Beiträge zum Stuttgarter Maschinenbau

Andreas Eckhardt

Überwachter bidirektionaler Datenaustausch in industriellen Echtzeit-Kommunikationsarchitekturen

stuttgarter **maschinenbau**
interdisziplinär und vielfältig

Universität Stuttgart
Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen (ISW)

Universität Stuttgart

Beiträge zum Stuttgarter Maschinenbau **Band 8**

Herausgeber: Prof. Dr.-Ing. Oliver Riedel
Prof. Dr.-Ing. Alexander Verl
Jun.-Prof. Dr. rer. nat. Andreas Wortmann

Andreas Eckhardt

Überwachter bidirektionaler Datenaustausch in industriellen Echtzeit-Kommunikationsarchitekturen

Fraunhofer Verlag

Kontaktadresse:
Institut für Steuerungstechnik der Werkzeugmaschinen
und Fertigungseinrichtungen ISW
Seidenstr. 36
70174 Stuttgart
info@isw.uni-stuttgart.de
https://www.isw.uni-stuttgart.de

Titelbild: Andreas Eckhardt

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.de abrufbar.

ISSN: 2750-655X
ISBN: 978-3-8396-1816-5

D 93

Zugl.: Stuttgart, Univ., Diss., 2021

Druck und Weiterverarbeitung:
Fraunhofer Verlag, Mediendienstleistungen

Für den Druck des Buches wurde chlor- und säurefreies Papier verwendet.

Nobelstraße 12
70569 Stuttgart
verlag@fraunhofer.de
www.verlag.fraunhofer.de

als rechtlich nicht selbständige Einheit der

Fraunhofer-Gesellschaft zur Förderung
der angewandten Forschung e.V.
Hansastraße 27 c
80686 München
www.fraunhofer.de

Geleitwort

Die deutsche Wirtschaft ist weltweit bekannt für ihren Anlagen- und Maschinenbau. Dabei ist die Universität Stuttgart mit ihren beiden Maschinenbaufakultäten – unter deren Dach sich 42 Institute befinden – die größte universitäre Einrichtung für den Maschinenbau in Deutschland. Unsere wissenschaftliche Exzellenz stützt sich dabei auf unsere zahlreichen Promovierenden und ihre hervorragenden Dissertationen. Viele dieser Dissertationen entstehen in lokaler, nationaler und internationaler Zusammenarbeit mit renommierten Universitäten und außeruniversitären Einrichtungen wie dem Deutschen Zentrum für Luft- und Raumfahrt, der Fraunhofer-Gesellschaft und der Max-Planck-Gesellschaft. Dabei reicht das inhaltliche Spektrum der Dissertationen von Biotechnik, Energietechnik, Fahrzeugtechnik, Kybernetik und Systemtechnik, Produktentwicklung und Konstruktionstechnik, Produktionstechnik bis hin zur Verfahrenstechnik und stützt sich auf die sechs Forschungsschwerpunkte Advanced Systems Engineering, Autonome Produktion, Software-Defined Manufacturing, Resiliente Versorgung, Biointelligenz und Dekarbonisierung der Industrie. Die Ergebnisse aus den Dissertationen zielen darauf ab, kunden-, produkt-, prozess- und mitarbeiterorientierte Technologie zielgerichtet und zeitnah zu entwickeln und anzuwenden.
Viele der im Rahmen der Forschungsarbeiten an den Instituten entstandenen Dissertationen werden in diesen »Beiträgen zum Stuttgarter Maschinenbau« veröffentlicht. Die beiden Fakultäten des Stuttgarter Maschinenbaus wünschen den Promovierenden, dass ihre Dissertationen aus dem Bereich des Maschinenbaus in der breiten Fachwelt als maßgebliche Beiträge wahrgenommen werden und so den Wissensstand auf ein neues Niveau heben.

Für den Stuttgarter Maschinenbau

Stefan Weihe
Prodekan Fakultät 4

Oliver Riedel
Prodekan Fakultät 7

Vorwort der Herausgeber

Innerhalb der Reihe »Beiträge zum Stuttgarter Maschinenbau« berichtet das Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen an der Universität Stuttgart (ISW) über seine Forschungsergebnisse. Das Institut beschäftigt sich in vielfältiger Form mit Steuerungs- und Automatisierungstechnik sowie dem Einsatz von modernen Methoden des Informationsmanagements. Dabei stehen Grundlagenforschung und anwendungsorientierte Entwicklung in einem stetigen Austausch, wodurch ein kontinuierlicher Technologietransfer in die Praxis sichergestellt wird.

Die am ISW entstandenen Dissertationen werden damit unter erweitertem Namen und inzwischen in vierter Generation in der bewährten Konzeption, die der Gründer des ISW Prof. Stute und sein Nachfolger Prof. Pritschow 1972 begonnen haben, durch die heutige Institutsleitung fortgesetzt.

Herrn Andreas Eckhardt M.Sc. möchten wir für die geleistete Arbeit danken, dem Verlag für die Aufnahme dieser Schriftenreihe in sein Angebot und der Druckerei für die saubere und zügige Ausführung. Möge das Buch von der Fachwelt gut aufgenommen werden.

Alexander Verl

Oliver Riedel

Überwachter bidirektionaler Datenaustausch in industriellen Echtzeit-Kommunikationsarchitekturen

Von der Fakultät Konstruktions-, Produktions- und Fahrzeugtechnik der Universität Stuttgart zur Erlangung der Würde eines Doktoringenieurs (Dr.-Ing.) genehmigte Abhandlung

vorgelegt von

Andreas Eckhardt

aus Biberach an der Riß

Hauptberichter:	Prof. Dr.-Ing. Alexander Verl
Mitberichter:	Prof. Dr.-Ing. Ludwig Leurs
Tag der mündlichen Prüfung:	09.11.2021

Institut für Steuerungstechnik der Werkzeugmaschinen und Fertigungseinrichtungen der Universität Stuttgart

2022

Vorwort

Diese Arbeit widme ich meinen Eltern.

Vielen Dank für alles.

Inhalt

Abkürzungsverzeichnis

Akronyme	**Beschreibung**
AMQP	Advanced Message Queuing Protocol
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Programming Interface
APU	Application Processing Unit
AR	Application Relationship
AR ASE	Application Relationship Application Service Element
AREP	Application Relationship End Point
AT	Acknowledge Telegram
BBUB	Buffer Buffer Unconfirmed Bidirectional
BBUU	Buffer Buffer Unconfirmed Unidirectional
BBUU-OM	Buffer Buffer Unconfirmed Unidirectional: One to Many
C2C	Steuerung-zu-Steuerung
C2D	Steuerung-zu-Feldgerät
C2E	Steuerung-zu-Enterprise Level
C2H	Steuerung-zu-HMI
C-CON	Connection Control
CD	Collision Detection
CIM	Computer-Integrated Manufacturing
CMCTL	Context Management Protocol Machine Controller
CPM	Consumer Protocol Machine
CPU	Central Processing Unit
CR	Communication Relationship

Akronyme	**Beschreibung**
CREP	Communication Relationship End Point
CSMA	Carrier Sense Multiple Access
CSN	Coordinated Shared Network
CTLIO	Context Management Input Oputput Protocol Machine Controller
CTLSM	Context Management Client Protocol Machine
DA	Destination Address
DHt	DataHoldFactor
DMA	Direct Memory Access
FAL	Fieldbus Application Layer
FCS	Frame-Check-Sequence
FPGA	Field Programmable Gate Array
FSPM	FAL Service Protocol Machine
GCL	Gate Control List
GEDF	Global Earliest Deadline First
GPOS	General Purpose Operating System
gPTP	generalized Precision Time Protocol
HDR	Header
HMI	Human Machine Interface
IEEE	Institute of Electrical and Electronics Engineers
IIC	Industrial Internet Consortium
IIoT	Industrial Internet of Things
IP	Internet Protocol
IRQ	Interrupt Request
IT	Information Technology
JSON	JavaScript Object Notation

Akronyme	**Beschreibung**
LAN	Local Area Network
MDT	Master Data Telegram
MQTT	Message Queuing Telemetry Transport
NIC	Network Interface Card
OOAD	Objektorientierte Analyse und Design
OPC	Open Platform Communications
OPC FLC	OPC Field Level Communication
OPC UA	Open Platform Communications Unified Architecture
OPC UA PubSub	OPC UA Publish Subscribe
OSI	Open Systems Interconnection
OT	Operational Technology
PDU	Protocol Data Unit
PPM	Provider Protocol Machine
Pre	Preamble
PTP	Precision Time Protocol
QoS	Quality of Services
QQCB-CL	Queue Queue Confirmed Bidirectional: Connectionless
QQCB-CO	Queue Queue Confirmed Bidirectional: Connection-Oriented
RT	Real-Time
RTAI	Real-Time Application Interface
RTC	Real-Time Channel
RTHAL	Real-Time Hardware Abstraction Layer
RTOS	Real-Time Operating Systems
SA	Source Address
S-DEV	Status-Device

Akronyme	**Beschreibung**
SDU	Service Data Unit
SFD	Start Frame Delimiter
SOA	Service-oriented Architecture
SVC	Service Channel
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSA	Transmission Selection Algorithm
TSN	Time-Sensitive Networking
UADP	UA Datagram Protocol
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network
WCET	Worst-Case Execution Time

Abbildungsverzeichnis

Tabellenverzeichnis

Kurzinhalt

Industrie 4.0 fordert eine durchgängige Vernetzung aller Ebenen der Automatisierungspyramide. Dies ist eine große Chance und eine Herausforderung für die Feldebene, da dort bisher eine Vielzahl von Bussystemen eingesetzt wird, die untereinander nicht kompatibel sind. Um auch hier eine durchgängige Kommunikationslösung zu bieten, engagieren sich viele Hersteller von Automatisierungslösungen bei dem neu entstehenden Kommunikationsstandard OPC Field Level Communication (OPC FLC). OPC FLC basiert auf Open Platform Communications Unified Architecture (OPC UA) und TSN. Um die Anforderungen der Feldebene zu erfüllen, wurde OPC UA um die entkoppelte Kommunikation OPC UA Publish Subscribe (OPC UA PubSub) erweitert. OPC UA PubSub wurde im Jahr 2018 veröffentlicht und ist dementsprechend noch ein neuer Standard, der nach und nach erweitert wird. Dadurch dass OPC UA PubSub auf einer entkoppelten Kommunikation basiert, sind die kommunizierenden Entitäten einander unbekannt. Damit die überlagerte Applikation jedoch auf Unregelmäßigkeiten bei der Kommunikation reagieren kann, ist eine gewisse Überwachung nötig. Bislang unterstützt OPC UA PubSub nur die Überwachung des Datenaustauschs auf Seiten des Subscribers. Dem Publisher ist nicht bekannt, ob seine Nachrichten bei den gewünschten Entitäten ankommen. Es fehlt eine Überwachung des Datenaustauschs beim Publisher. An dieser Stelle knüpft diese Arbeit an.

In einem ersten Schritt wird eine Evaluierungsplattform entworfen und eine Methode zur Erstellung einer Leistungsmetrik entwickelt. Die gewonnenen Erkenntnisse bieten die Basis für die weitere Entwicklung der Überwachung.

In einem zweiten Schritt wird die Überwachung des Datenaustauschs der Busprotokolle PROFINET IRT und Sercos III untersucht, verglichen und abstrahiert.

In einem dritten Schritt wird, ausgehend von der Abstraktion der Busprotokolle, eine Überwachung des Datenaustauschs zyklischer Kommunikation basierend auf OPC UA PubSub abgeleitet.

Im letzten Schritt wird die entwickelte Lösung validiert, indem zunächst eine Konkretisierung einer 1-zu-N-Kardinalität gefolgt von einer Simulation mit der Simulationssoftware Simulink durchgeführt wird. Außerdem wird im Rahmen eines implementierten Testaufbaus die Umsetzbarkeit demonstriert.

Short Summary

With the digitalization of manufacturing, the fourth industrial revolution – Industry 4.0 – begins. Industry 4.0 demands continuous communication between all levels of the automation pyramid. So far, a large number of various bus systems, that are not compatible with one another, have been used on the field level, disrupting the continuous communication. In order to offer a single continuous communication solution on the field level, many manufacturers of automation solutions rely on the newly emerging communication standard OPC Field Level Communication (OPC FLC). OPC FLC is based on Open Platform Communications Unified Architecture (OPC UA) and Time-Sensitive Networking (TSN). In order to meet the requirements at the field level, OPC UA has been extended to include the decoupled communication patternOPC UA Publish Subscribe (OPC UA PubSub). OPC UA PubSub was published in 2018 and is therefore rather a new standard that is gradually being further extended.

Due to the fact that OPC UA PubSub is based on decoupled communication, the communicating entities are unknown to each other. However, a certain amount of monitoring is necessary so that the higher-level application can react to irregularities in the communication. So far, OPC UA PubSub has only been supporting the monitoring of data exchange on the part of the subscriber. The publisher does not know whether his messages arrive at the desired entities. Mechanisms that are able to monitor the data exchange at the publisher are missing. This thesis addresses this problem by implementing monitoring mechanisms for a cyclic communication based on OPC UA PubSub.

In a first step, an evaluation platform is designed and a method to create a performance metric is developed. This is necessary because there is only little research done on OPC UA PubSub and the performance metric is unknown. The gained knowledge provides the foundation for further development of the monitoring mechanisms.

In a second step, the monitoring mechanisms of the bus systems PROFINET IRT and Sercos III are analyzed and abstracted.

In a third step, based on the abstraction of the bus systems, monitoring mechanisms for a cyclic communication based on OPC UA PubSub are derived.

In the last step, the developed solution is validated by giving a practical example followed by performing a simulation with the software Simulink. In addition to the simulation, the feasibility is demonstrated as part of an implemented prototype, which is partly used in a public funded project.

Begriffsdefinitionen

Begriff	Definition
Antwortzeit Response Time	Bei einer Instanz die Zeitspanne zwischen dem Zeitpunkt, zu dem sie die Erteilung eines Auftrags an eine andere Instanz beendet, und dem Zeitpunkt, zu dem bei der auftraggebenden Instanz die Übergabe des Ergebnisses der Auftragsabwicklung oder einer Mitteilung darüber an sie beginnt (Norm DIN 44300-7).
Auftrag	Von einer Person oder Funktionseinheit an eine Person oder Funktionseinheit gerichtete, in eine vereinbarte Form gefasste Forderung, eine bestimmte Datenverarbeitungsleistung zu erbringen (Norm DIN 44300-1).
Bearbeitungszeit Processing Time	Bei einer Instanz die Summe der Zeitspannen, während sie denselben Auftrag bearbeitet (Norm DIN 44300-7).
Betriebssystem Operating System	Programme eines digitalen Rechensystems, die zusammen mit den Eigenschaften der Rechenanlage die Grundlage der möglichen Betriebsarten des digitalen Rechensystems bilden und insbesondere die Ausführung von Programmen steuern und überwachen (Norm DIN 44300-4).
Bezeichner Identifier	Digitale Daten zur Identifikation von Objekten (Norm DIN 44300-3).
Bit Bit	Jedes der Zeichen eines Zeichenvorrats von zwei Zeichen (Norm DIN 44300-2).
Byte Byte	n-Bit-Zeichen bei dem n fest vorgegeben ist (Norm DIN 44300-2).
Dateikopf Header	Zusatzinformation vor den Nutzdaten.
Daten Data	Gebilde aus Zeichen oder kontinuierliche Funktionen, die aufgrund bekannter oder unterstellter Abmachungen Informationen darstellen, vorrangig zum Zweck der Verarbeitung oder als deren Ergebnis (Norm DIN 44300-2).

Datensicherheit Data Security	Sachlage, bei der Daten unmittelbar oder mittelbar so weit wie möglich vor Beeinträchtigung oder Missbrauch bewahrt sind, [...] (Norm DIN 44300-1).
Datentyp Data Type	Daten-Bauart zusammen mit vorgegebenen Operationen bei denen Ausprägungen der Bauart als Operanden beteiligt sind (Norm DIN 44300-3).
Datenverarbeitungs-leistung	Verarbeitung von Daten in Übereinstimmung mit gegebenen Arbeitsvorschriften innerhalb einer bestimmten Zeitspanne (Norm DIN 44300-1).
Durchsatz Throughput	Anzahl der Aufträge, die bei gegebenem Auftragsprofil von einer Instanz während einer Zeiteinheit abgewickelt werden (Norm DIN 44300-9).
Entkoppelte Kommunikation	Kommunikation bei denen sich die Entitäten nicht kennen.
Gerätestatus Device Status	Statuswort im Acknowledge Telegram (AT).
Gerätesteuerung Device Control	Steuerwort im Master Data Telegram (MDT).
Horizontale Kommunikation	Kommunikation innerhalb einer Ebene.
Industrie 4.0 Industry 4.0	Industrie 4.0 ist die vierte industrielle Revolution. Sie vernetzt intelligente Maschinen untereinander.
Informations-technologie (IT) Information Technology	IT ist in der Regel Software, die zur Datenspeicherung und Datenverarbeitung eingesetzt wird (Garimella 2018; Bruckner et al. 2019).
Instanz Instance	Funktionseinheit, die Aufträge erteilt oder erhält, erhaltene Aufträge ablehnt oder annimmt und angenommene Aufträge ganz oder teilweise selbst ausführt, weitergibt oder bei Unausführbarkeit zurückgibt (Norm DIN 44300-1).
Kompilierer Compiler	Übersetzer, der Quellanweisungen, die in einer nicht-maschinenorientierten Programmiersprache abgefasst sind, in Zielanweisungen einer maschinenorientierten Programmiersprache umwandelt (Norm DIN 44300-4).
Latenzzeit Latency	Zeitspanne bei einer Instanz zwischen dem Zeitpunkt, zu dem ein Auftrag, bestimmte Daten abzugeben oder anzunehmen, als ihr erteilt gilt, und dem Zeitpunkt, zu dem die Abgabe bzw. Annahme dieser Daten beginnt (Norm DIN 44300-7).

