungen	10.000
Minimale Verzögerungszeit	$4699\mu s$
Maximale Verzögerungszeit	$9702\mu s$
Durchschnittswert	$7179\mu s$
Varianz	$2007\mu s$
Abstand	$5003\mu s$

Tabelle 5.2: Messergebnisse ohne Buslast

Gateway und durch die Verzögerungen innerhalb der CAN-Anwendung ($\Delta t_{cAnwendung}$). Die größte Zeitdifferenz entsteht dabei wahrscheinlich durch die CAN-Lesefunktion im Gateway, welche nach Messungen ca. $35\mu s$ benötigt. Wie zuvor beschrieben ist der Startzeitpunkt des TTP/C-Round-Trip Zyklus (t_{Start}) mit dem Zeitpunkt belegt in dem die CAN-Nachricht in der Gateway-Applikation verfügbar ist. Dabei wird davon ausgegangen das CAN-Nachrichten, welche direkt zu diesem Zeitpunkt angekommen sind, noch in diesem Zyklus empfangen werden. Diese Annahme weicht aufgrund der langen Bearbeitungszeit (ca. $35\mu s$) der CAN-Lesefunktion im Gateway vom realen Zeitpunkt ab. Es ist davon auszugehen, dass zu einem Zeitpunkt innerhalb dieser Lesefunktion eine Nachricht, welche zu diesem Zeitpunkt ankommt, bereits nicht mehr in diesem Zyklus gelesen werden kann. Damit erhöht sich die feste Verzögerung durch den TTP/C-Round-Trip Zyklus ($\Delta t_{tZyklus}$), um einen nicht bestimmbaren Wert mit der maximalen Größe der Bearbeitungszeit der CAN-Lesefunktion.

Der maximale Wert der Verzögerung entsteht, wenn eine CAN-Nachricht gerade den Lesezeitpunkt des Gateways verpasst hat. Damit verzögert sich diese Nachricht zusätzlich um einen Clusterzyklus. Es ergibt sich der theoretische Wert von $9688\mu s$. Der gemessene Maximalwert überschreitet diesen Wert um $14\mu s$. Damit ist diese Differenz zwischen gemessenen und berechneten Wert des Maximalwertes um $3\mu s$ größer als die des Minimalwertes. Daraus folgt, dass der Abstand zwischen dem kleinsten und dem größten Wert nicht wie erwartet $5000\mu s$ beträgt, sondern $3\mu s$ höher ist. Diese Differenz wird wahrscheinlich von Jittern in der Zeitmessung im CAN-Controller oder durch Ungenauigkeiten im CAN-Bittiming verursacht. Die Annahme wurde durch eine Kontrollmessung, bei der das TTP/C-Netz durch einen weiteren CAN-Knoten ersetzt wurde, bestätigt. In der Kontrollmessung traten Jitter von über $4\mu s$ auf.

Die Abbildung 5.3 zeigt die Verteilung der Round-Trip Zeiten als Histogramm. Das Ergebnis der Messungen ist eine Gleichverteilung im Intervall von der minimalen Verzögerung bis zur maximalen Verzögerung. Damit variiert die Round-Trip Zeit mit dem halben Clusterzyklus um die durchschnittliche Verzögerung. Diese Verteilung wird durch die veränderliche Phasenverschiebung (φ_1) vom Zeitpunkt des Eintreffens der Nachricht bis zur Verarbeitung durch das Gateway in Gleichung 5.5 verursacht. Die Schwankungen der Phase φ_1 gehen auf die Asynchronität des Sendezeitpunktes zum zyklischen CAN-zu-TTP/C-Konvertierungstask zurück. Die Messungen bestätigen damit, dass sich die Verzögerungen der Nachrichten unter den beschriebenen Bedingungen aus einem im Voraus bestimmbaren Anteil ($\Delta t_{tZyklus}$ und $\Delta t_{cNachricht}$) und einem Jitter zusammensetzen. Die Schwankung der Messwerte von $5003\mu s$ besteht aus der Phasenverschiebung ($max\ (\varphi_1) = 5000\mu s$) von Empfangszeitpunkt im Gateway und dem Konvertierungstask, den Verzögerungen durch die Messanwendung ($\Delta t_{cAnwendung}$)

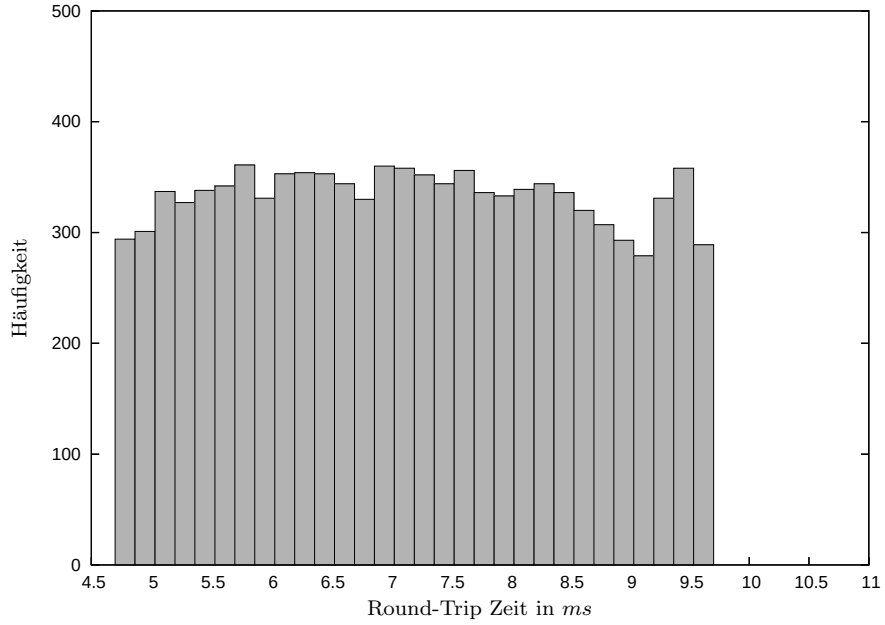


Abbildung 5.3: Histogramm der Round-Trip Zeiten ohne CAN-Busbelastung

und Ungenauigkeiten im Bittiming der CAN-Controller. Der Hauptanteil des Jitters ($5000\mu s$) ist daher mit der Phasenverschiebung (φ_1) zu begründen. Dieser große Jitter entsteht durch das Gateway bei der Einleitung von CAN-Nachrichten ins TTP/C-Netz.