Maximale Laufzeit Worst-Case Execution Time	Die maximal aufgetretene Laufzeit.
Nachricht Message	Gebilde aus Zeichen oder kontinuierliche Funktionen, die aufgrund bekannter oder unterstellter Abmachungen Informationen darstellen und die zum Zweck der Weitergabe als zusammengehörig angesehen und deshalb als Einheit betrachtet werden (Norm DIN 44300-2).
Nutzdaten Payload	Daten die zwischen zwei Entitäten ausgetauscht werden und keine Steuer- oder Protokollinformationen enthalten.
Operative Technologie (OT) Operational Technology	OT kann Hardware und Software sein, die direkt an der Überwachung, Steuerung und Regelung von Prozessen beteiligt ist(Garimella 2018; Norm IEEE 1934).
Puffer Buffer	Speicher, der Daten vorübergehend aufnimmt, die von einer Funktionseinheit zu einer anderen übertragen werden (Norm DIN 44300-6).
Paketumlaufzeit Round Trip Time	Dauer einer kreisförmigen Kommunikation.
Realzeitverarbeitung Real-Time Processing	Verarbeitungsart, bei der Programme zur Verarbeitung anfallender Daten ständig ablaufbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind(Norm DIN 44300-9).
Schnittstelle Interface	Gedachter oder tatsächlicher Übergang an der Grenze zwischen zwei gleichartigen Einheiten, wie Funktionseinheiten, Baueinheiten oder Programmbausteinen, mit den vereinbarten Regeln für die Übergabe von Daten oder Signalen (Norm DIN 44300-1).
Speicherdirektzugriff Direct Memory Access	Zugriffsart, die über ein Bussystem direkt auf den Speicher zugreift.
Stapelverarbeitung Batch Processing	Verarbeitungsart, bei der Aufträge vollständig beschrieben und als Ganzes erteilt sein müssen, bevor mit ihrer Abwicklung begonnen werden kann (Norm DIN 44300-9).
Übertragungszeit	Bei zwei Funktionseinheiten die Zeitspanne zwischen dem Beginn der Abgabe von Daten durch eine Funktionseinheit und dem Ende der Annahme dieser Daten durch die andere (Norm DIN 44300-7).
Vertikale Kommunikation	Kommunikation über Ebenengrenzen hinweg.

Verweis Reference	Logische Verbindung zwischen zwei Datenobjekten.
VLAN	Ein VLAN ist ein logisches Teilnetz innerhalb eines Netzwerks.
Zeitgeber Timer	Funktionseinheit, die absolute, relative oder inkrementelle Zeitangaben macht (Norm DIN 44300-5).
Zeitmultiplexverfahren Time-Division Multiplex Access	Es werden die Daten verschiedener Sender in Zeitschlitzen auf einem Kanal übertragen.
Zyklische Blockprüfung Cyclic Redundancy Check	Prüfung einer Folge von Binärzeichen auf Fehler unter Verwendung eines geeigneten Polynoms (Norm DIN 44300-8).
Zykluszeit Cycle Time	Zeitspanne bei einer Funktionseinheit zwischen dem Beginn zweier aufeinanderfolgender gleichartiger, zyklisch wiederkehrender Vorgänge (Norm DIN 44300-7).

1 Einleitung und Problemstellung

Industrie 4.0 ist die vierte industrielle Revolution und verlangt eine durchgängige Vernetzung aller Geräte und die digitale Kommunikation über alle Ebenen der Automatisierungspyramide hinweg. Unter Industrie 4.0 wird die intelligente Vernetzung von Maschinen verstanden. Durch die durchgängige Kommunikation löst sich die Automatisierungspyramide auf Netzwerkebene auf (Plattform Industrie 4.0 2018). Funktional betrachtet bleiben die Ebenen der Automatisierungspyramide jedoch bestehen. Es existieren viele Definitionen der Automatisierungspyramide, die auf der Computer-Integrated Manufacturing (CIM)-Pyramide aus den 70er und 80er Jahren basieren (Meudt et al. 2017). Die verschiedenen Definitionen unterscheiden sich meist durch die Stärke der Aggregation der Ebenen. Dementsprechend sind Definitionen zu finden, die nur aus drei Ebenen bestehen (Bauernhansl et al. 2014) und solche, die bis zu sieben Ebenen definieren (Norm IEC PAS 63088). Die Definition der Automatisierungspyramide nach IEC 62264 (Norm DIN EN 62264-1) ist weit verbreitet und wird auch in dieser Arbeit als Automatisierungspyramide zugrunde gelegt. Sie beschreibt anhand von fünf Ebenen die Struktur heutiger industrieller Fertigungen. Mit zunehmender Prozessnähe nimmt die Echtzeitanforderung zu und die Datenmenge ab (Pritschow 2006). Jede Ebene hat dabei eigene Aufgaben. In Abbildung 1.1 sind die einzelnen Ebenen der Automatisierungspyramide dargestellt und werden im Nachfolgenden kurz beschrieben.

Die **Ebene 0 – Feldebene** erfasst und passt den Produktionsprozess an. Es werden Eingangssignale erfasst und Ausgangssignale geschaltet. Der zeitliche Rahmen der Kommunikation befindet sich im einstelligen Millisekundenbereich bis zweistelligen Mikrosekundenbereich.

Die **Ebene 1 – Steuerungsebene** überwacht, führt und steuert den Produktionsprozess. Der zeitliche Rahmen der Kommunikation beträgt hier Stunden bis Millisekunden.

Die **Ebene 2 – Prozessleitebene** verwaltet die Arbeitsabläufe zur Herstellung der gewünschten Endprodukte. Es wird der Produktionsprozess optimiert, indem die Produktionsleistung analysiert und entsprechend darauf reagiert wird. Typische Aufgaben sind die Produktionseinplanung, Feinplanung und Betriebssicherheit. Der zeitliche Rahmen der Kommunikation beträgt hier Tage, Schichten, Stunden bis Sekunden.

Die **Ebene 3 – Betriebsleitebene** ist für die Feinplanung der Produktion, den Materialverbrauch, die Anlieferung von Gütern und den Versand von Produkten verantwortlich. Der zeitliche Rahmen beträgt hier Monate, Wochen und Tage. Zum Erfassen der Produktionsdaten kommuniziert die Betriebsleitebene mit der untergeordneten Produktionsleitebene.

Die **Ebene 4 – Unternehmensebene** ist für die Grobplanung der Produktion und die Bestellabwicklung verantwortlich.

Der **Technische Prozess** beschreibt den Produktionsprozess und ist die Grundlage für die Ebenen 0 bis 4.

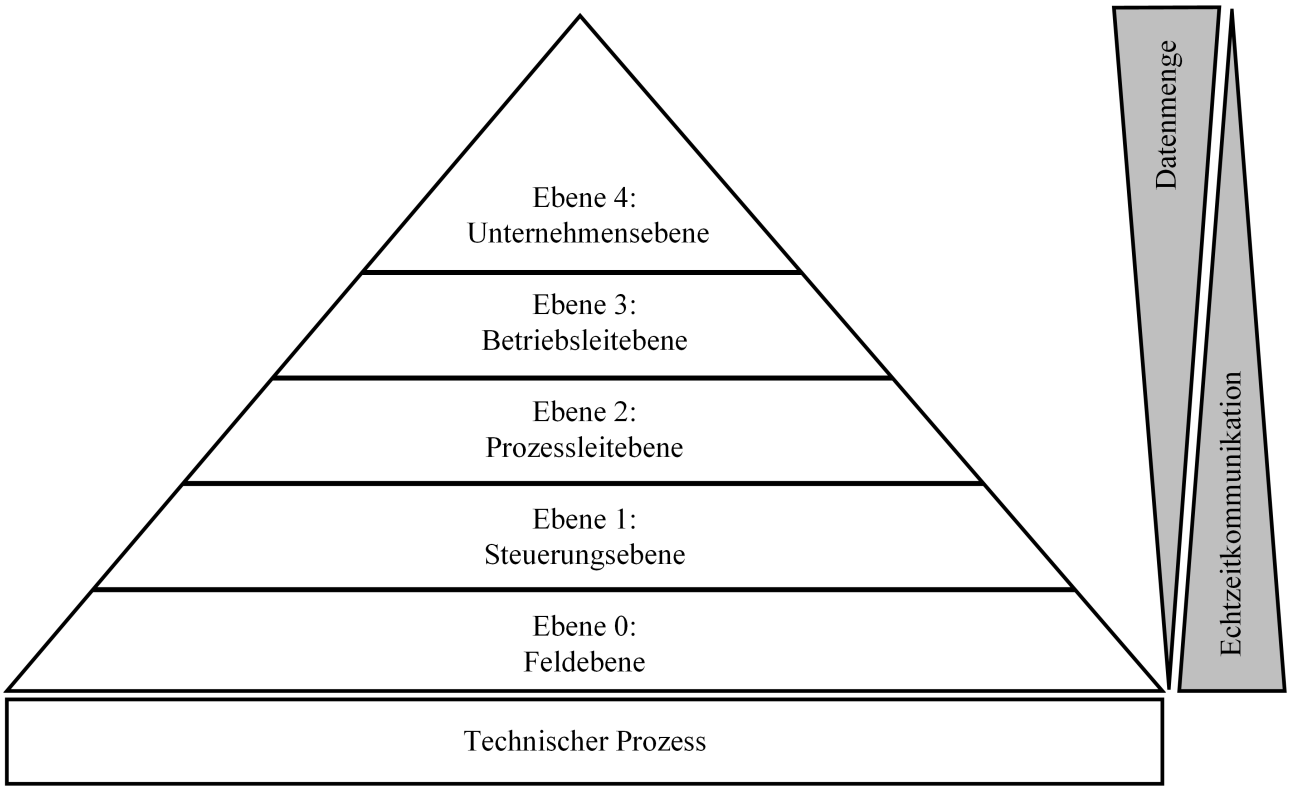


Abbildung 1.1: Hierarchie der Automatisierungspyramide (Norm DIN EN 62264-1).

Die Funktionen der Ebenen 0 und 2 werden zur Überwachung, zum Betrieb und zur Steuerung von Anlagen verwendet. Technologien dieser Art werden als Operational Technology (OT) bezeichnet. OT kann Hardware und Software sein, die direkt an der Überwachung, Steuerung und Regelung von Prozessen beteiligt ist (Garimella 2018; Norm IEEE 1934). In den restlichen Ebenen 3 und 4 wird Information Technology (IT) verwendet. IT ist in der Regel Software, die zur Datenspeicherung und Datenverarbeitung eingesetzt wird (Garimella 2018; Bruckner et al. 2019). Durch die zunehmende Digitalisierung wird die vertikale und horizontale Kommunikation immer wichtiger in der Automatisierungspyramide (Wollschlaeger et al. 2017; Wollschlaeger et

al. 2018; Siepmann et al. 2016). Unter der vertikalen Kommunikation wird die Kommunikation über die Grenzen einer Ebene hinweg verstanden. Die horizontale Kommunikation beschreibt die Kommunikation innerhalb einer Ebene. Es hat sich zwar Ethernet auf sämtlichen Ebenen durchgesetzt, trotzdem werden IT und OT bisher in getrennten Netzwerken betrieben. Grund hierfür sind die sehr hohen deterministischen Ansprüche der Feldebene an die Kommunikation. Damit der Datenverkehr der IT nicht den deterministischen Datenverkehr der OT stört, werden beide Netzwerke voneinander getrennt. Negative Folgen dieser Trennung sind z. B. ein erhöhter Aufwand bei der Datenübertragung über die Grenzen einer Ebene hinweg. Oft werden Gateway-Lösungen eingesetzt, die als Schnittstelle zwischen einzelnen Ebenen fungieren. Mit dem Gedankenansatz von Industrie 4.0 steigt das Verlangen nach konvergierenden Netzwerken. Als Folge müssen sich die heutigen OTs – vor allem die Bussysteme – wandeln. Vor der Zeit von Bussystemen bestand ein industrielles Netzwerk aus einzelnen Verbindungen zwischen Steuerungen und den zugehörigen Sensoren und Aktoren. Die daraus resultierenden Verkabelungskosten waren enorm. Aus dieser Not heraus wurde in der Mitte der 80er Jahre angefangen, sogenannte Bussysteme zu entwickeln. Die einzelnen Branchen mit ihren spezifischen Anforderungen entwickelten jeweils eigene Bussysteme. Das Resultat sind viele Lösungen, die nicht kompatibel zueinander sind (Norm ISO 11898-1; Norm DIN EN 61158-3-16). Außerdem stiegen die Anforderungen an die industriellen Kommunikationssysteme. Horizontale und vertikale Kommunikation spielten eine immer größere Rolle. Es wurde die Automatisierungspyramide definiert, die die einzelnen Fertigungsbereiche in Ebenen unterteilt. Da auf den Ebenen der Automatisierungspyramide unterschiedliche Netzwerkkonzepte verwendet wurden – Non-Ethernet-Bussysteme auf der Feldebene und Ethernet-basierte Lösungen im Office-Bereich – stellte sich die vertikale Integration als äußerst schwierig heraus. Um die Integrationskosten zu senken, wurden ab dem Jahr 2000 Ethernet-basierte Bussysteme entwickelt. In Abbildung 1.2 sind die Marktanteile von Ethernet-basierten und Non-Ethernet-basierten Bussystemen im Jahr 2018 dargestellt. Im Jahr 2008 lag der globale Marktanteil von Ethernet-basierten Bussystemen bei 19 % (Lechler 2011), 2017 schon bei 48 % (Drahos et al. 2018) und 2018 bei 52 % (HMS 2019). Die Entwicklung zeigt, dass die Ethernet-basierten Bussysteme die Non-Ethernet-basierten Bussysteme ablösen. Die Gründe für das schnelle Wachstum sind die gestiegene Forderung nach höherer Bandbreite sowie das Kostenersparnis durch günstige Hardware.

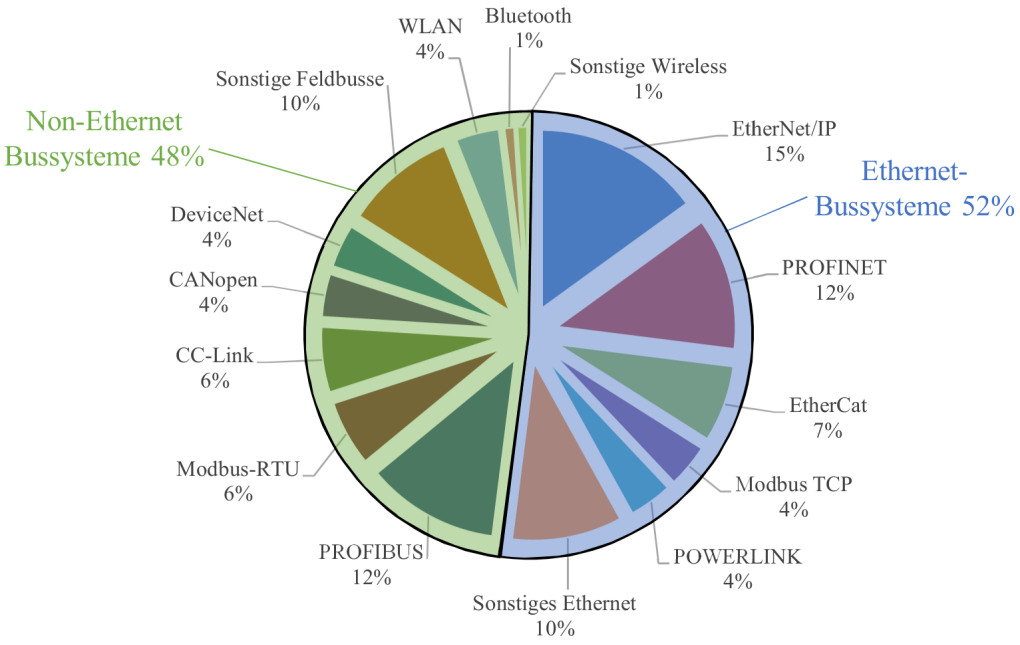


Abbildung 1.2: Marktanteile industrieller Bussysteme weltweit im Jahr 2018 (HMS 2019).

Bisher ist mit der Einführung von Ethernet-basierten Bussystemen das Problem der vertikalen Integration nicht gelöst, da die entwickelten Ethernet-basierten Bussysteme inkompatibel zueinander sind (Wollschlaeger et al. 2017). Hauptgrund dafür ist, dass Ethernet das zufällige Buszuteilungsverfahren Carrier Sense Multiple Access (CSMA) mit Collision Detection (CD) nutzt und somit Ethernet der nötige Determinismus fehlt (Prytz et al. 2005). Um die Echtzeitfähigkeit garantieren zu können, verwenden die Busprotokolle Anpassungen, die inkompatibel zu Standard-Ethernet sind. Dadurch können die Busprotokolle in drei Echtzeitklassen unterteilt werden. Prinzipiell nimmt der Determinismus von Klasse 1 bis Klasse 3 zu. In Abbildung 1.3 sind die drei Klassen dargestellt.

Klasse 1: Bussysteme dieser Klasse nutzen das Standard-Ethernet mit dem Internet Protocol (IP) als Netzwerkschicht und dem Transmission Control Protocol (TCP) oder dem User Datagram Protocol (UDP) als Transportschicht. Durch die Verwendung von IP mit TCP (Norm RFC 793) oder UDP ist die Echtzeitfähigkeit jedoch eingeschränkt. Es können Zykluszeiten von bis zu 100 ms erreicht werden. Parallel zur Buskommunikation kann weitere Standard-Ethernet-Kommunikation der IT ausgeführt werden. Modbus und Ethernet/IP sind Beispiele für Bussysteme in dieser Klasse (Jasperneite et al. 2007; Norm DIN EN 61158-1).

Klasse 2: Bussysteme dieser Klasse nutzen das Prioritätenschema auf dem MAC-Layer, wie es im Standard Institute of Electrical and Electronics Engineers (IEEE) 802.D/Q beschrieben ist

(Norm IEEE 802.1Q; Norm IEEE 802.1D). Zur weiteren Optimierung werden keine Transportschichten und Netzwerkschichten bei der Kommunikation verwendet. Es können aber weiterhin TCP und UDP mit IP-Kommunikation neben der Buskommunikation verwendet werden. Durch die Priorisierung der Buskommunikation hält sich der Einfluss der TCP/IP-Kommunikation in Grenzen. Es können Paketumlaufzeiten von 10 ms erreicht werden. PROFINET RT ist ein Beispiel für ein Bussystem dieser Klasse (Jasperneite et al. 2007; Norm DIN EN 61158-1).

Klasse 3: Anwendungsfälle wie z. B. die Bewegungssteuerung stellen hohe Anforderungen an die Echtzeitfähigkeit. Um diese noch weiter zu optimieren, verwendet diese Klasse ein Zeitschlitzverfahren. Ein solches Zeitschlitzverfahren setzt eine hohe Synchronisation der Uhren voraus. Um die Uhren zu synchronisieren, werden weitere nicht-Ethernet-konforme Mechanismen verwendet. Oft wird spezielle Hardware und Software für diese Anpassung benötigt. PROFINET IRT und Sercos III sind Beispiele für Bussysteme dieser Klasse (Jasperneite et al. 2007; Norm DIN EN 61158-1).

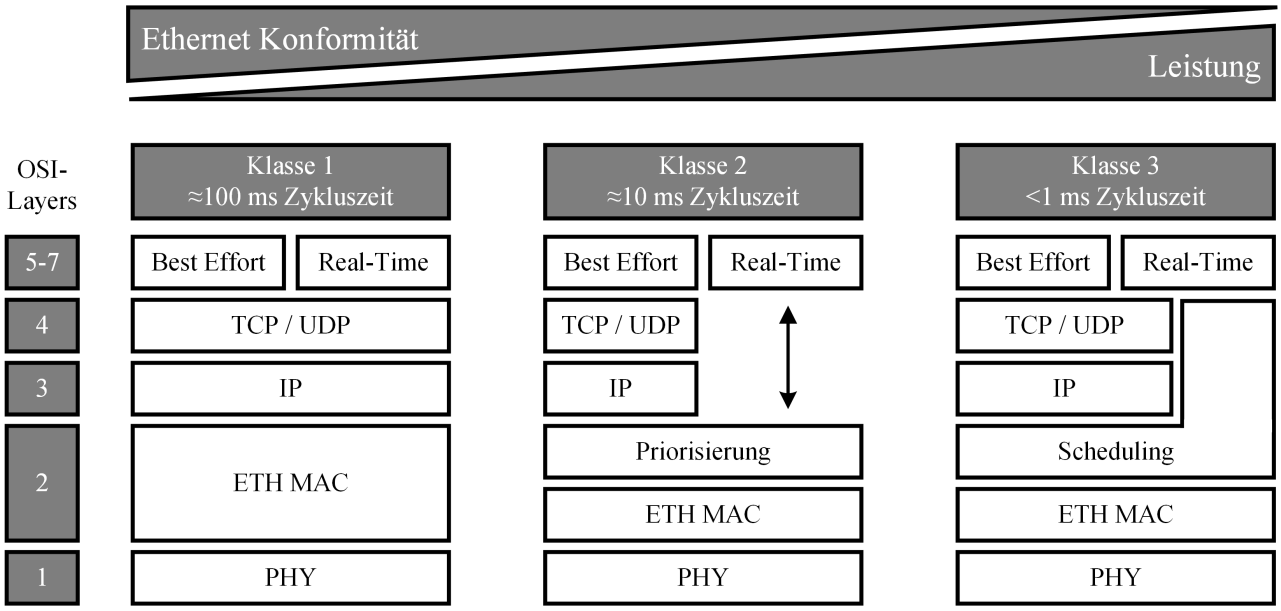


Abbildung 1.3: Klassifizierung von Industrial-Ethernet-Protokollen (Jasperneite et al. 2007).

Das Problem bisheriger Bussysteme der Klasse 3 ist, dass diese durch die Echtzeitanpassungen nicht mehr konform zum Ethernet-Standard sind (Wollschlaeger et al. 2017). Durch die Erweiterung von Ethernet um die Time-Sensitive Networking (TSN)-Standards, integriert die IEEE die nötigen Echtzeitmechanismen in Ethernet (Wollschlaeger et al. 2017). Um einen durchgängigen herstellerunabhängigen Standard zu definieren, wird Open Platform Communications Unified Architecture (OPC UA) als Interoperabilitätsstandard oberhalb der Schicht 5 verwendet. Das daraus resultierende Open Systems Interconnection (OSI)-Modell ist in Abbil-

dung 1.4 dargestellt. Durch das OSI-Referenzmodell existieren prinzipiell drei Pfade, wie es bei OPC UA in Kombination mit TSN vorgesehen ist.

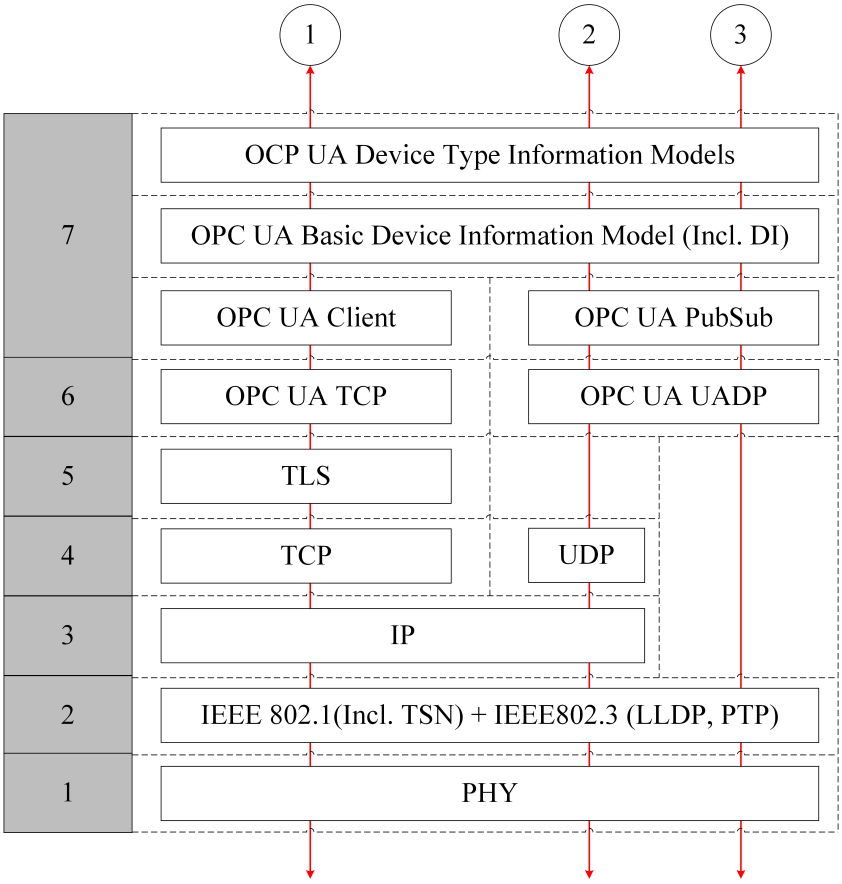


Abbildung 1.4: OSI-Referenzmodell von OPC UA und TSN (Bruckner et al. 2019).

Der erste Pfad verwendet TCP/IP und basiert auf einer Client/Server-Architektur. Dieser Pfad wird vor allem zur Konfiguration der Geräte verwendet. Der Datenverkehr ist in der Regel Best-Effort-Datenverkehr. Der zweite Pfad verwendet dabei UDP/IP in Kombination mit den OPC UA Publish Subscribe (OPC UA PubSub)-Mechanismen. Die Payload der Nachricht wird nach dem Message Mapping der Schicht 6 strukturiert. Auf Schicht 7 wird eine OPC UA PubSub-Architektur verwendet. Die Publisher und Subscriber enkodieren und dekodieren dabei die Applikationsdaten und versenden bzw. empfangen die Nachrichten. Der dritte Pfad unterscheidet sich vom zweiten Pfad dadurch, dass er keine Mechanismen der Schicht 3 bis 5 verwendet.

Alle Pfade nutzen zwar IEEE 802.1 inklusive TSN, aber dadurch, dass TSN eine Sammlung von Standards ist, besteht hier die Gefahr, dass unterschiedliche Sets von TSN-Standards zur Lösung von Anwendungsfällen definiert werden. Beispielsweise kann Hersteller A für den zyklischen Datenverkehr die Priorisierung ohne Zeitschlitzverfahren nutzen. Hersteller B nutzt das

Zeitschlitzverfahren. Diese beiden Lösungen wären dann, trotz TSN, nicht kompatibel zueinander.

1.1 Problemstellung

Durch die zunehmende Digitalisierung und der damit einhergehenden Vernetzung von Maschinen in Automatisierungsanlagen steigen die Anforderungen an die Interoperabilität (Bauernhansl et al. 2014). Die Querkommunikation zwischen Maschinen ist meistens mit hohem Aufwand verbunden, da Maschinen unterschiedliche Bustechnologien verwenden, die nicht zueinander kompatibel sind (Jasperneite et al. 2007). Um die hohen Anforderungen an die Interoperabilität zu erfüllen, wurde die OPC Field Level Communication (OPC FLC) Initiative gegründet. Sie soll die nächste Generation der Buskommunikation definieren. Dazu wird ein einheitliches und herstellerunabhängiges Busprotokoll und die dazugehörigen Informationsmodelle definiert, die den Anforderungen der Feldebene gerecht werden. In der Geschichte der Feldebene war man der Herstellerunabhängigkeit noch nie so nahe. Deshalb wird in dieser Arbeit die von der OPC FLC definierte industrielle Echtzeit-Kommunikationsarchitektur verwendet. OPC FLC soll auf den Standards OPC UA und TSN basieren. Unter TSN versteht man eine Sammlung mehrerer Standards, die in der IEEE Organisation standardisiert werden. TSN erweitert Ethernet unter anderem um notwendige Echtzeitmechanismen für die Feldebene. OPC UA bietet neben der bisherigen Client/Server-Architektur auch eine OPC UA PubSub-Architektur, die auf Grund der entkoppelten Kommunikation auch für leistungsschwächere Geräte geeigneter ist. Der erste Release Candidate von OPC UA PubSub wurde im Februar 2018 veröffentlicht und wird in weiteren Iterationen erweitert.

Dadurch, dass OPC UA PubSub auf einer entkoppelten Kommunikation basiert, sind alle an der Kommunikation beteiligten Entitäten einander unbekannt (Norm OPC UA Part 14). Demnach existiert keine direkte Verbindungen zwischen den Entitäten, wie es z. B. bei TCP mit dessen 3-Way-Handshake der Fall ist. Wenn z. B. der Subscriber aus gewissen Gründen herunterfährt, ist dies dem Publisher nicht bewusst. Wenn der Anwendungsfall eine Überwachung des Datenaustauschs erforderlich macht, muss die Überwachung auf Applikationsebene umgesetzt werden. In Abbildung 1.5 ist das Kommunikationsmodell des OPC UA PubSub-Standards dargestellt. Bisher beschreibt der Standard nur einen Mechanismus auf Seiten des Subscribers zur Überwachung einer unidirektionalen Verbindung zwischen Publisher und Subscriber. Der Subscriber überwacht die zyklisch empfangenen Nachrichten, indem er ein maximales Zeitfenster zwischen zwei empfangenen Nachrichten definiert. Beim Überschreiten des Zeitfensters reagiert der Subscriber mit einem Zustandswechsel auf den Zustand *Error*. Da der Informationsfluss

jedoch immer nur von Publisher zu Subscriber verläuft und kein Rückkanal definiert ist, ist dem Publisher der Zustand des Subscribers unbekannt.

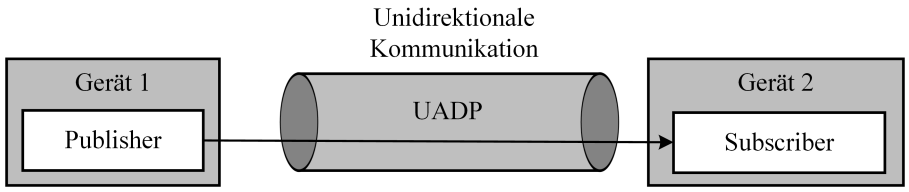


Abbildung 1.5: Unidirektionales OPC UA PubSub-Kommunikationsmodell.

Um standardkonform eine bidirektionale Verbindung zwischen zwei Geräten aufzubauen, ist es möglich, ein weiteres Paar, bestehend aus Publisher und Subscriber, zu instanziieren. In Abbildung 1.6 ist eine solche nicht-überwachte bidirektionale Verbindung dargestellt.

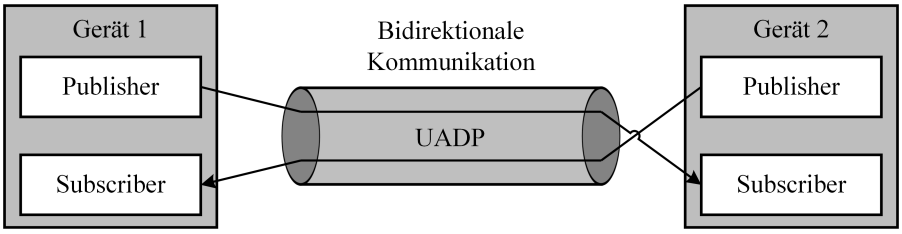


Abbildung 1.6: Bidirektionales OPC UA PubSub-Kommunikationsmodell.

Trotz des Rückkanals ist jedoch noch keine Überwachung der Verbindung gewährleistet, da im OPC UA PubSub-Standard diese Funktionalität noch nicht vorhanden ist. Damit an dieser Stelle keine Vielzahl an Überwachungslösungen auf Applikationsebene entstehen, muss eine Lösung im OPC UA PubSub-Standard definiert werden. Um dieses Problem zu lösen, wird in dieser Arbeit ein Systemdesign definiert, das den Datenaustausch über OPC UA PubSub überwachen kann. Dadurch ist es der Applikation möglich, flexibel auf Ausfälle in der Kommunikation zu reagieren. Dieser überwachte Datenaustausch ist in Abbildung 1.7 dargestellt.

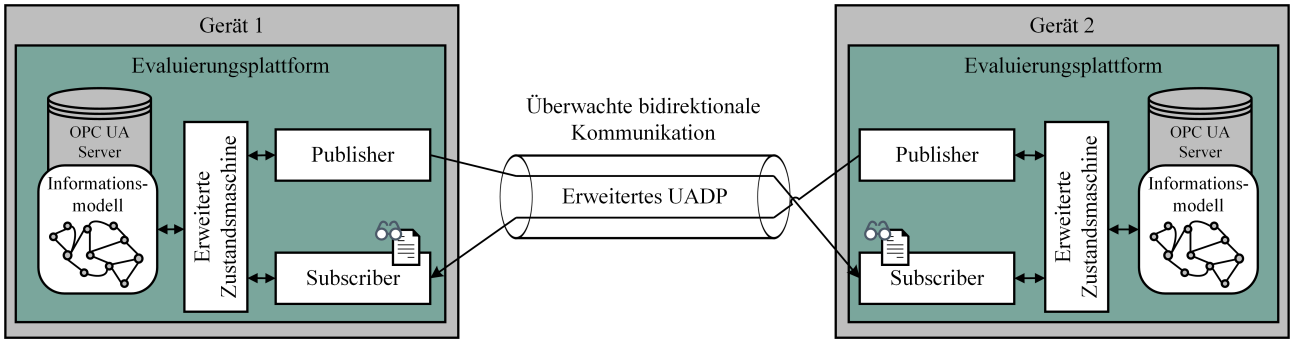


Abbildung 1.7: Anwendungsfall einer überwachten OPC UA PubSub-Verbindung.