Die Ende-zu-Ende Verzögerungszeit von TTP/C nach CAN hat praktisch keinen Jitter. Dies ist durch die Betrachtungen aus Unterabschnitt 3.4.4 begründet und konnte durch die Messergebnisse bestätigt werden. Die Ende-zu-Ende Verzögerungszeit besteht aus der Länge des TTP/C-Verarbeitungszyklus, der Zeit für die Übermittlung der CAN-Nachricht ($\Delta t_{cNachricht} = 588\mu s$) und den Verzögerungen in der CAN-Anwendung ($\Delta t_{cAnwendung2}$) beim Empfang und der Verarbeitung der Antwort. Der TTP/C-Verarbeitungszyklus beinhaltet alle Verarbeitungsschritte innerhalb des TTP/C-Netzes zur Verarbeitung von Ereigniskanälen, welche ins CAN-Netz gesendet werden. Er beginnt mit dem Zeitpunktes an dem der Round-Trip Task beendet wird und endet bei der CAN-Sendezeit (t_{Ende}) des Gateways. Damit ist die Länge des TTP/C-Verarbeitungszyklus $2142\mu s$. Die gesamte Ende-zu-Ende Verzögerungszeit ist demnach, ohne Berücksichtigung der Verzögerung in der CAN-Anwendung, in Gleichung 5.6 dargestellt.

$$\Delta t_{TTP2CAN} = 2142\mu s + 588\mu s = 2730\mu s \tag{5.6}$$

Die Latenzzeiten von CAN nach TTP/C sind in der Messanwendung durch den großen Jitter geprägt. Der maximale Jitter entspricht dabei der Periode des CAN-Verarbeitungszyklus. Der CAN-Verarbeitungszyklus umfasst die CAN-Lesefunktion, den Konvertierungstask, den sendenden TTP/C-Zeitschlitz und die Phasenverschiebung (φ_1') zum Anwendungstask (Round-Trip Task). Da der CAN-Verarbeitungszyklus einmal im Clusterzyklus aufgerufen wird, nehmen die maximalen Schwankungen in den Kommunikationsverzögerungen die Länge des Clusterzyklus an. Die Verzögerung vom Empfang der Nachricht im Gateway bis zur Zustellung zum

TTP/C-Task ergibt sich aus der Differenz zwischen der Startzeit des Round-Trip Tasks und dem Lesezeitpunkt der CAN-Nachrichten (t_{Start}). Unter der Annahme, dass der Zeitpunkt t_{Start} mit $1799\mu s$ für die Bestimmung hinreichend genau ist, wird in diesem Messszenario eine Verzögerung durch das Gateway von $1170\mu s$ veranschlagt. Die Gleichungen 5.7 und 5.8 stellen die minimalen und maximalen Verzögerungen einer Nachricht von CAN nach TTP/C dar. Damit ist der Jitter der Nachricht durch das Gateway bedeutend größer als die eigentliche Ende-zu-Ende Verzögerung.

$$min\ \Delta t_{CAN2TTP} = 1170\mu s + 588\mu s = 1758\mu s \tag{5.7}$$

$$max\ \Delta t_{CAN2TTP} = 1170\mu s + 588\mu s + 5000\mu s = 6758\mu s \tag{5.8}$$

Eine Verkleinerung des Clusterzyklus oder eine mehrfache Planung des CAN-Verarbeitungszyklus verringert den Jitter. Der Nachteil dabei ist, dass durch eine Verkleinerung des Clusterzyklus die gesamte TTP/C-Anwendung über weniger Ausführungszeit für die lokalen Tasks und weniger Platz für andere Nachrichten verfügt. Auch eine mehrfache Planung des CAN-Verarbeitungszyklus schränkt die Ausführungszeit für andere Tasks auf dem Gateway ein. Zusätzlich verringert sich die Kommunikationsbandbreite und die verfügbare Ausführungszeit der Ereignisnachrichten verarbeitenden Knoten. Eine Verringerung des Jitters wird also in beiden Fällen mit der Verringerung der verfügbaren Ausführungszeiten auf den TTP/C-Knoten und der geringeren freien Kommunikationsbandbreite bezahlt. Der Jitter kann nur soweit minimiert werden, bis er die Zeitspanne, welche für den CAN-Verarbeitungszyklus notwendig ist, erreicht. Durch Planung des CAN-Verarbeitungszyklus oder Änderungen am Clusterzyklus kann der Jitter also nicht beliebig verringert werden.

Damit müssen für die NRT-Ereigniskanäle, welche nach Best-Effort versendet werden, hohe Jitter akzeptiert werden. Der Prototyp des Gateways (siehe Kapitel 4) unterstützt nur NRT-Ereigniskanäle. Es handelt sich daher im Folgenden um eine theoretische Betrachtung. Die Nachrichten der SRT-Ereigniskanäle müssen genau wie die NRT-Kanäle asynchron versendet werden, daher geht in die Nachrichtenlatenzen dieser Kanäle auch der gleiche Jitter mit ein. Aufgrund der Dimension des Jitters folgt, dass durch ein TTP/C-CAN-Gateway keine SRT-Ereigniskanäle mit engen Fristen gesendet werden können. Für solche Nachrichten ist nicht sichergestellt, dass sie rechtzeitig ankommen. Um in diesem Messszenario ohne CAN-Busbelastung eine SRT-Ereignisnachricht ins TTP/C-Netz zu senden, benötigt die Nachricht theoretisch eine relative Deadline von mindestens $6758\mu s$ nach Gleichung 5.8. Eine Synchronisation des CAN- mit dem TTP/C-Netz ermöglicht für die variable Phasenverschiebung (φ_1) einen Wert, der nur durch Ungenauigkeiten in der Synchronisation (ε) schwankt. Dies führt zu einer deutlichen Verringerung des Jitters und erlaubt eine Echtzeitkommunikation mit kurzen Fristen zwischen dem CAN- und TTP/C-Netz.

5.1.3 Ergebnisse mit CAN-Buslast

Dieser Unterabschnitt beschreibt die Veränderungen der minimalen und maximalen Verzögerungen und der Verteilung der Round-Trip Zeiten, wenn andere Nachrichten auf dem CAN-Bus zugelassen werden. Dazu sendet ein weiterer CAN-Knoten (mit gleichem Mikrocontroller und gleicher Entwicklungsumgebung) als Lastgenerator ununterbrochen Nachrichten ins CAN-Netz. Diese Lastnachrichten haben eine geringere Priorität als die beiden CAN-Nachrichten, welche

zwischen dem Gateway und dem messenden Knoten ausgetauscht werden. Die Länge einer Lastnachricht beträgt $147Bit$, damit ist die Übertragungszeit der Nachricht auf dem CAN-Bus ebenfalls $588\mu s$. Die Tabelle 5.3 stellt die gemessenen Werte dieses Messszenarios dar.

Bezeichnung	**Wert**
Anzahl der Messungen	10.000
Minimale Verzögerungszeit	$4832\mu s$
Maximale Verzögerungszeit	$10861\mu s$
Durchschnittswert	$7763\mu s$
Varianz	$2123\mu s$
Abstand	$6029\mu s$

Tabelle 5.3: Messergebnisse mit Buslast

Die erwartete minimale Round-Trip Zeit beträgt genau die gemessene Verzögerung des vorherigen Messaufbaus. Dabei muss der folgende Fall im Messszenario eintreten:

- Der messende Knoten und der Lastgenerator beginnen gleichzeitig die Busarbitrierung mit dem Aufschalten des SOF-Bits, wobei der Lastgenerator die Arbitrierung verliert und der messende Knoten die Nachricht verzögerungsfrei senden kann ($\Delta t_{c1} = 0$).
- Die CAN-Nachricht erreicht das Gateway zum spätesten Zeitpunkt, um im beginnenden TTP/C-Round-Trip Zyklus gelesen zu werden ($\varphi_1 = 0$).
- Das Gateway beginnt gleichzeitig mit dem Lastgenerator die Übermittlung seiner Nachricht mit dem Aufschalten des SOF-Bits. Das Gateway gewinnt und sendet die Nachricht verzögerungsfrei ($\Delta t_{c2} = 0$).