1.2 Struktur der Arbeit

Die Arbeit besteht aus insgesamt acht Kapiteln. In Kapitel 1 wird in die Thematik eingeleitet und die Problemstellung dargestellt. In Kapitel 2 werden die zu erfüllenden Anforderungen beschrieben. Es wird zwischen allgemeinen Anforderungen und überwachungsspezifischen Anforderungen unterschieden. In Kapitel 3 werden die Grundlagen zum Verständnis der Arbeit sowie der Stand der Technik aufgezeigt. Des Weiteren wird die Zielsetzung dieser Arbeit beschrieben. In Kapitel 4 wird eine Evaluierungsplattform für den Datenaustausch basierend auf OPC UA PubSub und TSN definiert. Die Evaluierungsplattform dient als Basis für die folgende experimentelle Messung in Kapitel 5. Anhand der experimentellen Messung wird eine Leistungsmetrik für OPC UA PubSub und TSN erstellt. In Kapitel 6 werden zunächst der Datenaustausch der Klasse 3 Busprotokolle PROFINET IRT und Sercos III abstrahiert, um darauf einen überwachten Datenaustausch basierend auf OPC UA PubSub abzuleiten. In Kapitel 7 wird die Implementierung über eine Konkretisierung, eine Simulation und ein Testaufbau validiert. Im letzten Kapitel 8 werden die Ergebnisse zusammengefasst und ein Ausblick gegeben.

2 Anforderungen an die Kommunikation

In diesem Kapitel werden die zu erfüllenden Anforderungen beschrieben. Dazu wird zunächst zwischen vier Kommunikationsanwendungsfällen unterschieden. Diese Anwendungsfälle besitzen unterschiedliche Anforderungen an die Kommunikation, die als Rahmenbedingungen für die Überwachung anzusehen sind. Das heißt, die Überwachung muss innerhalb der Kommunikationsanforderungen funktionieren. Des Weiteren werden überwachungsspezifische Anforderungen definiert.

2.1 Anwendungsfälle und allgemeine Anforderungen

Kommunikationsanwendungsfälle aus der Sicht einer Steuerung lassen sich in vier Kategorien unterteilen – Steuerung-zu-Enterprise Level (C2E), Steuerung-zu-HMI (C2H), Steuerung-zu-Steuerung (C2C) und Steuerung-zu-Feldgerät (C2D). Es existieren noch weitere Anwendungsfälle z. B. Sensor-zu-Cloud, jedoch beschränkt sich die Arbeit auf die zuvor genannten vier Anwendungsfälle. In Abbildung 2.1 sind die vier Anwendungsfälle dargestellt. Es wird zwischen horizontaler und vertikaler Kommunikation unterschieden. Die vertikale Kommunikation beschreibt die Kommunikation über Ebenen hinweg. Beispiele dafür sind die C2E-Kommunikation und die C2D-Kommunikation. Die horizontale Kommunikation beschreibt die Kommunikation innerhalb einer Ebene. Die Anwendungsfälle C2H und C2C zählen zu dieser Kategorie. In dieser Arbeit wird der Fokus vor allem auf die Anwendungsfälle C2C und C2D gesetzt, da sie eine zyklische Kommunikation verwenden.

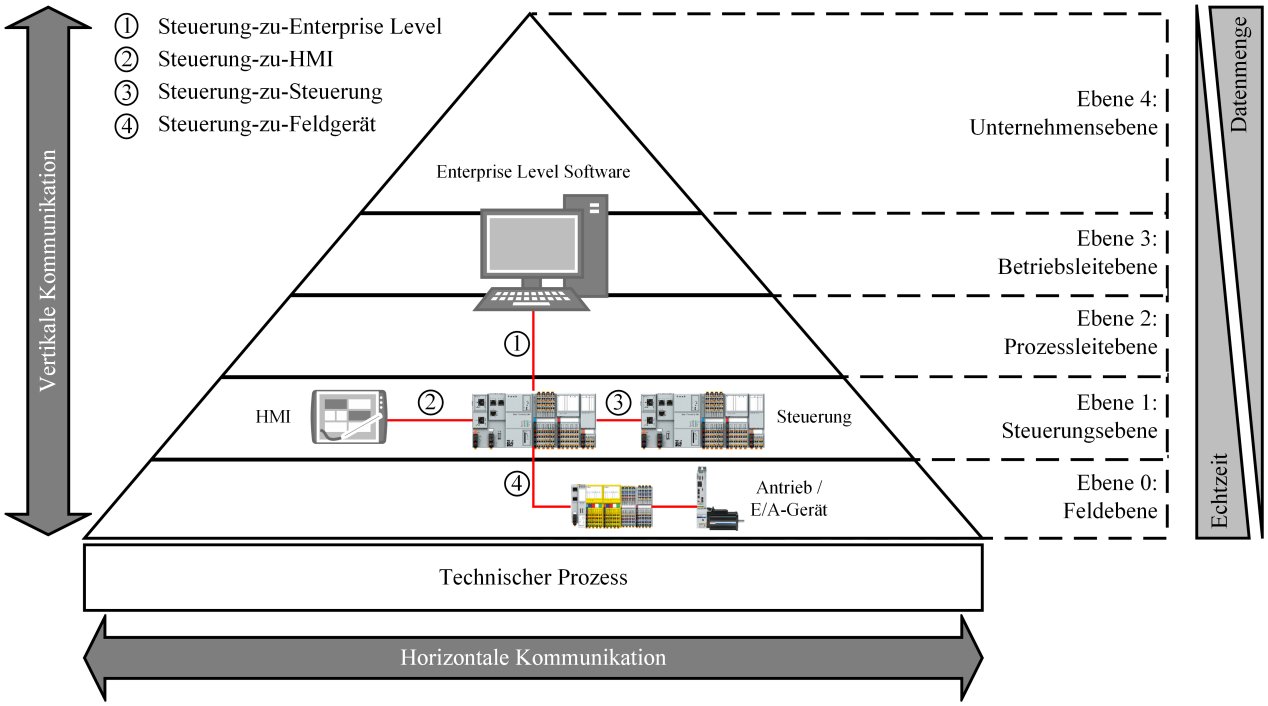


Abbildung 2.1: Kommunikationsanwendungsfälle einer Steuerung (Eckhardt et al. 2018).

Durch die vier Anwendungsfälle entstehen unterschiedliche Datenströme mit jeweils spezifischen Eigenschaften. Neben den Datenströmen aus der Sicht einer Steuerung gibt es noch weitere Datenströme in industriellen Netzwerken. Die IEEE Organisation hat dazu im IEEE 802.1Q Standard insgesamt acht Datenströme zur Kategorisierung industrieller Netzwerke definiert.

Um die Kategorie *Critical Applications* noch weiter zu präzisieren, definiert das Industrial Internet Consortium (IIC) drei weitere Unterkategorien. Das IIC definiert zur genauen Beschreibung der Datenströme acht Eigenschaften (IIC Traffic Types 2018).

In Tabelle 2.1 werden die Eigenschaften der Datenströme kurz beschrieben. In Tabelle 2.2 sind die Datenströme aufgelistet. In Tabelle 2.3 sind die Anwendungsfälle und deren Eigenschaften aufgelistet.

Tabelle 2.1: Eigenschaften zur Kategorisierung von Datenströmen (IIC Traffic Types 2018).

Eigenschaft	**Beschreibung**
Periodizität	Es wird zwischen zyklischer und azyklischer Übertragung unterschieden.
Periodendauer	Die Periodendauer beschreibt die Dauer eines Zyklus.
Synchronisation	Es wird zwischen Applikationen unterschieden, die zum Netzwerk synchronisiert sind oder nicht.
Garantie	Es wird zwischen drei Garantien unterschieden. ▪ Die Deadline-Garantie sagt aus, dass die Übertragung der Pakete spätestens bis zu einem bestimmten Zeitpunkt abgeschlossen ist. ▪ Die Latenz-Garantie garantiert eine Übertragung an alle Empfänger in einer bestimmten Zeitspanne. ▪ Die Bandbreiten-Garantie sagt aus, dass die Daten an allen Empfängern ankommen, sofern der Sender die reservierte Bandbreite nicht überschreitet.
Jitter	Beschreibt die Varianz der Laufzeit von Datenpaketen.
Paketverlust	Die Toleranz gegen Paketverlust beschreibt, wie viele Pakete hintereinander ausfallen dürfen, bevor die Applikation in einen Fehlerzustand wechselt.
Paketgröße	Beschreibt die typische Größe der Pakete in Bytes.
Kritikalität	Beschreibt, wie kritisch die Pakete sind. Drei Kategorien stehen zur Auswahl: ▪ **Hoch:** Beschreibt die Übertragung von Daten innerhalb der Anforderungen an die Quality of Services (QoS). Ansonsten droht eine Fehlfunktion beim System. Eine erneute Übertragung der Daten ist aus zeitlichen Gründen nicht möglich. ▪ **Mittel:** Beschreibt die Übertragung von relevanten Daten, die aber im Fall einer Störung zu keinem Fehlverhalten des Systems führen. Verloren gegangene Pakete werden erneut übertragen. ▪ **Niedrig:** Beschreibt die Übertragung von Daten, die nicht essentiell für die Operation des Systems sind. Im Fall eines verloren gegangenen Pakets kann das Paket erneut übertragen werden.

Tabelle 2.2: Verschiedene Datenströme nach IEEE 802.1Q und dem IIC.

Name		**Beschreibung**
Voice		Entsteht beim Streamen von Audiodaten.
Video		Entsteht beim Streamen von Videodaten.
Network Control		Enthält die Nachrichten für die netzwerkinterne Steuerung.
Best Effort		Standard-Kommunikation. Die Nachrichten sind nicht priorisiert.
Excellent Effort		Auch CEO's Best Effort genannt.
Background Traffic		Großteil des Datenverkehrs oder andere Netzwerkaktivitäten.
Internetwork Control		Enthält die Nachrichten für die netzwerkübergreifende Steuerung.
Critical Applications		
	Isochronous	Die Applikationen sind untereinander synchronisiert.
	Cyclic	Die Kommunikation verläuft zyklisch, ohne Synchronisation der Applikationen.
	Alarms and Events	Es werden kritische Alarme und Events erzeugt.

Tabelle 2.3: Eigenschaften der allgemeinen Kommunikationsanwendungsfälle.

Eigenschaft	**C2E**	**C2H**	**C2C**	**C2D**
Periodizität	Azyklisch	Azyklisch/ Zyklisch	Zyklisch	Zyklisch
Periodendauer	-	-	2 - 20 ms	100 µs - 2 ms
Synchronisation	Nein	Nein	Nein	Ja
Garantie	Keine	Latenz	Latenz	Deadline
Jitter	-	App.-abhängig	Ja	0
Paketverlust	Ja	Ja	1 - 4	0 - 1
Paketgröße	Variabel	Variabel	Statisch	Statisch
Kritikalität	Gering	Hoch	Hoch	Hoch

Steuerung-zu-Enterprise Level: Bei der Kommunikation zwischen einer Steuerung und Enterprise Systemen handelt es sich in der Regel um eine azyklische Kommunikation. Die azyklische Kommunikation basiert meistens auf TCP/IP-Verbindungen zwischen einem Client und einem Server. Die Mechanismen von TCP sorgen für eine zuverlässige Übertragung und Fehlererkennung. TCP erkennt verloren gegangene Pakete und führt darauf ein erneutes Übertragen der verlorenen Pakete aus. Die Menge an Daten kann im Vergleich zu den anderen

Anwendungsfällen sehr groß sein, da z. B. ganze Applikationsprogramme übertragen werden. Die Datenströme haben keine nennenswerten Anforderungen an Jitter und die Kritikalität ist gering.

Steuerung-zu-HMI: Der Hauptanwendungsfall bei der Kommunikation zwischen einer Steuerung und einem Human Machine Interface (HMI) besteht in der Anzeige des Zustands der Steuerung. Sie basiert auf einer zyklischen und azyklischen Kommunikation und fordert von der Netzwerk-Infrastruktur eine anwendungsfallabhängige Latenz-Garantie im Bereich von 100 ms bis 1 s. Die Client/Server-Architektur von OPC UA hat sich für diesen Anwendungsfall als der De-facto-Standard herauskristallisiert (Candido et al. 2010; Schwarz et al. 2013). Dabei variiert üblicherweise die Paketgröße zwischen 50 und 1500 Bytes. Bei der Überwachung von Alarmen ist die Kritikalität hoch.

Steuerung-zu-Steuerung: Bei der Querkommunikation zwischen Steuerungen handelt es sich um eine zyklische Datenübertragung mit einer üblichen Periodendauer zwischen 2 bis 20 ms. Die Applikation ist in den meisten Fällen nicht mit dem Netzwerk synchronisiert. Die Netzwerk-Infrastruktur garantiert eine anwendungsfallabhängige Latenzgarantie. Außerdem kann die Applikation einen gewissen Jitter tolerieren und ist robust gegen Paketausfall. Übliche Paketgrößen sind hier 50 bis 1000 Bytes mit einer hohen Kritikalität. Im Fall einer Motion-Applikation können sich die Anforderungen jedoch erheblich verschärfen.

Steuerung-zu-Feldgerät: Steuerungen übertragen zyklisch Daten zu einem oder mehreren Feldgeräten. Typische Periodenwerte sind 100 µs bis 2 ms bei einer Paketgröße von 30 bis 100 Bytes. Um diese Performance zu erreichen, muss die Applikation mit dem Netzwerk synchronisiert sein, damit z. B. ein Mechanismus wie das Zeitschlitzverfahren von TSN verwendet werden kann. Die TSN-Mechanismen garantieren, dass die Datenpakete bis spätestens zu einer Deadline übertragen werden und dabei ein sehr kleiner Jitter auftritt. Die Toleranz gegen Paketverlust ist hier äußerst gering. Bei Sercos III wird standardmäßig ein Paketverlust von maximal einem Paket toleriert. Fällt ein weiteres Paket hintereinander aus, würde die Applikation in einen Fehlerzustand gehen und einen Fehler in der Kommunikation anzeigen. Die Kritikalität des Datenstroms ist sehr hoch.

2.2 Spezifische Anforderungen an eine Überwachung

Die spezifischen Anforderungen an eine Überwachung setzen sich aus protokoll- und anwendungsspezifischen Anforderungen zusammen. In Abbildung 2.2 sind die Anforderungen und ihre Kategorisierung dargestellt und werden nachfolgend kurz beschrieben.

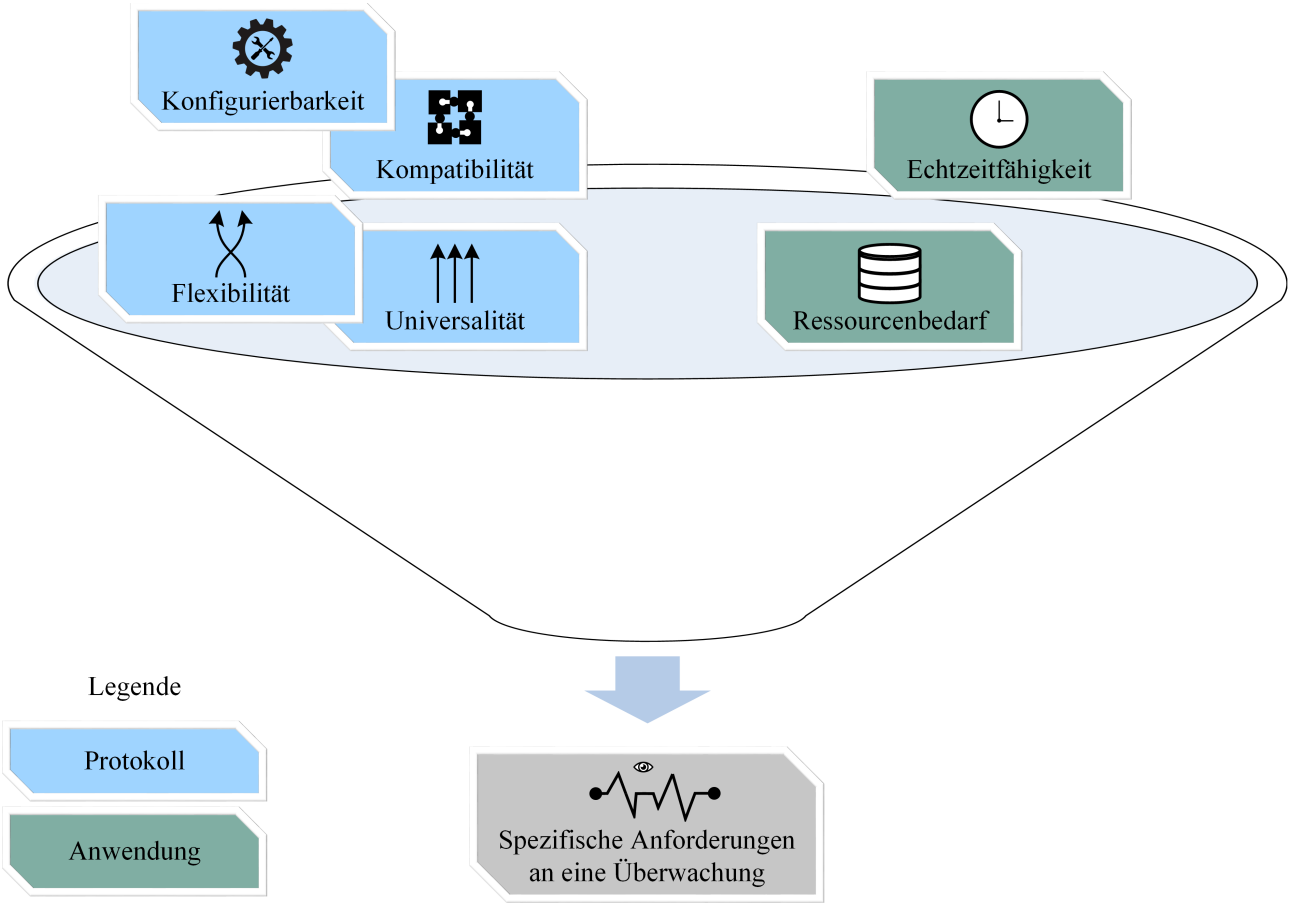


Abbildung 2.2: Herleitung der spezifischen Anforderungen an die Überwachung.

Kompatibilität: Die Erweiterungen am OPC UA PubSub-Protokoll müssen kompatibel zum bisherigen OPC UA PubSub-Protokoll sein und gering gehalten werden. Neue Typen des Informationsmodells, die zur Konfiguration eines überwachten Datenaustauschs nötig sind, müssen kompatibel zu den Typen des bisherigen Informationsmodells sein. Die Zustandsmaschine zur Überwachung des OPC UA PubSub-Datenaustauschs muss kompatibel zu den bisherigen Zustandsmaschinen sein.

Flexibilität: Der parallele Einsatz eines überwachten Datenaustauschs zum standardmäßigen Datenaustausch im OPC UA PubSub-Protokoll ist eine Grundanforderung. Dadurch kön-

nen Geräte, die keinen überwachten Datenaustausch unterstützen, trotzdem parallel eingesetzt werden. Es müssen die Kardinalitäten 1-zu-1 und 1-zu-N unterstützt werden. Der Datenaustausch muss uni- und bidirektional funktionieren. OPC UA PubSub ist sehr flexibel und soll durch die Überwachung nicht eingeschränkt werden. Als Beispiel dienen, die in Abbildung 2.3 dargestellten, grundlegende Anwendungsfälle eines überwachten Datenaustauschs. Der erste Anwendungsfall beschreibt den Austausch von Prozessdaten und den Status des Geräts A über den Publisher A an den Subscriber B. Der Publisher B überträgt lediglich den Status vom Subscriber B an den Subscriber A. Der zweite Anwendungsfall unterscheidet sich vom ersten Anwendungsfall insofern, dass der Publisher B den Rückkanal außerdem für die Übertragung von Prozessdaten verwendet. Der dritte Anwendungsfall zeigt die Überwachung einer 1-zu-N-Verbindung (N=2). Außerdem zeigt er die Kombination des ersten und zweiten Anwendungsfalls. Die Geräte A und B tauschen bidirektional Prozessdaten und Status aus. Die Geräte A und C tauschen hingegen nur zwischen dem Publisher A und Subscriber C Prozessdaten aus.

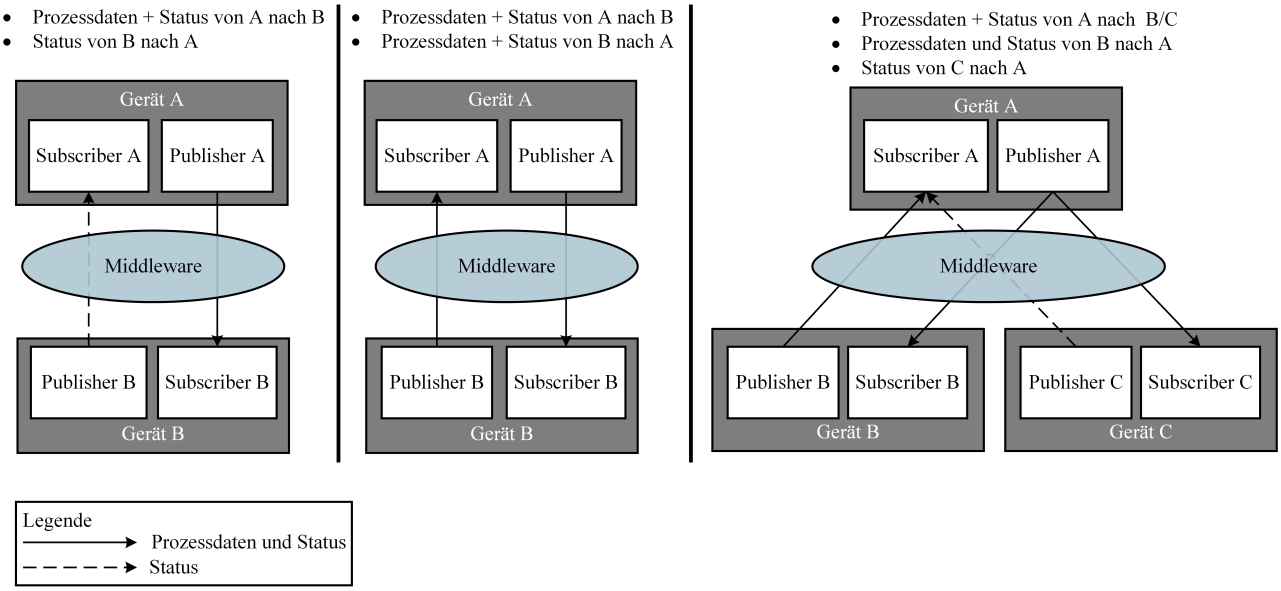


Abbildung 2.3: Drei abstrahierte Anwendungsfälle.

Universalität: Die Universalität zeichnet sich durch die Unabhängigkeit zu den Transportprotokollen aus. Der überwachte Datenaustausch soll auf dem Message Mapping UA Datagram Protocol (UADP) basieren und wahlweise über die Transportprotokollen OPC UA-Ethernet und OPC UA-UDP verwendet werden.

Echtzeitfähigkeit: Der überwachte Datenaustausch darf die Leistungsfähigkeit der Kommunikation zwischen Publisher und Subscriber nicht nennenswert beeinflussen. Das heißt, es soll

bereits Vorhandenes aus dem UADP-Protokoll verwendet werden, und nur wenn es unumgänglich ist, das UADP-Protokoll um neue Strukturen erweitern.

Konfigurierbarkeit: Der überwachte Datenaustausch muss auch über die Client/Server-Architektur konfigurierbar sein. Neue Typen des Informationsmodells sollen sich bei der Strukturierung an bisherige Typen orientieren. Die neuen Typen zur Konfiguration eines überwachten Datenaustauschs müssen kompatibel zum bisherigen Konfigurationsmodell sein. Die Vorgehensweise der Konfiguration eines überwachten Datenaustauschs soll sich mit der Vorgehensweise der Konfiguration eines nicht-überwachten Datenaustauschs decken.

Ressourcenbedarf: Der überwachte Datenaustausch darf keinen übermäßigen Ressourcenbedarf auf den Geräten erzeugen.

3 Stand der Technik und Zielsetzung

3.1 OSI-Schichtenmodell

Im OSI-Schichtenmodell wird die Kommunikation in sieben Schichten gegliedert. Dabei sind die Schichten 1 bis 4 transportorientierte Schichten und die Schichten 5 bis 7 anwendungsorientierte Schichten. Eine Schicht besteht immer aus der Service Data Unit (SDU) und der Protocol Data Unit (PDU). Die SDU von Schicht X ist die PDU von Schicht X+1. In Abbildung 3.1 ist dieses Prinzip dargestellt. Nachfolgend werden die Schichten kurz erklärt.

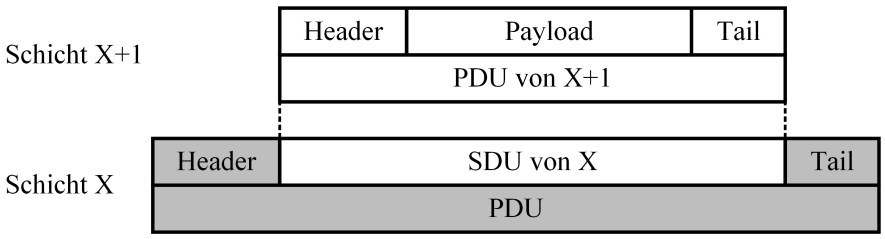


Abbildung 3.1: Verkapselung der PDU und der SDU.

Schicht 1 – Physical Layer: Die Übertragungsschicht ist die erste Schicht des OSI-Referenzmodells und definiert die elektrische, mechanische und funktionale Schnittstelle zum Medium. Sie stellt eine physikalische Verbindung zwischen Geräten her und überträgt die Bitströme. In dieser Schicht entspricht die PDU einem Bit.

Schicht 2 – Data Link Layer: Die Sicherungsschicht bietet sämtliche Funktionalitäten für eine fehlerfreie Übertragung zwischen den Geräten. Es ist ein verbindungsloser Modus als auch ein verbindungsbasierter Modus möglich. Die Schicht teilt Bitströme in Frames auf. Des Weiteren bietet die Schicht Dienste zum Aufbauen, Erhalten und Schließen von Netzwerkverbindungen an. Außerdem beinhaltet diese Schicht Funktionen zur Fehlererkennung, Fehlerbehebung und Datenflusskontrolle. In dieser Schicht entspricht die PDU einem Frame. Typische Protokollbeispiele sind IEEE 802.3 Ethernet (Norm IEEE 802.3) oder IEEE 802.11 WLAN (Norm IEEE 802.11)

Schicht 3 – Network: Die Vermittlungsschicht bietet sämtliche Funktionalitäten für die Weitervermittlung von Datenpaketen über gesamte Netzwerke hinweg an. Die wichtigsten Aufgaben dieser Schicht sind das Bereitstellen von netzwerkübergreifenden Adressen, das Routing und die Fragmentierung von Datenpaketen. Ein typisches Protokoll dieser Schicht ist das IP. In dieser Schicht entspricht die PDU einem Paket.

Schicht 4 – Transport: Die Transportschicht segmentiert die Datenströme und fügt Informationen hinzu, um sie zwischen Sitzungen zu übertragen. Sie dient als das Bindeglied zwischen den transportorientierten und anwendungsorientierten Schichten. Zu den typischen Transportprotokollen dieser Schicht zählen das TCP (Norm RFC 793) und das UDP (Norm RFC 768). In dieser Schicht entspricht die PDU einem Segment.

Schicht 5 – Session: Die Sitzungsschicht organisiert und verwaltet den Austausch von Daten. Dafür bietet die Sitzungsschicht Dienste an, die eine Sitzungsverbindung zwischen zwei Entitäten der Darstellungsschicht aufbauen.

Schicht 6 – Presentation: Die Darstellungsschicht liefert eine syntaktisch einheitliche Darstellung der Daten, die zwischen den Anwendungsinstanzen ausgetauscht werden. Mögliche Darstellungsarten sind z. B. die Anordnung von Bits oder die unterschiedlichen Kodierungen eines Zeilenendes. Eine weitere Funktion der Schicht ist die Kodierung der Daten.

Schicht 7 – Application: Die Anwendungsschicht ist die oberste Schicht und stellt den Zugriffspunkt auf die OSI-Umgebung für den Anwendungsprozess dar. Außerdem bildet die Schicht die Applikationsprofile der Bussysteme ab.

3.2 Kommunikationsprotokolle

Im Kapitel 1 sind bereits die drei möglichen Kommunikationspfade der Echtzeitklasse 3 beschrieben. In diesem Unterkapitel werden die verwendeten Protokolle IP, UDP und TCP zusammengefasst und miteinander verglichen.

Die IEEE 802.3 Ethernet-Kommunikation: Bei dieser Kommunikation handelt es sich um die ausschließliche Nutzung der Dienste der Schichten 1 und 2. Die Schicht verpackt die Daten in Ethernet-Frames und tauscht die Frames zwischen den Entitäten aus. Zur Adressierung fügt die Schicht nach der Preamble und dem Start Frame Delimiter (SFD) eine Ziel-MAC-Adresse, eine Source-MAC-Adresse und ein Ethertype ein. Im IEEE 802.1Q Standard sind außerdem weitere 4 Bytes vorgesehen, die sogenannte QoS definieren. Über diese QoS kann das Virtual Local Area Network (VLAN) und die Priorität der Frames identifiziert werden. Ein VLAN ist ein logisches Teilnetz innerhalb eines Netzwerks. Die Ziel-MAC-Adresse kann entweder aus Unicast-, Multicast- oder Broadcast-Adressen bestehen. Die Schicht kann fehlerhafte Frames über das Frame-Check-Sequence (FCS)-Feld identifizieren. Die meisten TSN-Protokolle basieren auf dieser Kommunikation.

Die UDP/IP-Kommunikation: Bei UDP/IP handelt es sich um eine entkoppelte Kommunikation basierend auf Datagrammen. Die Schicht verwendet IP-Adressen zur Adressierung. Die IP-Adresse ist entweder eine Unicast-, Multicast- oder Broadcast-Adresse. Da UDP auf einer entkoppelten Kommunikation basiert, bietet UDP keine Mechanismen für eine zuverlässige Kommunikation an. Ein Mechanismus für eine zuverlässige Kommunikation ist z. B. ein erneutes Senden eines Datagramms im Fall eines verloren gegangenen Datagramms. Wenn solch ein Mechanismus eine Anforderung der Applikation ist, muss dieser in höheren Schichten realisiert werden. Beim UDP können die Datagramme ungeordnet ankommen und werden nicht sortiert. Das UDP basiert auf dem IP der Schicht 3 und kann deswegen Routing verwenden, wodurch Datagramme netzwerkübergreifend versendet werden können.

Die TCP/IP-Kommunikation: TCP/IP beschreibt ein verbindungsorientiertes Protokoll basierend auf Segmenten. TCP nutzt IP-Adressen zur Adressierung. Da es sich um ein verbindungsorientiertes Protokoll handelt, sind hier nur Unicast-Adressen erlaubt. Die Header-Größe (20 bis 60 Bytes) ist abhängig von den Einstellungen. TCP garantiert eine verlässliche Verbindung im Sinne einer erneuten Übertragung der Segmente im Fall eines Datenverlusts. Das TCP sortiert empfangene Segmente in die richtige Reihenfolge ein. Außerdem nutzt das TCP das unterliegende IP, weshalb über die Routing-Mechanismen eine netzwerkübergreifende Kommunikation möglich ist.

Vergleich der Kommunikationsprotokolle: In der Tabelle 3.1 sind die drei Protokolle und ihre Eigenschaften gegenübergestellt. Jedes der Protokolle hat unterschiedliche Charakteristiken, die je nach Anwendungsfall zu Vor- oder zu Nachteilen werden können. Der Vergleich zeigt,

dass der Standard IEEE 802.3 Ethernet die Anforderungen aus Kapitel 2.1 aus folgenden Gründen am besten erfüllt. Aufgrund der kleineren Header-Größe ist das Protokoll leichtgewichtiger als UDP/IP und TCP/IP, wodurch es zu einer Leistungssteigerung kommt. Außerdem werden die Netzwerk-Stack-Übergänge zwischen den Schichten 2 bis 4 vermieden. Diese Übergänge sorgen in der Regel bei Standard-Betriebssystemen für nicht vorhersehbare Verzögerungen in der Kommunikation.

In der Feldebene werden meist ressourcenarme Geräte verwendet, die ressourcenbedingt nur eine begrenzte Anzahl von Verbindungen aufbauen können. Das Verwalten einer Verbindung verbraucht zu viele Ressourcen, weshalb eine verbindungsorientierte Kommunikation für diese Geräte nicht ideal ist. Darüber hinaus können die Vorteile einer derartigen Kommunikation oftmals wegen den kurzen Zykluszeiten bei einer zyklischen Feldbuskommunikation nicht genutzt werden. Beispielsweise ist der Retransmit-Mechanismus von TCP zu langsam, um Daten erneut in einem Zyklus zu übertragen. TSN erweitert den IEEE 802.3 Ethernet Standard um Echtzeitmechanismen. Diese Echtzeitmechanismen sind die Grundlage für die Buskommunikation und deshalb auch das wichtigste Kriterium für die Auswahl des IEEE 802.3 Ethernet Standards als Kommunikationsprotokoll.