Der in der Messung ermittelte Wert ist mit $133\mu s$ größer als der erwartete Wert für die minimale Round-Trip Zeit. Damit konnte der Fall der minimalen Verzögerung nicht gemessen werden. Es ist möglich, dass diese Zeit aufgrund der geringen Wahrscheinlichkeit des gleichzeitigen Eintretens aller drei Teilfälle nicht gemessen wurde. Die Wahrscheinlichkeit[4], dass eine zum beliebigen Zeitpunkt gesendete Nachricht genau mit dem SOF-Bit einer Lastnachricht zusammenfällt, ist $p_c = \frac{1}{147}$. Für den Fall, dass eine CAN-Nachricht genau den letztmöglichen Zeitpunkt der Lesefunktion trifft, kann bei der Länge des Clusterzyklus von $5000\mu s$ und einer CAN-Bitzeit von $4\mu s$[5] die Wahrscheinlichkeit $p_{tPeriode} = \frac{1}{1250}$ angenommen werden. Damit ergibt sich die Wahrscheinlichkeit für das Eintreten dieses Falles nach Gleichung 5.9. Nach dieser Berechnung gibt es 27 Millionen verschiedene Möglichkeiten für die Zusammensetzung der Round-Trip Zeit. Die Wahrscheinlichkeit ist daher sehr gering, dass der Fall der minimalen Round-Trip Zeit bei 10.000 gemessenen Nachrichten eintrifft.

$$\begin{aligned} P &= p_{c1} \cdot p_{tPeriode} \cdot p_{c2} \\ P &= \frac{1}{147} \cdot \frac{1}{1250} \cdot \frac{1}{147} = 3,7 \cdot 10^{-8} \end{aligned} \tag{5.9}$$

[4]Es wird von den hier verwendeten $147Bit$ langen CAN Nachrichten ausgegangen.
[5]Dies entspricht der CAN-Bitrate von $250kBit/s$.

Für die maximale Round-Trip Zeit wird ein Wert angenommen der im Vergleich zur Messung ohne Buslast um die Übermittlungszeit für zwei Lastnachrichten ($2 \cdot \Delta t_{cNachricht} = 1176\mu s$) größer ist. Für diese maximale Latenz muss genau der folgende Fall auftreten:

- Die Nachricht im messenden Knoten wird direkt nach dem Senden des SOF-Bits einer Lastnachricht bereit. Die zu messende Nachricht muss die Länge der Lastnachricht abwarten, bis die Busarbitrierung beginnen kann ($\Delta t_{c1} = \Delta t_{cNachricht} = 588\mu s$).
- Die CAN-Nachricht erreicht genau nach dem Lesezeitpunkt das Gateway. Das Gateway bearbeitet diese Nachricht erst im nächsten TTP/C-Round-Trip Zyklus ($\varphi_1 = \Delta t_{Periode} = 5000\mu s$).
- Der Lastgenerator beginnt das Senden einer Nachricht. Das Gateway ist ab dem SOF-Bit der Lastnachricht bereit zum Senden der eigenen Nachricht. Die Nachricht verzögert sich daher um die Länge der Lastnachricht ($\Delta t_{c2} = \Delta t_{cNachricht} = 588\mu s$).

Der gemessene Wert ist jedoch statt $1176\mu s$ nur um $1159\mu s$ größer. Es wird daher auch hier angenommen, dass der Maximalwert aufgrund seiner geringen Wahrscheinlichkeit nach Gleichung 5.9 nicht gemessen wurde.

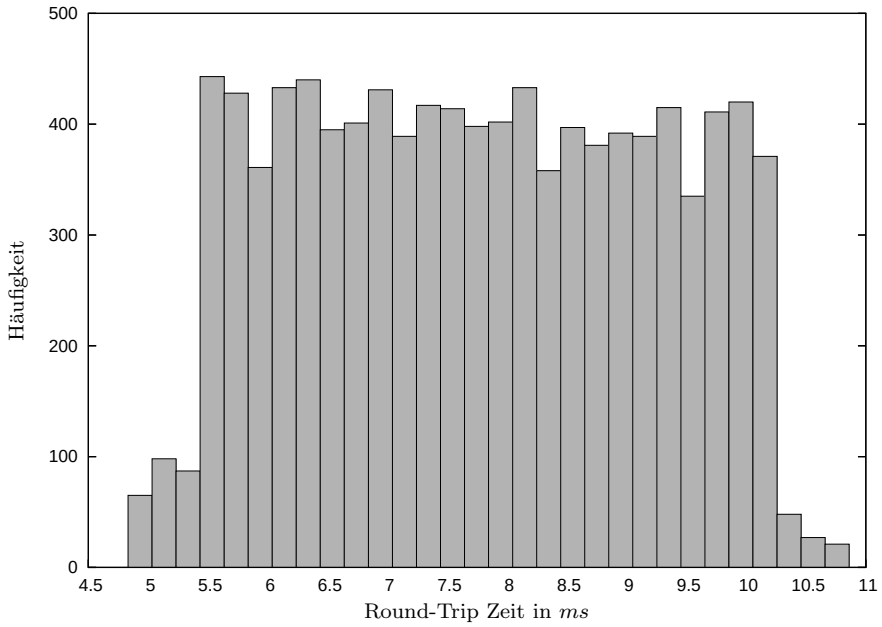


Abbildung 5.4: Histogramm der Round-Trip Zeiten mit CAN-Busbelastung

Die Abbildung 5.4 zeigt die Verteilung der Round-Trip Zeiten für die Messung mit CAN-Buslast. Wie im Vergleich zum Histogramm in Abbildung 5.3 zu erkennen ist, dehnt sich die Verteilung zu größeren Verzögerungen hin aus. Die geringen Werte an den Rändern der Verteilung sind durch die geringe Wahrscheinlichkeit dieser Fälle zu erklären. Die hohen Häufigkeiten in der Mitte der Verteilung sind auf die höhere Wahrscheinlichkeit dieser Round-Trip Zeiten durch die Überlagerung verschiedener Einzelfälle mit der Wahrscheinlichkeit nach Gleichung 5.9 zurückzuführen.

Aus den Messungen wird abgeleitet, dass die Ende-zu-Ende Verzögerung vom CAN-Sender zum TTP/C-Knoten bei der Belastung des Busses einem größeren Jitter unterliegt. Die maximale Schwankung der Kommunikationsverzögerung ist mit der Periode des CAN-Verarbeitungszyklus und der Länge der längsten Nachricht, die den CAN-Knoten beim Senden blockieren kann, gegeben. Die Gesamtverzögerung einer Nachricht vom TTP/C-Sender zum CAN-Empfänger erhält durch die Blockierung des CAN-Busses ebenfalls eine zusätzliche variable Verzögerung von maximal der Übermittlungszeit der längsten CAN-Nachricht. Damit unterliegt auch diese Kommunikationsrichtung einem Kommunikationsjitter, welcher aber im Gegensatz zur Kommunikation von CAN nach TTP/C bedeutend geringer ist. Beide Ergebnisse gehen von der getroffenen Annahme aus, dass die gemessene Nachricht die höchste Priorität auf dem Bus hat.

Die ermittelten Ende-zu-Ende Verzögerungszeiten bei der Konvertierung decken sich mit den erwarteten Verzögerungen aus Unterabschnitt 3.4.4 und entsprechen auch den Betrachtungen aus [LA99] über den Kommunikationsjitter bei zeitgesteuerter Kommunikation mit ereignisgesteuerten Knoten. Ein minimaler Kommunikationsjitter kann nach Unterabschnitt 3.4.4 nur bei einer globalen Zeit und der Verwendung von HRT-Ereigniskanälen erreicht werden.