Tabelle 3.1: Vergleich von IEEE 802.3 Ethernet, UDP/IP und TCP/IP (Norm IEEE 802.3; Norm RFC 768; Norm RFC 793; Norm RFC 8304).

Eigenschaften	**802.3 Ethernet**	**IP/UDP**	**IP/TCP**
Header Größe [Bytes]	14 - 18	IP:20 UDP: 8	IP: 20 TCP: 20 - 60
PDU	Frame	Datagramm	Segment
Verbindungsorientiert	Nein	Nein	Ja
Verlässliche Verbindung	Nein	Nein	Ja
Übertragung	Ungeordnet	Ungeordnet	Geordnet
Flusssteuerung	Nein	Nein	Ja
Staukontrolle	Nein	Nein	Ja
Skalierbarkeit	Ja	Ja	Nein
Unicast	Ja	Ja	Ja
Multi-/Broadcast	Ja	Ja	Nein
Routing	Nein	Ja	Ja

3.3 Time-Sensitive Networking

TSN ist eine Sammlung aus mehreren Standards, die von der Arbeitsgruppe *TSN Task Group* (IEEE TSN Task Group 18. 03. 2020) vorangetrieben werden. Die Arbeitsgruppe ist die Nachfolgerin der Arbeitsgruppe *Audio Video Bridging Task Group* (IEEE AVB Task Group 18. 03. 2020). Die Erweiterungen von TSN bieten eine Übertragung von Daten mit geringer Latenz und hoher Verfügbarkeit an. Außerdem ermöglicht TSN konvergente Netzwerke und fördert damit die Verschmelzung von IT und OT. Vorteile von TSN sind unter anderem:

- Robuste und zuverlässige Übertragung von Daten
- Garantierte Latenz
- Konvergente Netzwerke
- Vertikale und horizontale Integration
- Skalierbarkeit. TSN skaliert mit Ethernet, z. B. 100 Mbps bis 10 Gbps
- Gateway-Ersatz zwischen den OSI-Schichten

In Tabelle 3.2 sind einige der IEEE TSN-Standards aufgelistet. Am relevantesten für diese Arbeit sind die Standards IEEE 802.1AS-2011: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks und IEEE 802.1Q-2018 Amendment 25: Enhancements for Scheduled Traffic. Die beiden Standards sind in den nächsten zwei Unterkapiteln kurz zusammengefasst.

Tabelle 3.2: Übersicht über die IEEE TSN-Standards.

Name	**Beschreibung**
802.1AS-2011 (Norm IEEE 802.1AS)	Timing and Synchronization for Time-Sensitive Applications
802.1Q-2018 (Norm IEEE 802.1Q)	Bridges and Bridged Networks
802.1Qbv-2015 (Norm IEEE 802.1Qbv)	Amendment 25: Enhancements for Scheduled Traffic
802.1Qbu-2016 (Norm IEEE 802.1Qbu)	Amendment 26: Frame Preemption
802.1AB-2016 (Norm IEEE 802.1AB)	Station and Media Access Control Connectivity Discovery
802.1AX-2014 (Norm IEEE 802.1AX)	Link Aggregation
802.1BA-2011 (Norm IEEE 802.1BA)	Audio Video Bridging Systems
802.1CB-2017 (Norm IEEE 802.1CB)	Frame Replication and Elimination for Reliability
802.1CM-2018 (Norm IEEE 802.1CM)	Time-Sensitive Networking for Fronthaul

3.3.1 IEEE 802.1AS

Der Standard definiert das generalized Precision Time Protocol (gPTP) um zeitbewusste Systeme zu synchronisieren, damit die Anforderungen der zeitkritischen Applikationen erfüllt werden können. Es gibt zwei unterschiedliche zeitbewusste Systeme:

- Zeitbewusste Endstation
- Zeitbewusste Bridge

Beide Systeme können als Grandmaster verwendet werden. Ein Grandmaster bestimmt die Referenzzeit. Der Unterschied zwischen einer Endstation und einer Bridge ist, dass eine Bridge noch Korrekturen an den zeitlichen Informationen tätigt, die zwischen den zeitbewussten Systemen ausgetauscht werden. Dadurch können Verzögerungen im Local Area Network (LAN) oder in der Bridge selber ausgeglichen werden.

Der Standard sieht Methoden zur Verzögerungsmessung für die folgende Netzwerk-Technologien vor:

- IEEE 802.3 Ethernet mit full-duplex point-to-point Verbindungen
- IEEE 802.3 Ethernet Passive Optical Network-Verbindungen
- IEEE 802.11 Wireless-Verbindungen
- Coordinated Shared Network (CSN)

Für diese Arbeit ist ausschließlich der erste Punkt mit einer Ethernet full-duplex point-to-point Verbindung relevant. Implementierungen erreichen bereits eine Synchronisationsgenauigkeit von deutlich unter 1 µs (Lo Bello et al. 2019).

3.3.2 IEEE 802.1Q

Ohne TSN verfügt Standard-Ethernet über kein Zeitmultiplexverfahren. Um die Anforderungen von zeitsensiblen Applikationen zu erfüllen, wurde die Erweiterung 802.1Qbv – Enhancements for Scheduled Traffic, die bereits Bestandteil von 802.1Q ist, definiert. Typische Applikationen mit solchen Anforderungen sind in der Automatisierungstechnik oder der Automobilindustrie vorzufinden. IEEE 802.1Qbv erweitert den Ethernet-Standard um ein Zeitmultiplexverfahren. Zeitkritische Frames enthalten z. B. prozessrelevante Daten, die für den nächsten Steuerungszyklus benötigt werden. Üblicherweise handelt es sich dabei um zyklischen Datenverkehr. Eine verspätete Übertragung solcher Frames kann zu Ungenauigkeit oder sogar zu Fehlfunktion der Maschine führen. Bisher werden bewusst separate Netzwerke für diesen hoch kritischen Datenverkehr zur Verfügung gestellt. Die steigende Anforderung nach konvergenten Netzwerken fördert die Entwicklung von Mechanismen wie IEEE 802.1Qbv. Die alleinige Priorisierung des Datenverkehrs über Traffic Classes reicht für diese Art von Applikationen nicht aus, da im Worst-Case zu dem Zeitpunkt der geplanten Übertragung bereits ein Frame mit niedriger Priorität gerade das Übertragen begonnen hat. Dies verursacht eine Verzögerung, die der Verarbeitung eines Frames mit maximaler Größe entspricht. Diese Art der Verzögerung könnte theoretisch in jedem Knoten im Netzwerk auftreten und dadurch eine inakzeptable Latenz verursachen (Jiang et al. 2018). Die Evaluierung der Leistung von TSN hinsichtlich der Latenz, dem Jitter oder dem Paketverlust ist schon länger Gegenstand der Forschung (Lo Bello et al. 2019; Jiang et al. 2018; Nasrallah et al. 2019; Zhao et al. 2018). Nachfolgend werden die Auswahlmechanismen von IEEE 802.1Q zur Wahl des nächsten zu übertragenden Frame kurz vorgestellt.

Als Nächstes ist der prinzipielle Ablauf einer Übertragung beschrieben. Jeder Frame gehört zu einer Traffic Class, die wiederum einer Traffic Queue zugewiesen ist. Es können auch mehrere Traffic Classes einer einzigen Traffic Queue zugewiesen werden. Über einen Transmission Selection Algorithm (TSA) werden die zu übertragenden Frames ausgewählt. Im Standard 802.1Q werden vier TSAs vorgestellt:

Strict Priority: Dieser Algorithmus wählt die Frames basierend auf der Priorisierung der Queues aus. Es wird ein Frame einer Queue ausgewählt, wenn kein Frame in einer Queue mit höherer Priorität vorliegt. Ein Nachteil dieses Algorithmus ist, dass prioritäre Queues solche mit niedriger Priorität verdrängen können.

Credit-based Shaper: Der Credit-based-Shaper-Algorithmus ist der komplizierteste Algorithmus und für die hier vorliegende Arbeit nicht relevant. Vollständigkeitshalber ist jedoch das Prinzip nachfolgend kurz erklärt. Der Credit-based-Shaper-Algorithmus weist jeder Queue einen Credit zu. Wenn der Shaper Frames aus der Queue zur Übertragung auswählt, baut der Shaper gleichzeitig den Credit ab. Überträgt der Shaper Frames einer anderen Queue, so füllt sich der Credit wieder auf. Im Vergleich zum Strict Priority-Algorithmus, verteilt der Credit-based Shaper die Bandbreite fairer zwischen den Queues. Dem Verdrängen einzelner Queues wird dadurch vorgebeugt.

Enhanced Transmission Selection: Diese Erweiterung nutzt das Zeitmultiplexverfahren, um eine gewisse Bandbreite für die Traffic Classes zu reservieren. Das heißt, zu einem definierten Zeitpunkt wird die komplette Bandbreite einer oder mehrerer Traffic Classes zugewiesen. Dadurch können zeitkritische und zeitunkritische Applikationen koexistieren, ohne sich gegenseitig zu beeinflussen. In Abbildung 3.2 ist das Prinzip des Zeitmultiplexverfahrens dargestellt.

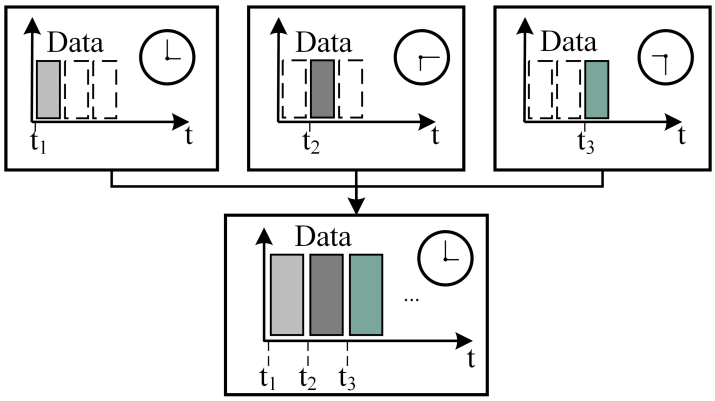


Abbildung 3.2: Funktionsweise des Zeitmultiplexverfahrens.

Qbv – Enhancements for scheduled traffic: Der IEEE 802.1Qbv Standard war zunächst ein eigenständiger Nachtrag zum IEEE 802.1Q-Standard. Inzwischen ist er jedoch auch in diesem integriert. 802.1Qbv beschreibt unter anderem die Umsetzung einer Transmission Selection mit sogenannten Gates. In Abbildung 3.3 ist diese Umsetzung dargestellt. Es existieren bis zu acht Queues, denen die Traffic Classes zugewiesen werden. Für jede Queue kann ein TSA ausgewählt werden. Ein TSA wählt, entsprechend seiner Logik, die Frames aus, die beim nächsten Öffnen der Gates übertragen werden. Ein Gate kann sich entweder im geöffneten Zustand (Open = o) oder im geschlossenen Zustand (Closed = C) befinden. Wenn ein Gate geöffnet ist, werden die Frames, die vom TSA ausgewählt wurden, übertragen. Wenn das Gate geschlossen ist, werden keine Frames dieser Queue übertragen. Für jeden Port wird eine Gate Control List (GCL) angelegt, die die Zustände der Gates zu einem relativen Zeitpunkt enthält.

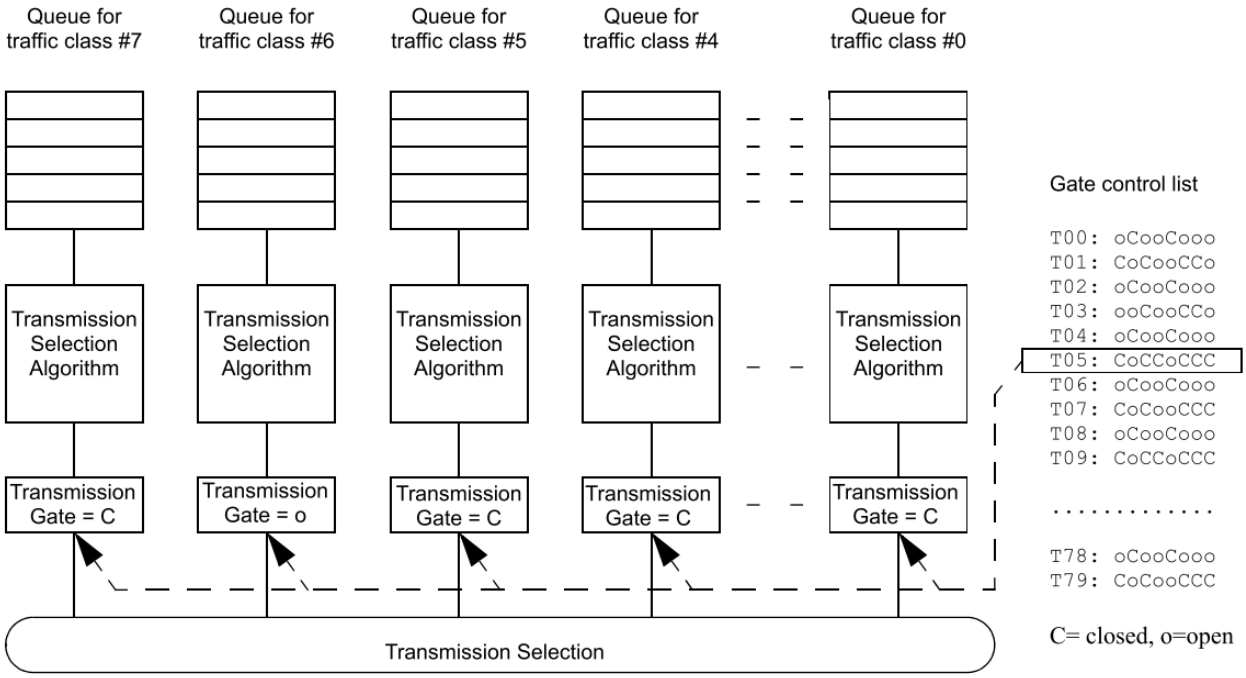


Abbildung 3.3: Enhanced Transmission Selection und Gate Control List (Norm IEEE 802.1Q).

3.4 OPC Unified Architecture

OPC UA ist eine Sammlung von Standards, die eine interoperable, plattformübergreifende, IT-sichere und über alle OSI-Schichten durchgängige Industrie-4.0-Kommunikation gewährleisten (Rauen et al. 2017). OPC UA ist der Nachfolger des OPC-Standards, der auf der Microsoft Technologie COM / DCOM basiert (Norm DCOM). OPC UA bietet die Möglichkeit, herstellerunabhängig und plattformübergreifend Daten zu modellieren und zu übertragen. Die Übertra-

gung der Daten basiert dabei auf einer Service-oriented Architecture (SOA). Außerdem bietet OPC UA ein ganzheitliches Sicherheitskonzept an. OPC UA besteht aus 14 Standards, die in der Tabelle 3.3 aufgelistet sind.

Tabelle 3.3: OPC UA-Standards.

Overview (Norm OPC UA Part 1)	Data Access (Norm OPC UA Part 8)
Security (Norm OPC UA Part 2)	Alarms and Conditions (Norm OPC UA Part 9)
Address Space (Norm OPC UA Part 3)	Programs (Norm OPC UA Part 10)
Services (Norm OPC UA Part 4)	Historical Access (Norm OPC UA Part 11)
Information Model (Norm OPC UA Part 5)	Discovery (Norm OPC UA Part 12)
Mappings (Norm OPC UA Part 6)	Aggregates (Norm OPC UA Part 13)
Profiles (Norm OPC UA Part 7)	PubSub (Norm OPC UA Part 14)

Die Standards bieten Mechanismen zum Austausch von Informationen zwischen IT und OT. OPC UA verfügt unter anderem über die Möglichkeit, Geräte über Informationsmodelle zu modellieren. Dazu bereiten die Informationsmodelle die Daten der Geräte über eine eindeutige Semantik und Syntax zu Informationen auf. Informationsmodelle bestehen aus OPC UA-Nodes und Referenzen. Nodes repräsentieren Dinge und Objekte. Referenzen beschreiben die Art der Beziehung zu anderen Nodes. Außerdem bietet OPC UA unterschiedliche Übertragungsmechanismen, die z. B. auf der OPC UA Client/Server-Architektur oder der OPC UA PubSub-Architektur basieren. Letztere wurde im Februar 2018 in der 14-ten Spezifikation veröffentlicht und erweitert damit das Portfolio von OPC UA um eine entkoppelte Kommunikation. OPC UA hat sich bereits mit der Client/Server-Architektur bei vielen Anwendungsfällen zu dem De-facto-Standard in der heutigen Industrie durchgesetzt. Bisher ist OPC UA aber nur bedingt in die Feldebene vorgedrungen. Mit OPC UA PubSub will OPC UA nun auch einen Übertragungsmechanismus bereitstellen, der die Anforderungen an die Kommunikation heutiger flexibler Fertigungssysteme erfüllt. Ziel ist es, OPC UA auch auf der Feldebene zu etablieren und dadurch eine durchgängige interoperable Kommunikationslösung zu bieten.