5.2 Anwendungsszenario

Eine Zielssetzung der Arbeit ist es, den Gateway-Prototypen anhand eines Anwendungsszenarios zu evaluieren. Als Szenario für eine modellübergreifende Anwendung wurde ein mobiler Roboter, bestehend aus einem ereignis- und einem zeitgesteuerten Teilsystem, ausgewählt. Die verteilte Roboteranwendung wird in eine Ebene für die Navigation und eine weitere für die Fahrkontrolle aufgeteilt. Dabei wird jede Schicht durch ein Teilsystem repräsentiert. Die Abbildung 5.5 stellt die funktionale Aufteilung dar.

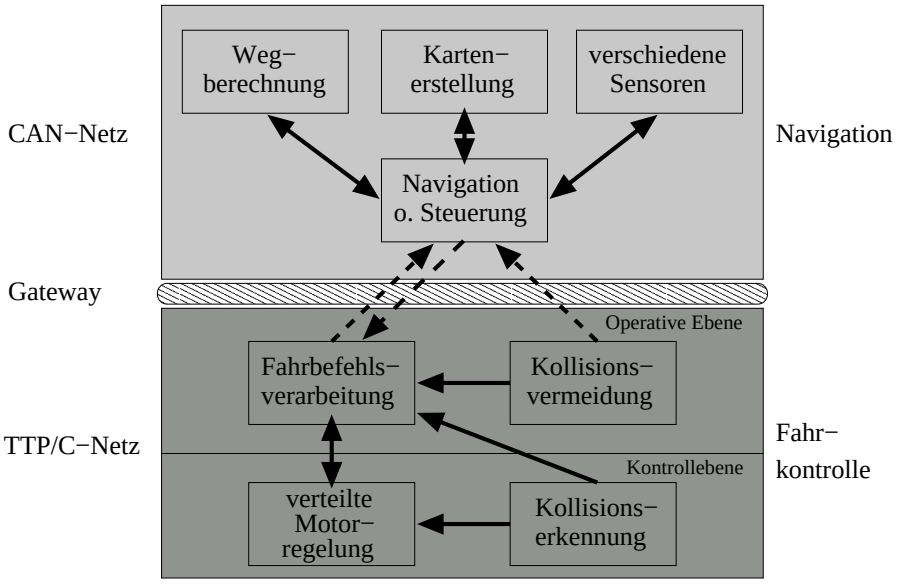


Abbildung 5.5: Roboterstruktur

Die Navigationsebene ist für die eigenständige Bewegung des Roboters zuständig. Dabei ist die Anwendung in dieser Ebene flexibel gestaltbar. Es können unterschiedliche Methoden zur Navigation, wie z.B. Kartenerstellung, Hinderniserkennung oder Wegberechnung mit verschiedenen austauschbaren Sensoren auf dieser Ebene kombiniert werden. Es ist auch möglich, die Steuerung des Roboters durch einen Benutzer oder mit Hilfe einer Kooperation mit anderen Robotern zu realisieren. Es wird in der Navigationsebene nicht vorgegeben, welche Komponenten wie zusammenarbeiten. Dies ermöglicht eine flexible erweiterbare Navigation. Allein die Kommunikationsschnittstelle zur Fahrkontrolle ist in der Navigationsebene fest definiert. Daher eignet sich für die Anforderungen an die Kommunikation der verteilten Anwendung dieser Schicht das erweiterbare ereignisgesteuerte Kommunikationsprotokoll CAN.

Die Fahrkontrolle teilt sich in zwei Bereiche auf: die operative Ebene und die Kontrollebene. In der operativen Ebene werden die Fahrbefehle von der Navigationsebene verarbeitet und die Position des Roboters überwacht. Es wird dazu der Navigation eine Schnittstelle angeboten, welche abstrakte Fahrbefehle wie *<Fahre vorwärts, Geschwindigkeit, Entfernung>* oder *<Drehe nach links, Geschwindigkeit, Winkel>* definiert. Neben der Umsetzung der Fahrbefehle, wird in der operativen Ebene auch die Verhinderung von Kollisionen im Nahbereich realisiert. Die Kollisionsvermeidung ist aufgrund der strengen zeitlichen Anforderungen und der Notwendigkeit der sofortigen Reaktion in der operativen Ebene zu finden, denn die Navigation kann diese zeitliche Anforderung für die Kollisionsvermeidung nicht garantieren. Die operative Ebene stellt der Navigation Informationen über Geschwindigkeit, gefahrene Strecke und die Daten der Kollisionsvermeidung bereit.

Die Kontrollebene realisiert die verteilte Geschwindigkeitsregelung der Motoren, welche von der operativen Ebene kontrolliert wird. Neben der Motorregelung kann eine Kollisionserkennung in dieser Ebene implementiert werden. Um minimale Reaktionszeiten zu erreichen, werden die Sensoren direkt mit den Motorcontrollern verbunden.

Die Kommunikation in der Fahrkontrollebene ist zeit- und sicherheitskritisch. Denn es kann zu einer Schädigung der Umwelt oder des Roboters führen, wenn eine Verzögerung der Nachrichten für die Kollisionserkennung bzw. -vermeidung auftritt. Eine Unterbrechung oder Verzögerung der Kommunikation zur verteilten Regelung der Motoren, kann Ungenauigkeiten oder gröbere Fehler in der Lageregelung des Roboters hervorrufen und somit wird eine falsche Position für die Navigation ermittelt. Aus diesen Gründen verwendet die Anwendung in der Fahrkontrolle das zeitgesteuerte deterministische TTP/C-Protokoll. Damit werden Kommunikation und Anwendung in der Fahrkontrolle statisch im Voraus geplant. Dies ist jedoch unproblematisch, da die Anwendung in diesem Bereich nicht erweiterbar sein muss. Die Flexibilität der Gesamtanwendung wird durch die dynamisch erweiterbare Navigationsebene des Roboters garantiert.

Zwischen der Fahrkontrolle und der Navigationsebene wird die Kommunikation durch das in dieser Arbeit vorgestellte Gateway (siehe Abbildung 5.5) realisiert. Es leitet dazu die Fahrbefehle von der Navigation in die Fahrkontrolle und Informationen zur Geschwindigkeit, der zurückgelegten Entfernung und die Daten der Kollisionsvermeidung von der Fahrkontrolle an die Navigationsschicht weiter. Die Nutzung der Fahrbefehle in der operativen Ebene erlaubt die Steuerung des mobilen Roboters unabhängig von der Erzeugung der Fahrbefehle in der Navigation.

Der im Rahmen dieser Arbeit umgesetzte Prototyp des mobilen Roboters (siehe Abbildung 5.6) ist in der Navigationsebene nur durch die direkte Kontrolle des Benutzers steuerbar. Es

werden dazu CAN-Nachrichten mit den bereits beschriebenen Fahrbefehlen verwendet. In der Fahrkontrolle wurden im Prototyp neben der Fahrbefehlsverarbeitung, der Regelung und der Motorsteuerung auch Sensoren zur Kollisionsvermeidung integriert.