In den nachfolgenden zwei Unterkapiteln wird die OPC UA Client/Server-Architektur und die OPC UA PubSub-Architektur kurz vorgestellt. In einem weiteren Schritt wird bewertet, ob die bisherigen Lösungen die Anforderungen an den überwachten Datenaustausch erfüllen.

3.4.1 Client/Server-Architektur

Die Client/Server-Architektur von OPC UA basiert auf einer gekoppelten 1-zu-1-Verbindung zwischen einem Client und einem Server. Diese gekoppelte Verbindung basiert auf TCP/IP. Der OPC UA-Server stellt Informationen z. B. über ein physikalisches Gerät oder eine Software zur Verfügung. Der OPC UA-Server stellt dabei die Informationen in seinem Adressraum strukturiert dar. Um die Informationen auszulesen oder zu schreiben, verbindet sich der OPC UA-Client mit dem OPC UA-Server. Über die OPC UA-Dienste Read, Write, Browse und Subscribe kann der OPC UA-Client auf die Informationen zugreifen. Die OPC UA Client/Server-Dienste basieren auf dem Anfrage/Antwort-Prinzip. Bei diesem Prinzip baut der OPC UA-Client eine Verbindung zum OPC UA-Server auf. Bei dieser Verbindung handelt es sich um eine gekoppelte 1-zu-1-Verbindung zwischen OPC UA-Client und OPC UA-Server. Der OPC UA-Client und der OPC UA-Server müssen diese Verbindung aufrechterhalten, was zu einem zusätzlichen Ressourcenverbrauch auf den Geräten führt. Der Dienst Subscription unterscheidet sich stark von der Funktionsweise eines Subscribers und darf deshalb nicht verwechselt werden. Eine Subscription aus der Client/Server-Architektur hat nichts mit dem Subscriber einer Publish/Subscribe-Architektur zu tun. Eine Subscription basiert auf einer verbindungsorientierten Kommunikation. Beim Subscription-Dienst erstellt OPC UA-Client eine Session zum OPC UA-Server. Eine Session wird zum Verwalten der Verbindung zwischen OPC UA-Client und OPC UA-Server genutzt. In der OPC UA-Server-Application werden die Subscriptions und die MonitoredItems angelegt. Letztere überwachen die Nodes und sammeln Daten. Die Subscriptions sind Entitäten, die zyklische Daten über die Session an den OPC UA-Client senden. Die Subscriptions bieten die Möglichkeit, verloren gegangene Nachrichten zu erkennen und erneut zu senden. In Abbildung 3.4 ist dieser Mechanismus dargestellt.

Die typischen Eigenschaften einer Client/Server-Architektur sind folgende Punkte:

- Eine Kommunikation mit einer 1-zu-1-Kardinalität
- Ein erhöhter Ressourcenbedarf beim Verwalten der Kontext-Informationen bestehender Verbindungen
- Eine gekoppelte Kommunikation basierend auf dem TCP-Protokoll:
  - Datenintegrität und Zuverlässigkeit
  - Fehlererkennung und Reaktion auf unterschiedlichen Ebenen
  - Flusssteuerung
  - Staukontrolle

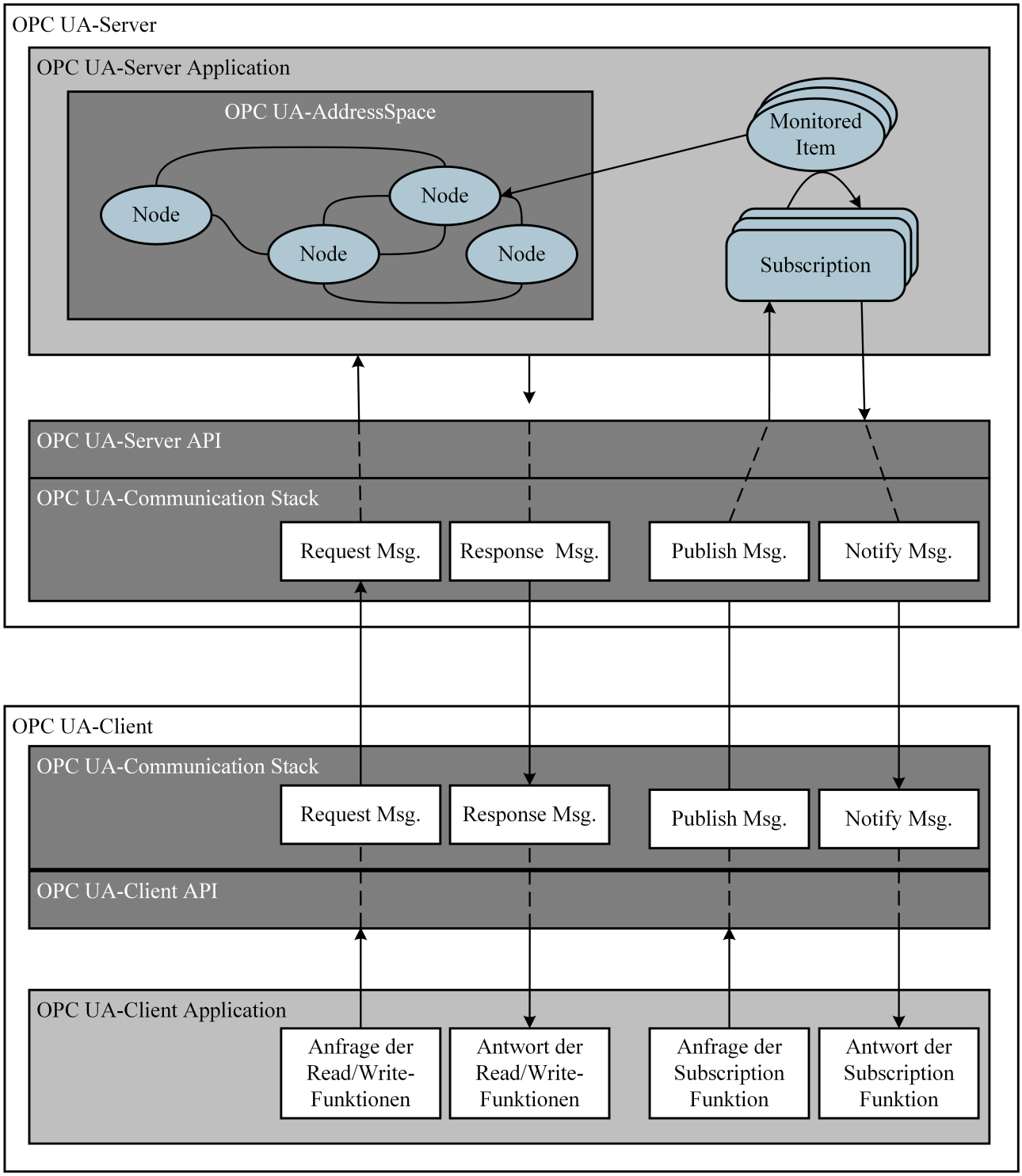


Abbildung 3.4: Client/Server-Architektur.

Die Eigenschaften der Client/Server-Architektur eignen sich nur bedingt für die Kommunikation in der Feldebene. Auf Grund der zusätzlichen Kontext-Informationen ist die verwendete Hardware oftmals nicht performant genug, um viele 1-zu-1-Verbindungen zu verwalten. Außer-

dem sind in der Feldebene Zykluszeiten zwischen 125 µs und 10 ms üblich, weshalb die Vorteile von TCP oftmals nicht zum Tragen kommen. Beispielsweise hilft es bei Paketausfall nichts, die Retransmit-Funktionen von TCP zu nutzen, um das Paket erneut zu übertragen, da der Paketinhalt inzwischen veraltet ist. Diese beiden Eigenschaften machen Client/Server-Architekturen eher ungeeignet für die Feldebene. Um nun OPC UA-Kommunikationstechnologie in der Feldebene zu etablieren, erweitert OPC UA PubSub den Funktionsumfang von OPC UA um eine entkoppelte Kommunikation.

3.4.2 Publish/Subscribe-Architektur

Bei der OPC UA PubSub-Architektur findet die Kommunikation zwischen einem Sender (Publisher) und einem oder mehreren Empfängern (Subscriber) statt. Es ist eine Kardinalität von 1-zu-N möglich. Der große Unterschied zu der Client/Server-Architektur liegt darin, dass die Kommunikation zwischen dem Publisher und den Subscribern entkoppelt stattfindet. Entkoppelt bedeutet, dass die Publisher und Subscriber keine Verbindung über z. B. einen 3-Way-Handshake aufbauen, bevor der eigentliche Datenaustausch ausgeführt wird. In Abbildung 3.5 wird zunächst eine inhaltliche Übersicht über OPC UA PubSub gegeben, sowie die Wechselwirkung mit der Client/Server-Architektur dargestellt. OPC UA PubSub basiert auf der Verwendung einer Middleware. Es werden zwei Konzepte als Middleware angeboten – die brokerbasierte und die brokerlose Middleware. Ein Broker ist eine Instanz zwischen den kommunizierenden Entitäten, die die Nachrichten verwaltet und weiterleitet. Neben den Middleware-Konzepten sind im OPC UA PubSub-Standard vier Transportprotokolle definiert. Die Transportprotokolle Message Queuing Telemetry Transport (MQTT) (Norm OASIS MQTT) und Advanced Message Queuing Protocol (AMQP) (Norm OASIS AMQP) sind Protokolle für das brokerbasierte Konzept (Norm OPC UA Part 14). Die beiden anderen Message Protocols, OPC UA-Ethernet und OPC UA-UDP, sind die üblichen Protokolle für das brokerlose Konzept. Neben den Middleware Konzepten und Transportprotokollen werden noch zwei Message Mappings definiert. Diese beschreiben die Enkodierung des Nachrichteninhalts. Das Message Mapping JavaScript Object Notation (JSON) wird üblicherweise für das Broker-basierte Konzept verwendet (Norm RFC 8259). Message Mapping UADP basiert auf einem Binary Encoding. Des Weiteren beinhaltet die Spezifikation ein Informationsmodell zur Modellierung der Publisher und Subscriber. Damit können die Publisher und Subscriber im Adressraum eines OPC UA-Servers abgebildet und konfiguriert werden. Im nachfolgenden wird eine brokerlose Kommunikation über OPC UA-Ethernet und -UDP und dem Message Mapping UADP vorgestellt.

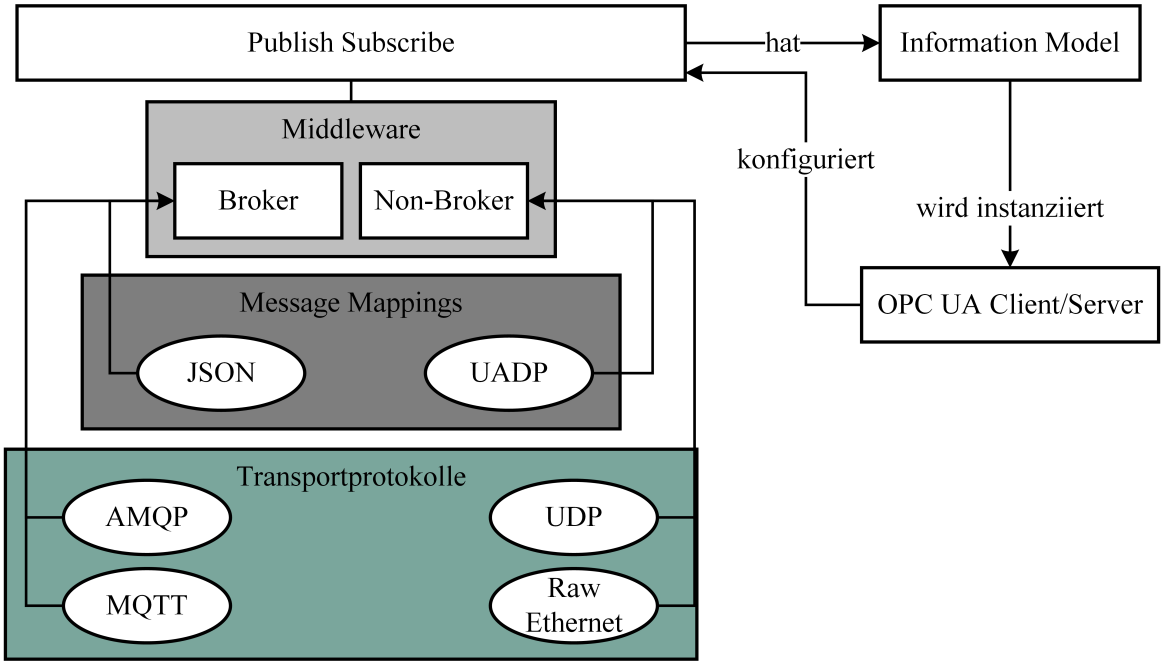


Abbildung 3.5: OPC UA PubSub-Portfolio.

3.4.2.1 Brokerbasierte Middleware

Bei der brokerbasierten Middleware wird ein Broker für die Verwaltung und Verteilung von Nachrichten verwendet. Ein Publisher sendet seine Nachrichten an den Broker. Beliebig viele Subscriber können ihr Interesse an den Nachrichten beim Broker anmelden. OPC UA PubSub definiert hier keine neue Broker-Architektur, sondern greift auf bereits etablierte Architekturen z. B. MQTT und AMQP zurück. Im nachfolgenden wird MQTT als Beispiel kurz vorgestellt und auf die angebotenen QoS eingegangen.

Message Queuing Telemetry Transport: MQTT basiert auf einer Client/Server-Architektur und nutzt Publish Subscribe-Mechanismen um Nachrichten zwischen dem Server und einem oder mehreren Clients zu verteilen. Ein Broker ist ein Server und die Publisher/Subscriber sind Clients. Ein Publisher veröffentlicht seine Daten beim Broker und einer oder mehrere Clients können sich auf diese Daten subskribieren. Dadurch erreicht MQTT eine Kardinalität von 1-zu-N. MQTT ist nicht auf ein einzelnes Protokoll zur Kommunikation mit dem Broker eingeschränkt. Es ist nur wichtig, dass das verwendete Transportprotokoll die Pakete richtig anordnet und verlustfrei überträgt. Im Standard wird dafür das Transportprotokoll TCP verwendet (Norm RFC 793). Einige alternativen sind die Transportprotokolle Transport Layer Security (TLS) (Norm RFC 9446-1.3) und WebSocket (Norm RFC 6455). Die drei QoS von MQTT werden nun vorgestellt.

At Most Once: Die Nachricht wird entweder einmal oder keinmal erfolgreich übertragen. In Abbildung 3.6 ist der Kontrollalgorithmus der Übertragung einer Nachricht mit QoS 0 zwischen MQTT-Sender und MQTT-Receiver dargestellt. Der Sender sendet dazu die Nachricht mit der QoS 0 an den Receiver. Die empfangene Nachricht ist dann Eigentum des Receivers.

Abbildung 3.6: QoS 0: At Most Once.

At Least Once: Die Nachricht wird mindestens einmal oder mehrfach erfolgreich übertragen. In Abbildung 3.7 ist der Kontrollalgorithmus der Übertragung einer Nachricht mit QoS 1 zwischen MQTT-Sender und MQTT-Receiver dargestellt. Der Sender sendet eine Nachricht mit der QoS 1 und mit einer Nachrichten-ID an den Receiver. Solange der Sender keine Empfangsbestätigung (PUBACK) erhalten hat, ist für ihn die Nachricht noch nicht erfolgreich vom Empfänger empfangen. In diesem Fall würde nach angemessener Zeit ein erneutes Senden ausgelöst werden. Beim Empfangen einer Nachricht mit der QoS 1 sendet der Receiver eine Empfangsbestätigung an den Sender zurück, die die zugehörige Nachrichten-ID enthält.

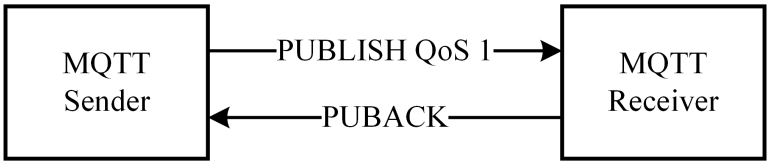


Abbildung 3.7: QoS 1: At Least Once.

Exactly Once: Die Nachricht wird genau einmal erfolgreich übertragen. In Abbildung 3.8 ist der Kontrollalgorithmus der Übertragung einer Nachricht mit QoS 2 zwischen MQTT-Sender und MQTT-Receiver dargestellt. Dabei handelt es sich um den höchsten QoS die das MQTT bietet. Bei diesem QoS ist weder Datenverlust noch Duplikation akzeptabel. Dazu sendet der Sender zunächst die Nachricht mit einer ID und dem QoS 2 an den Receiver. Der Receiver sendet eine PUBREC-Nachricht an den Sender zurück. Sobald der Sender die Empfangsbestätigung erhalten hat, kann er die originale Nachricht löschen. Der Sender antwortet mit einer PUBREL-Nachricht mit der gleichen Nachrichten-ID. Sobald der Empfänger mit der Verarbeitung des Nachrichteninhalts der Publish-Nachricht fertig ist, antwortet er mit einer PUBCOMP-Nachricht auf die PUBREL-Nachricht. Während des Verarbeitungsprozesses muss der Receiver

die Nachrichten-ID abspeichern. Dies ist wichtig, um eine erneute Verarbeitung der Nachricht zu vermeiden.