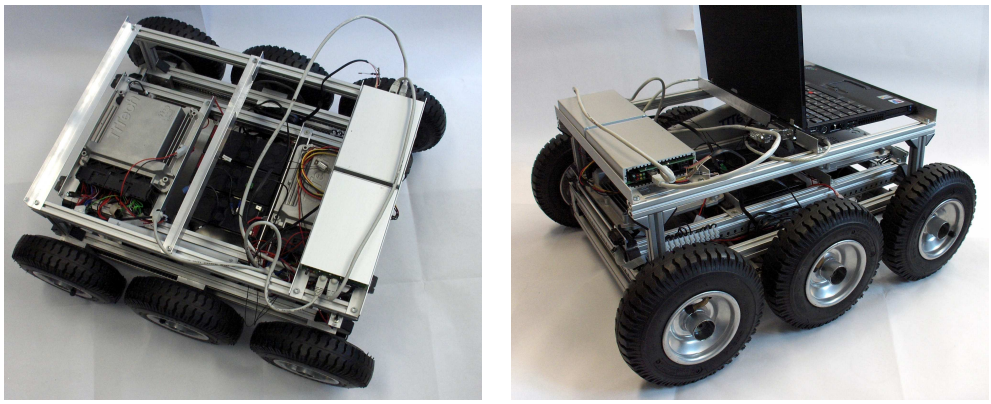


Abbildung 5.6: Prototyp des mobilen Roboters mit TTP/C-CAN-Gateway

Die Verwendung der Fahrbefehle im TTP/C-Netz verdeutlicht die Unterstützung von sporadischen Ereignisnachrichten durch Gateway und TTP/C-Anwendung. Das Gateway bietet der CAN-Navigationsanwendung verschiedene Ereigniskanäle an. Zur Begrenzung der genutzten Bandbreite im CAN-Netz werden nur abonnierte Kanäle weitergeleitet. Weiterhin dürfen nicht alle Ereigniskanäle aus der Fahrkontrolle ins CAN-Netz geleitet werden. So können keine Ereigniskanäle zur Motorsteuerung das Gateway passieren, da diese Informationen im CAN-Netz nicht von Bedeutung sind. Die CAN-Anwendung kann entscheiden wie die Fahrbefehle erzeugt und wie die bereitgestellten Ereigniskanäle aus der Fahrkontrolle genutzt werden. Eine Erweiterung der CAN-Anwendung zur Realisierung einer autonomen Navigation auf mehreren Knoten führt zu einer Aufwertung des Anwendungsszenarios. Die Erweiterung der Navigation beeinflusst aber nicht die Funktionsweise des Gateways oder der TTP/C-Anwendung und behält die Struktur der Beispielanwendung bei. Durch dieses Anwendungsszenario wird die funktionale Aufteilung einer Gesamtanwendung in ein ereignis- und zeitgesteuertes Teilsystem und deren Kommunikation durch ein Gateway dargestellt. Das beschriebene Anwendungsszenario zeigt die Nutzbarkeit des entwickelten Konzeptes und liefert eine flexibel erweiterbare Plattform für einen mobilen Roboter.

Kapitel 6

Zusammenfassung

Eine Kopplung zwischen dem zeit- und ereignisgesteuerten Kommunikationsmodell ermöglicht eine verteilte Anwendung, welche eine Kommunikation zwischen Knoten aus beiden Modellen erlaubt. Die Kombination von Teilanwendungen in lokalen heterogenen Systemen bildet eine föderierte Architektur in der alle Systeme zu einer Gesamtanwendung gehören. Die einzelnen Teilanwendungen können auf Systemen mit unterschiedlicher Hard- und Softwarekonfiguration und verschiedenen Kommunikationsprotokollen realisiert werden. Sie nutzen daher die Vor- und Nachteile der in dem System verwendeten Paradigmen. Eine gemeinsame Middleware verbindet die Teilanwendungen und ermöglicht die transparente Kommunikation aller Knoten in der Anwendung. An den Übergängen zwischen den heterogenen Teilsystemen sind Gateways positioniert, welche die Kommunikation zwischen den Netzen ermöglichen.

Mit dieser Arbeit wurde ein solches Gateway zwischen dem zeit- und dem ereignisgesteuerten Kommunikationsmodell entwickelt. Das Gateway ermöglicht eine Interoperabilität von CAN- und TTP/C-Knoten auf Basis der Middleware COSMIC. Bestehende oder neu entwickelte COSMIC-Anwendungen erhalten dadurch die Möglichkeit TTP/C-Teilsysteme zu integrieren. Dabei werden die Anforderungen an das Gateway nach einem globalen Adressierungsschema, der Kontrolle des Informations- und Kontrollflusses zwischen den Netzen und einer Unterstützung von Echtzeitkommunikation im entwickelten Konzept erfüllt. Die konsistente Adressierung und die Echtzeitfähigkeit der Kommunikation werden mit Hilfe der Middleware COSMIC erreicht. Sie basiert auf dem Publish/Subscribe-Konzept und bietet eine inhaltsbasierte Kommunikation auf Basis von Ereigniskanälen. COSMIC ist u.a. für das CAN-Protokoll verfügbar. Im Rahmen der Arbeit wurden die Mechanismen dieser Middleware für das zeitgesteuerte TTP/C-Protokoll eingeführt. Die Verwendung der COSMIC-Ereigniskanäle für TTP/C bietet der zeitgesteuerten Anwendung die Möglichkeit dynamische Kanäle und sporadische Ereignisnachrichten effektiv zu nutzen und so die Kommunikation flexibler zu gestalten. Die Adaption der Middleware für das TTP/C-Protokoll und die Entwicklung eines Gateways zur Kopplung mit dem CAN-Protokoll gestatten die Integration von TTP/C-Netzen in eine heterogene COSMIC-Gesamtanwendung. Es ist den kommunizierenden Anwendungsteilen dabei nicht bekannt, wo die abonnierten Ereigniskanäle generiert oder wohin die von ihnen publizierten Nachrichten gesendet werden. Diese transparente Kommunikation wird durch die globale inhaltsbasierte Adressierung von COSMIC gewährleistet.

Das Gateway kontrolliert den Informationsfluss zwischen den Netzen und setzt die Minimierung des weitergeleiteten Nachrichtenverkehrs durch. Dazu ermöglicht es die Filterung von

Ereigniskanälen und erlaubt so die Definition der Informationsmenge, welche die Netzgrenzen überschreiten darf. Eine Abonnementverwaltung sichert zu, dass nur abonnierte Ereigniskanäle vom Gateway in das jeweilige Netz geleitet werden. Damit stellt das Gateway alle Ereigniskanäle aus dem TTP/C-Netz bereit, die für eine CAN-Anwendung interessant sind. Es können dann verschiedene CAN-Anwendungen mit der TTP/C-Anwendung gekoppelt werden. Der jeweiligen CAN-Anwendung werden dabei nur die benötigten Kanäle zugestellt. Dies ermöglicht eine dynamische Erweiterung, Veränderung oder den Austausch der CAN-Anwendung ohne Änderung des TTP/C-Teilsystems. Die Unterstützung von Echtzeitklassen durch COSMIC ermöglicht dem Gateway eine priorisierte Konvertierung und Übermittlung der Ereigniskanäle. Im Konzept des Gateways sind für die Echtzeitklassen passende Konvertierungsmechanismen vorgesehen, die die Latenz der Nachrichtenübermittlung und den Kommunikationsjitter minimieren. Das entwickelte Konzept für das Gateway beschreibt die zeitlichen Verzögerungen durch die netzübergreifende Kommunikation. Dabei verursacht die Abbildung von asynchroner ereignisgesteuerter Kommunikation in ein zeitgesteuertes periodisches Kommunikationschema große Schwankungen der Ende-zu-Ende Verzögerung. Eine zeitsynchrone Kommunikation erlaubt hingegen eine bidirektionale Echtzeitkommunikation zwischen den Netzen mit minimalem Jitter.

Eine prototypische Implementierung des vorgestellten Konzeptes umfasst Ereigniskanäle ohne Echtzeiteigenschaften. Die Implementierung des Gateways konnte für die verfügbaren Nicht-Echtzeitkanäle erfolgreich durch die Messung der Nachrichtenverzögerungen evaluiert werden. Durch die umfangreichen Möglichkeiten der Konfiguration kann das Gateway vielfältig eingesetzt werden. Die Programmierschnittstelle kommt mit wenigen Objekten und Methoden aus und die Implementierung des Kontrolltasks sowie der Konvertierungstasks sind daher mit wenigen Befehlen schnell realisiert. Zur Nutzung der Ereigniskanäle in der TTP/C-Anwendung sind entsprechende Klassen und Schnittstellen entwickelt und umgesetzt. Diese Schnittstellen ermöglichen durch minimalen Mehraufwand in der TTP/C-Anwendung die komfortable Nutzung der Ereigniskanäle. Dynamisch befüllte Zeitschlitze und Ereignisnachrichten werden in der TTP/C-Anwendung durch diese Programmierschnittstelle einfach unterstützt.

Die Anwendbarkeit des Gateways wurde anhand eines mobilen Roboters gezeigt. Das Roboterszenario setzt sich aus der Navigation und der Fahrkontrolle zusammen. Dabei wird das navigierende Teilsystem durch ein CAN-Netzwerk und die Fahrkontrolle durch eine TTP/C-Cluster repräsentiert. Es wurde gezeigt, wie eine Anwendung sinnvoll in ein zeit- und ereignisgesteuertes Teilsystem zerlegt wird und wie die Teilsysteme mit Hilfe des Gateways interoperieren.

Der im Rahmen der Arbeit implementierte Prototyp erlaubt keine Echtzeitkommunikation. Die Echtzeitkommunikation erfordert eine Überwachung der Fristen und der weiteren zeitlichen Eigenschaften der weiterzuleitenden Ereigniskanäle im Gateway. Dabei muss darauf geachtet werden, dass alle Knoten im CAN- und im TTP/C-Netz den Ereigniskanälen die gleichen Eigenschaften zuordnen. Das Gateway kann aber nicht zusichern, dass die zeitlichen Eigenschaften der Ereigniskanäle systemweit konsistent vergeben sind. Für die Entwicklung einer netzübergreifenden verteilten Anwendung wird daher ein Werkzeug benötigt, welches den Informationsfluss global definiert und so die Ereigniskanaleigenschaften den jeweiligen Teilanwendungen konsistent zuweist. Ein solches Werkzeug kann die Konfiguration des Gateways weitestgehend automatisieren und die Kopplung von TTP/C- und CAN-Anwendungen weiter erleichtern.

Zur Durchsetzung weicher Echtzeitanforderungen muss der Prototyp um die dynamische Prioritätszuweisung nach dem EDF-Verfahren und um die Berechnung der Restgültigkeiten ein-

treffender Nachrichten erweitert werden. Die Unterstützung der harten Echtzeitkommunikation benötigt weiterhin eine Einführung einer globalen Zeit durch die Synchronisation des TTP/C- und des CAN-Netzes. Derzeit beschäftigt sich eine weitere Diplomarbeit in der Arbeitsgruppe mit der Zeitsynchronisation innerhalb eines CAN-Netzes unter RT-Linux. Die Kopplung beider Arbeiten kann eine Echtzeitkommunikation zwischen TTP/C und CAN mit minimalen Schwankungen der Nachrichtenverzögerung ermöglichen.

Literaturverzeichnis

[Alb04] Albert, A.: Comparision of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems. *Proceedings Embedded World Conference 2004*, S. 235–252, 2004.

[BK02] Beveridge, M.; Koopman, P.: Jini meets embedded control networking: a case study in portability failure. *Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems, 2002.(WORDS 2002).*, S. 11–18, 2002.

[Bor03] Borovicka, M.: Design of a gateway for the interconnection of real-time communication hierarchies. Master's thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.

[Bru03] Brudna, C.: Publisher/Subscriber Implementation for TCP/IP and CAN-Bus (pubsubs-1.4). 2003.

[CDK02] Coulouris, G.; Dollimore, J.; Kindberg, T.: *Verteilte Systeme - Konzepte und Design.* Pearson Studium, München, 2002.

[CES03] Claesson, V.; Ekelin, C.; Suri, N.: The Event-Triggered and Time-Triggered Medium-Access Methods. *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, 2003.

[Coo03] Cooling, J.: *Software Engeneering For Real-Time Systems.* Pearson Education Limited, Essex, 2003.

[Edw99] Edwards, W. K.: *Core Jini.* Prentice Hall, London, 1999.

[EHK+05] Elmenreich, W.; Haidinger, W.; Kirner, R.; Losert, T.; Obermaisser, R.; Tr\"odhandl, C.: TTP/A Smart Transducer Programming - A Beginner's Guide - Version 0.5. 2005.

[Ets94] Etschberger, K.: *CAN Controller-Area-Network Grundlagen, Protokolle, Bausteine, Anwendungen.* Carl Hanser Verlag, München Wien, 1994.

[Fle05] FlexRay Consortium: *FlexRay Communications System Protocol Specification Version 2.1 Revision A.* 2005.

[FMD+00] Führer, T.; Müller, B.; Dieterle, W.; Hartwich, F.; Hugel, R.; Walther, M.: Time Triggered Communication on CAN (Time Triggered CAN-TTCAN). *7th international CAN Conference*, 2000.

[GHJV96] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: *Entwurfsmuster*. Addison Wesley, 1996.

[GS94] Gergeleit, M.; Streich, H.: Implementing a Distributed High-Resolution Real-Time Clock using the CAN-Bus. *1st international CAN-Conference*, Band 94, 1994.

[Hea03] Heath, S.: *Embedded Systems Design*. Newnes, Oxford, 2003.

[HLS97] Harrison, T. H.; Levine, D. L.; Schmidt, D. C.: The design and performance of a real-time corba event service. In *OOPSLA '97: Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, S. 184–200. ACM Press, New York, NY, USA, 1997.

[HMFH00] Hartwich, F.; Müller, B.; Führer, T.; Hugel, R.: CAN Network with Time Triggered Communication. *7th international CAN Conference*, 2000.

[HMFH02] Hartwich, F.; Müller, B.; Führer, T.; Hugel, R.: Timing in the TTCAN Network. *8th international CAN Conference*, 2002.

[IEC03] IEC 61158-2: *Digital data communications for measurement and control - Fieldbus for use in industrial control systems - Part 2: Physical layer specification and service definition. Edition 3.0.* 2003.

[ISOa] ISO 9141:1989: *Road vehicles – Diagnostic systems – Requirements for interchange of digital information.*

[ISOb] ISO/IEC 7498-1:1994: *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.*

[KB02] Kaiser, J.; Brudna, C.: A publisher/subscriber architecture supporting interoperability of the can-bus and the internet. In *2002 IEEE International Workshop on Factory Communication Systems*. Väesteras, Schweden, August 28–30 2002.

[KB03] Kopetz, H.; Bauer, G.: The time-triggered architecture. *Proceedings of the IEEE*, Band 91, Nr. 1, S. 112 – 126, Januar 2003.

[KBM03] Kaiser, J.; Brudna, C.; Mitidieri, C.: A real-time event channel model for the can-bus. In *11th Annual Workshop on Parallel and Distributed Real-Time Systems*. Nice, France, April 22–26 2003. Held in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS 2003).

[KBM05] Kaiser, J.; Brudna, C.; Mitidieri, C.: Cosmic: A real-time event-based middleware for the can-bus. *Journal of Systems and Software*, Band 77, Nr. 1, S. 27–36, July 2005. Special issue: Parallel and distributed real-time systems.

[KEM00] Kopetz, H.; Elmenreich, W.; Mack, C.: A Comparison of LIN and TTP/A. *Research Report 4/2000*, 2000.

[KHK+97] Kopetz, H.; Hexel, R.; Krüger, A.; Millinger, D.; Schedl, A.: A synchronization strategy for a ttp/c controller. *SAE Congress and Exhibition, February 1997, Detroit, MI, USA SAE Paper No. 960120*, Feb. 1997.

[KJH+00] Kim, K.; Jeon, G.; Hong, S.; Kim, T.; Kim, S.: Integrating subscription-based and connection-oriented communications into the embedded CORBA for the CAN bus. *Real-Time Technology and Applications Symposium, 2000. RTAS 2000. Proceedings. Sixth IEEE*, S. 178–187, 2000.

[KM99] Kaiser, J.; Mock, M.: Implementing the real-time publisher/subscriber model on the controller area network (CAN). In *2nd. Int. Symp. on Object-Oriented Real-time distributed Computing (ISORC99)*. Saint-Malo, France, May 1999.

[KO87] Kopetz, H.; Ochsenreiter, W.: Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, Band 36, Nr. 8, S. 933–940, 1987.

[Kop97] Kopetz, H.: *Real-Time Systems Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Boston / Dordrecht / London, 1997.

[Kop98] Kopetz, H.: A Comparison of CAN and TTP. *Technische Universität Wien, Institut für Technische Informatik*, 1998.

[KR93] Kopetz, H.; Reisinger, J.: The Non-Blocking Write Protocol NBW: A Solution to a Real-Time Synchronization Problem. In *Proceedings of the 14th Real-Time Symposium*, S. 131–137, 1993.

[KS97] Krishna, C. M.; Shin, K. G.: *Real-Time Systems*. McGraw-Hill Companies, Inc., New York, 1997.

[LA99] Lonn, H.; Axelsson, J.: A Comparison of Fixed-Priority and Static Cyclic Scheduling for Distributed Automotive Control Applications. *Euromicro Conference on Real-Time Systems*, S. 142–149, 1999.

[LIN00] LIN Group: *LIN Specification Package Revision 1.2*. 2000.

[LK98] Livani, M. A.; Kaiser, J.: EDF consensus on CAN bus access for dynamic real-time applications. In *IPPS/SPDP Workshops*, S. 1088–1097, 1998.

[LKJ98] Livani, M.; Kaiser, J.; Jia, W.: Scheduling hard and soft real time communication in the controller area network (CAN). In *23rd IFAC/IFIP Workshop on Real Time Programming*. Shantou, China, June 23-25 1998.

[LR05] Liggesmeyer, P.; Rombach, D. (Hrsg.): *Software Engineering eingebetteter Systeme : Grundlagen - Methodik - Anwendungen*. Elsevier, Spektrum, Akademischer Verlag, München; Heidelberg, 2005.

[MFH+02] Müller, B.; Führer, T.; Hartwich, F.; Hugel, R.; Weiler, H.: Fault Tolerant TTCAN Networks. *8th international CAN Conference*, 2002.

[Nau05] Nauth, P.: *Embedded Intelligent Systems*. Oldenbourg Verlag, München; Wien, 2005.

[Obe02] Obermaisser, R.: CAN emulation in a time-triggered environment. In *Proceedings of the 2002 IEEE International Symposium on Industrial Electronics (ISIE)*. IEEE, Januar 2002.

[Obe05] Obermaisser, R.: *Event-Triggered and Time-Triggeres Control Paradigms.* Springer Science+Business Media, Inc., New York, 2005.

[OMG03] OMG: *RealTime-CORBA Specification - Version 2.0.* Object Management Group, 2003.

[PBG99] Peller, M.; Berwanger, J.; Grießbach, R.: *byteflight specification DRAFT Version 0.5.* BMW AG, 1999.

[PEP02] Pop, T.; Eles, P.; Peng, Z.: Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. *Proceedings of the tenth international symposium on hardware/software codesign*, S. 187–192, 2002.

[PK99] Poledna, S.; Kroiss, G.: TTP/C: Zeitsteuerung unterstützt Fehlertoleranz und Zusammensetzbarkeit. 1999.

[Pla02] Plankensteiner, M.: Sicherheit beim Bremsen und Lenken: Das Time-Triggered Protocol TTP und die dafür verfügbaren Produkte - ein aktueller Statusreport. 2002.

[PNK00] Poledna, S.; Niedersüss, M.; Kroiss, G.: TTP/C - Fault-Tolerant Real-Time Communication with integrated High-Level Services. 2000.

[Pop98] Pope, A.: *The CORBA Reference Guide.* Addison-Wesley, Reading, 1998.

[Rei02] Reißenweber, B.: *Feldbussysteme zur industriellen Kommunikation.* Oldenbourg Industrieverlag GmbH, München, 2002.

[Rob91] Robert Bosch GmbH: *CAN Specification Version 2.0.* 1991.

[RSH01] Richardson, P.; Seih, L.; Haniak, P.: A Real-Time Control Network Protocol for Embedded Systems Using Controller Area Network (CAN). *IEEE Electronics and Information Technology Conference, June*, 2001.

[Rus03] Rushby, J.: A Comparison of Bus Architectures for Safety-Critical Embedded Systems. 2003.

[RV95] Rufino, J.; Veríssimo, P.: A Study On The Inaccessibility Characteristics Of The Controller Area Network. *2nd international CAN-Conference*, 1995.

[Sch05] Scholz, P.: *Softwareentwicklung eingebetteter Systeme.* Springer Verlag, Berlin Heidelberg, 2005.

[Ser99] Serain, D.: *Middleware.* Springer, London, 1999.

[Tan03] Tanenbaum, A. S.: *Computer Networks.* Pearson Education, Inc., New Jersey, 2003.

[TB94] Tindell, K.; Burns, A.: Guaranteeing message latencies on control network (CAN). *1st international CAN-Conference*, 1994.

[TBW95] Tindell, K.; Burns, A.; Wellings, A.: Calculating Controller Area Network (CAN) message response times, 1995.

[TTA03] TTA Group: *Time-Triggered Protocol TTP/C High-Level Specification Document Protocol Version 1.1.* 2003.

[TTT05a] TTTech Computertechnik AG: *TTPBuild - The Node Design Tool for the Time-Triggered Protocol TTP.* 2005.

[TTT05b] TTTech Computertechnik AG: *TTPOS - The Time-Triggered Fault-Tolerant Operating System.* 2005.

[Tv03] Tanenbaum, A. S.; van Steen, M.: *Verteilte Systeme - Grundlagen und Paradigmen.* Pearson Studium, München, 2003.

[Ver94] Verssimo, P.: Ordering and timeliness requirements of dependable real-time programs, 1994.

[ZS97] Zuberi, K.; Shin, K.: Scheduling Messages on Controller Area Network for Real-Time CIM Applications. 1997.

[ZS06] Zimmermann, W.; Schmidgall, R.: *Bussysteme in der Fahrzeugtechnik - Protokolle und Standards.* Vieweg, Wiesbaden, 2006.

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe.

Magdeburg, den 8. November 2006

Stefan Metzlaff

Anhang A: Skripte für $^{TTP}Plan$ und $^{TTP}Build$

Auslesen des TTP/C-Nachrichtenplans aus $^{TTP}Plan$

```
out = open('./Output.txt', 'w')


macroTick = 5000 / 1000

for MuM in TTA.Cluster_Mode_uses_Message.links() :
 print >> out, "\n============\nMessage Name:", MuM.message.name, "\n=================="
 print >> out, "Period: ", MuM.a_period, "us Phase: ", MuM.a_phase,  "us Transmission Duration: ", ...    ⇒
...MuM.transmission_duration, "us Transmission End:", MuM.a_phase + MuM.transmission_duration, "us "
 for MuNF in TTA.Message_in_N_Frame.links():
  if(MuM.message.name == MuNF.message.name):
  print >> out, "N-Frame: ", MuNF.n_frame.name, "Offset (in Bit): ", MuNF.offset, "\nPart Position"...    ⇒
..., MuNF.part_pos
  for FiRS in TTA.Frame_in_R_Slot.links():
   if(MuNF.n_frame.name == FiRS.frame.name):
    print >> out, "R-Slot: ", FiRS.r_slot.name, " Channel: ", FiRS.channel
    print >> out, "R-Slot Phase: ", FiRS.r_slot.phase, "(", FiRS.r_slot.phase∗ macroTick ,"us )"
    print >> out, "R-Slot Duration: ", FiRS.r_slot.net_duration, "(", FiRS.r_slot.net_duration∗ mac...    ⇒
...roTick ,"us )"
    print >> out, "R-Slot End: ", FiRS.r_slot.phase+FiRS.r_slot.net_duration, "(", (FiRS.r_slot.pha...    ⇒
...se+FiRS.r_slot.net_duration)∗ macroTick ,"us )"

out.close
```

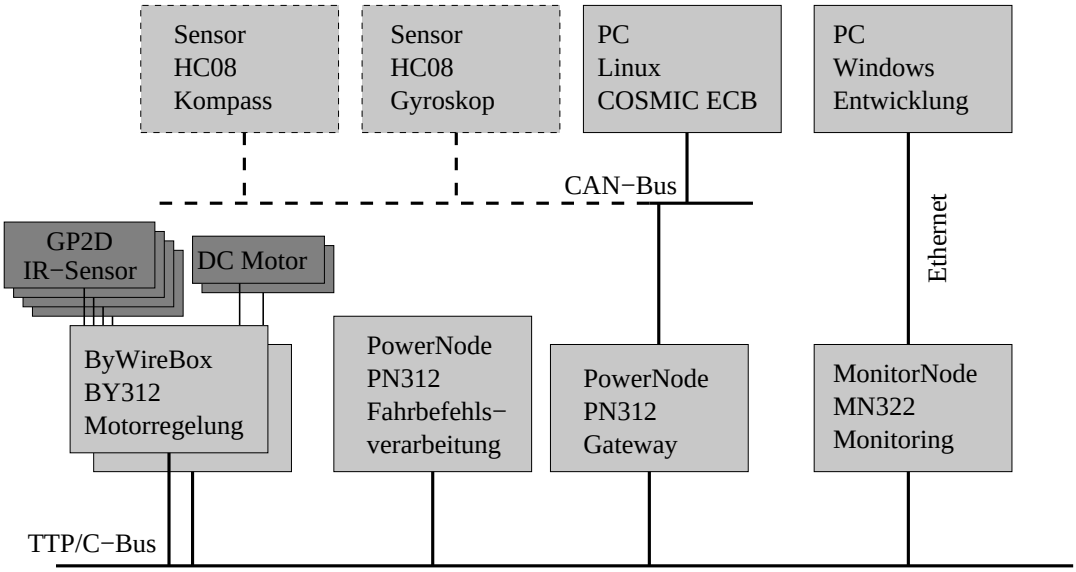
Auslesen des TTP/C-Taskplans aus $^{TTP}Build$

```
out = open('./Output.txt', 'w')

for TCrT in TTA.Node.Task_Chain_runs_Task.links() :
 print >> out, "\n================================"
 print >> out, "Task Name: ", TCrT.task.name
 print >> out, "================================"
 print >> out,  "Time Budget: ", TCrT.task.time_budget, "us Deadline: ", TCrT.deadline, "us \nEarli...    ⇒
...est Activation: ", TCrT.earliest_activation, "us Latest Completion: ", TCrT.latest_completion, "us "
 print >> out,  "\nTask Chain Name: ", TCrT.task_chain.name, "  Number in Chain: ", TCrT.sequence_n...    ⇒
...umber ,"\nTask_Chain Phasis: ", TCrT.task_chain.phase, "us Task Chain Budget: ", TCrT.task_chain.tim...    ⇒
...e_budget, "us Chain End:",  TCrT.task_chain.phase+TCrT.task_chain.time_budget, "us "

out.close
```

Anhang B: Roboterbeschreibung



Hardwareaufbau des Roboters

Der entwickelte mobile Roboter baut auf dem Fahrgestell der 6-rädrigen Version des 'Volks-Bot RT' vom Fraunhofer Institut für Autonome Intelligente Systeme auf. Die Motorregelung wird von zwei TTTech By-Wire-Boxen BY312 vorgenommen, dabei regelt jede By-Wire-Box einen Motor. Der Roboter wird über zwei Maxon DC RE-40 Motoren mit einem 74:1 Getriebe angetrieben. Ein Motor ist mit den drei Rädern einer Seite verbunden. Pro By-Wire-Box sind zwei SHARP GP2D12 Infrarot-Abstandssensoren für die Kollisionsvermeidung angeschlossen. Der TTP/C-Cluster besteht aus den zwei By-Wire-Boxen und zwei TTTech PowerNodes PN312. Ein PowerNode übernimmt die Verarbeitung der Fahrbefehle des Roboters. Er teilt damit den By-Wire-Boxen die einzustellende Geschwindigkeit mit. Der andere PowerNode agiert als Gateway zwischen dem CAN- und dem TTP/C-Netz. Die Spannungsversorgung erhält der Roboter aus drei 12V Batterien. Wovon zwei die By-Wire-Boxen und die Motoren betreiben. Die dritte Batterie versorgt die zwei PowerNodes, die vier Abstandssensoren und einen MonitorNode. Ein Laptop übernimmt die Brokerfunktionen im CAN-Netz und setzt die CAN-Fahrbefehle für die Roboterbewegungen ab. Die Fahrbefehle werden vom Gateway für die Verarbeitung ins TTP/C-Netz weitergeleitet. Das Gateway stellt dem CAN-Netz Informationen von der Fahrbefehlsverarbeitung über die aktuelle Geschwindigkeit und die Entfernung zum Ziel zur Verfügung. Eine Erweiterung der CAN-Anwendung durch z.B. ein Gyroskop und einen Kompass (beide implementiert auf einem HC08) ist Gegenstand aktueller Bemühungen. Der

TTTech MonitorNode MN322 ermöglicht die Überwachung der TTP/C-Kommunikation auf dem Entwicklungsrechner. Auf dem Entwicklungsrechner sind die TTTech Entwicklungstools (^{TTP}OS 4.10, $^{TTP}Plan$ 5.4, $^{TTP}Build$ 5.4, $^{TTP}Load$ 6.4 und $^{TTP}View$ 6.4) und der Wind River Diab C++ Compiler 5.3.1 für den MPC555 installiert.