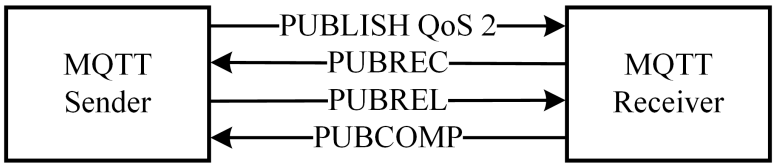
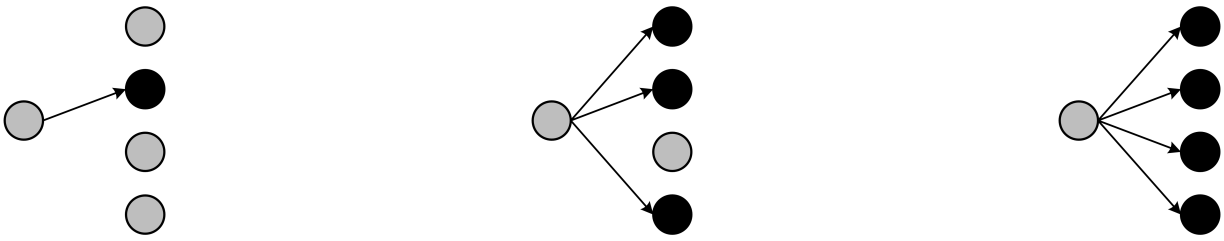


Abbildung 3.8: QoS 2: Exactly Once.

3.4.2.2 Brokerlose Middleware

Bei der brokerlosen Middleware wird ausschließlich die Netzwerkinfrastruktur zur Verteilung der Nachrichten verwendet. Das heißt, es gibt keine Zwischenentität wie den Broker. Das hat den Vorteil, dass es zu keinen weiteren Verzögerungen bei der Übertragung durch einen Broker kommt. Um die Nachrichten zu übertragen, werden die Transportprotokolle OPC UA-Ethernet und -UDP verwendet.

OPC UA-Ethernet und -UDP: Beide Protokolle sind die bevorzugten Transportprotokolle für die brokerlose Kommunikation. OPC UA-Ethernet besitzt einen eigenen IANA Ethertype B62C (IEEE Ethertype 18. 03. 2020). In Abbildung 3.9 sind die üblichen Adressierungsschemata von OPC UA-Ethernet und -UDP dargestellt. Bei der Unicast-Adressierung empfängt nur das Gerät mit der entsprechenden Unicast-Adresse die Nachrichten. Bei Multicast empfangen nur die Geräte die Nachricht, die sich zuvor auf die Multicast-Adresse registriert haben. MAC-Multicast-Adressen haben einen Präfix von 24-Bit 01:00:5E:XX:XX:XX. IPv4-Multicast-Adressen befinden sich zwischen den IPv4-Adressen 224.0.0.0 und 239.255.255.255 (IANA 2019 2019). Bei der Broadcast-Adressierung empfangen alle Geräte die Nachricht. Bei der MAC-Broadcast-Adresse handelt es sich um die FF:FF:FF:FF:FF:FF und die IPv4-Broadcast-Adresse ist die höchste Adresse in ihrer Klasse, z. B. 192.168.1.255/24.



(a) *Unicast.* **(b)** *Multicast.* **(c)** *Broadcast.*

Abbildung 3.9: Adressierungsschemata von OPC UA-Ethernet und -UDP.

Der Publisher kann wahlweise zwischen den Transportprotokollen OPC UA-Ethernet und -UDP auswählen. Der Publisher integriert seine Daten in die Payload dieser verbindungslosen Protokolle. Dazu erstellt der Publisher zunächst eine NetworkMessage. Im OPC UA PubSub-Standard ist im Message Mapping UADP die Struktur einer NetworkMessage definiert.

Da es sich um eine entkoppelte Kommunikation zwischen Publisher und Subscriber handelt, sind in jeder NetworkMessage gewisse IDs enthalten, die den Publisher und die weiteren Dateninhalte der NetworkMessage identifizieren.

3.4.2.3 Message Mapping: UADP

Das Message Mapping UADP basiert auf dem Open Platform Communications (OPC) Binary Encoding. UADP wurde für eine entkoppelte Kommunikation zwischen einem Publisher und mindestens einem Subscriber entwickelt. UADP strukturiert die Payload von z. B. OPC UA-Ethernet oder -UDP in Form einer NetworkMessage. In Abbildung 3.10 ist der Aufbau einer NetworkMessage dargestellt. Eine NetworkMessage besteht aus einem Header, einer Payload und einem Tail. Sollte Security aktiviert sein, so wird die Payload verschlüsselt und signiert. Der Header bleibt unverschlüsselt, da er die ID des Publishers (PublisherId) zur Identifikation der NetzwerkMessage enthält.

Die Payload der NetworkMessage besteht aus einer oder mehreren DataSetMessages. Eine DataSetMessage setzt sich aus einem Header und einem DataSet zusammen. Der DataSetMessage Header beinhaltet die WriterId zur Identifikation der DataSetMessage. Eine DataSetMessage ist eine Gruppierung von DataSetFields. Ein DataSetField entspricht wiederum einer Variable. Zum Beispiel könnte ein Temperaturwert eines Sensors als DataSetField übertragen werden.

Im DataSetMessage Header sind außerdem Informationen enthalten, welche Enkodierung bei den DataSetFields verwendet wurde. Mögliche Enkodierungen sind hier entweder ein Raw Encoding, ein Variant oder ein DataValue. Das Raw Encoding entspricht den aneinander gereihten DataSetFields, ohne weitere Typinformationen oder Semantik. Beim Variant Encoding wird vor jedem DataSetField ein Type-Identifier beigefügt. Das DataValue Encoding ergänzt das DataSetField um weitere Semantik wie z. B. Zeitstempel oder Sequenznummer. Ein Subscriber filtert über die PublisherId und die WriterId die relevanten DataSetMessages. Der Publisher kann dadurch über eine Nachricht Subscriber-spezifische Informationen an die Subscriber senden. Die Subscriber können beim Dekodieren über die IDs die DataSetMessages, die nicht für sie bestimmt sind, überspringen.

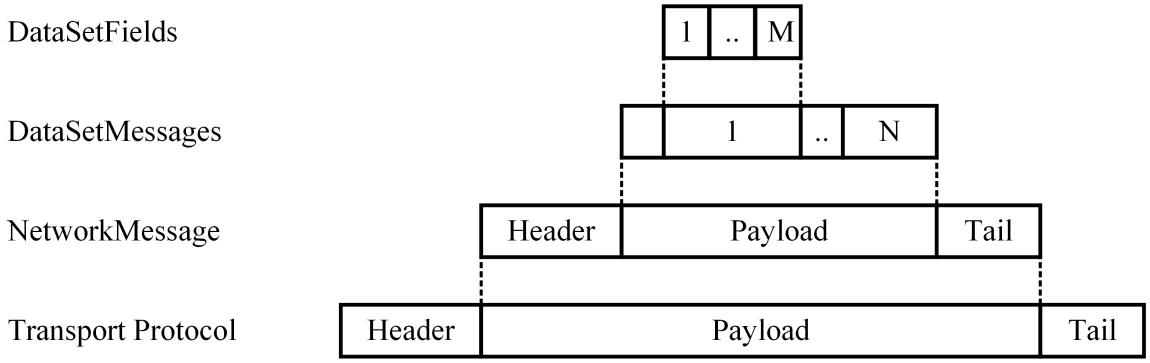


Abbildung 3.10: Verkapselung eines Transportprotokolls mit UADP.

3.4.2.4 Statisches UADP Header Layout

UADP ist sehr flexibel und wird für viele unterschiedliche Anwendungsfälle genutzt. Über sogenannte Header Flags wird eine NetworkMessage an den Anwendungsfall angepasst. Konfigurationsmöglichkeiten sind z. B. Security, Zeitstempel, Sequenznummern und die statische oder dynamische Anordnung von DataSetMessages. Über die statische Anordnung von DataSetMessages ist garantiert, dass sich die Reihenfolge der DataSetMessages in einer NetworkMessage nicht ändert.

Über die Konfiguration der Header Layouts der NetworkMessage und der DataSetMessage wird zusätzlich an Effizienz gewonnen. Effizienz bedeutet hier, dass für die gleiche Menge von Nutzdaten, weniger Bytes für die Kommunikation nötig sind, sowie weniger Prozesslast beim Enkodieren und Dekodieren der NetworkMessage entsteht.

Für den Anwendungsfall der zyklischen Kommunikation von Echtzeitdaten wird das Header Layout aus der Abbildung 3.11 festgelegt. Zusätzlich können noch die Security Header aktiviert werden, wenn es der Anwendungsfall verlangt.

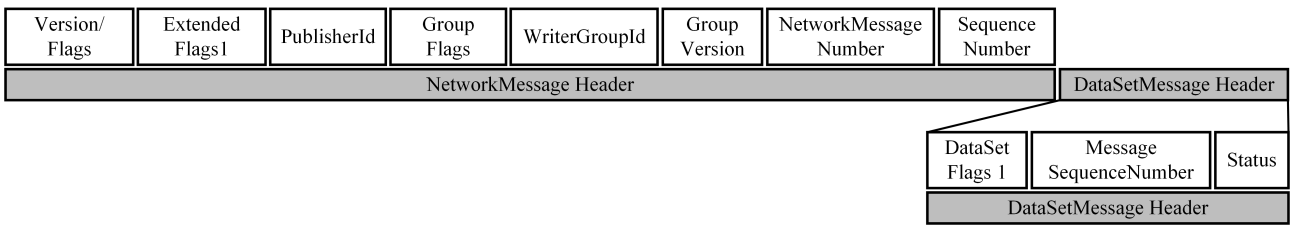


Abbildung 3.11: Layout für statischen Austausch von Echtzeitdaten (Norm OPC UA A.6).

3.4.2.5 Dynamisches UADP Header Layout

Beim dynamischen UADP Header Layout ist die Anzahl und Anordnung der DataSetMessages flexibel. Das heißt, um die relevanten DataSetMessages aus der NetworkMessage zu filtern, muss der Subscriber beim Empfang der NetworkMessage die einzelnen Header auswerten. Das daraus resultierende Layout der Header ist in Abbildung 3.12 dargestellt. Zusätzlich kann Security aktiviert werden, wenn es der Anwendungsfall verlangt.

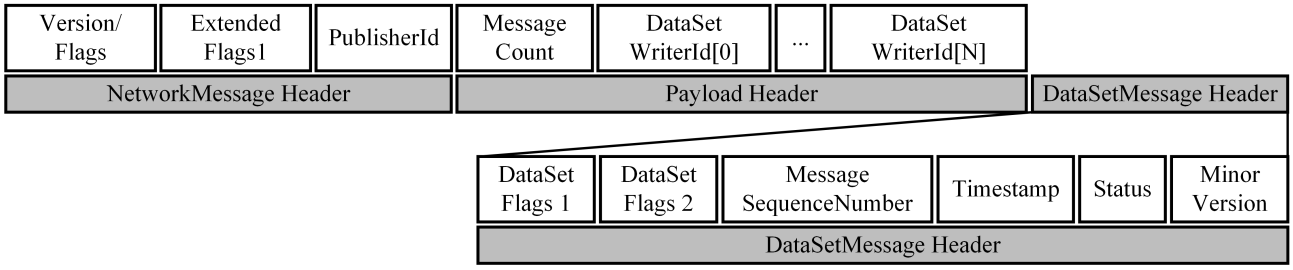


Abbildung 3.12: Layout für dynamische Kommunikation (Norm OPC UA A.6).

3.4.2.6 Zusammenhang der Entitäten und des UADP Message Mappings

Es existieren neben dem Publisher und Subscriber noch weitere abstrahierte Entitäten, die eine Funktion bei der Kommunikation erfüllen. Die Abstraktion eines Publishers ist in Abbildung 3.13 dargestellt. Auf Seiten des Senders gibt es zunächst eine Datenquelle, die z. B. ein OPC UA-Adressraum ist. Im Adressraum werden die Variablen als OPC UA-Nodes dargestellt. Eine DataSet gruppiert die Variablen. Ein DataSetWriter erzeugt eine DataSetMessage, die aus einem Header und einem DataSet besteht. Der Header enthält IDs zur Identifikation der DataSetMessage sowie Zeitstempel und Sequenznummern. Der Publisher erstellt die gesamte NetworkMessage, indem er den NetworkMessage Header mit den DataSetMessages zusammenfügt. Danach übergibt der Publisher die Nachricht dem Netzwerk-Stack. Da es sich um eine entkoppelte Kommunikation handelt, ist dem Publisher nicht bekannt, ob die Subscriber die Nachricht wirklich empfangen.

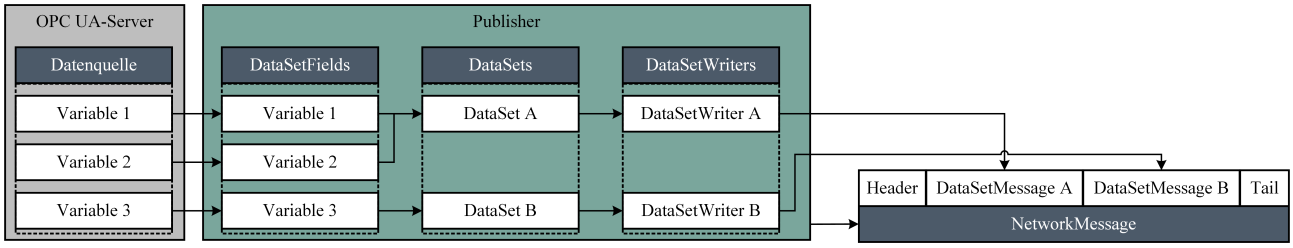


Abbildung 3.13: Abstraktion der Funktionalität eines Publishers.

Der Subscriber ist das Gegenstück zu einem Publisher. Die Abstraktion eines Subscribers ist in Abbildung 3.14 dargestellt. Er besteht abstrahiert aus einem oder mehreren DataSetReadern, SubscribedDataSets und DataSetFields. Der Subscriber ist verantwortlich, die Nachricht zu empfangen und nach den PublisherIds zu filtern. Der DataSetReader dekodiert die relevante DataSetMessage und kopiert den Inhalt in das SubscribedDataSet. Sollte der Subscriber mehrere DataSetReader besitzen, so können diese über DataSetReaderGroups gruppiert werden. Ein SubscribedDataSet besteht aus DataSetFields, die mit einer Datenquelle – z. B. ein OPC UA-Server – verknüpft sind.

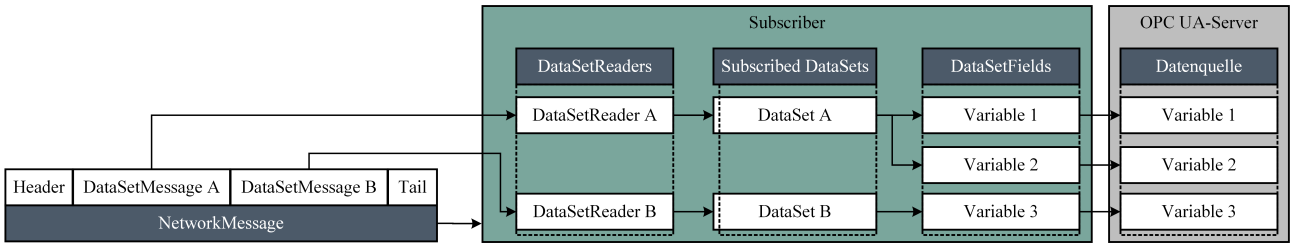


Abbildung 3.14: Abstraktion der Funktionalität eines Subscribers.

3.4.2.7 Zustandsmaschine

Prinzipiell bietet OPC UA die Möglichkeit, Zustandsmaschinen in einem Informationsmodell abzubilden (Norm OPC UA Part 5). Eine Zustandsmaschine, wie sie in Abbildung 3.15 dargestellt ist, wird in OPC UA über ObjectTypes, VariableTypes und ReferenceTypes modelliert. Dazu werden Zustände als StateTypes und Transitionen als TransitionTypes modelliert.

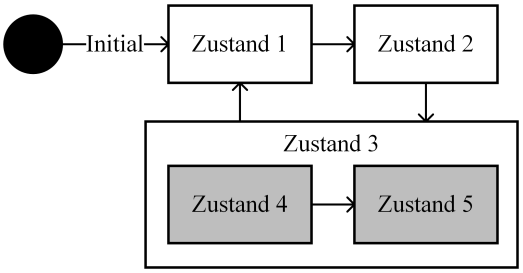


Abbildung 3.15: Zustandsmaschine mit Unter-Zustandsmaschine (Norm OPC UA Part 5).

OPC UA bietet die Möglichkeit, durch die Definition eigener StateTypes und TransitionTypes eigene Zustandsmaschinen zu modellieren. In Abbildung 3.16 ist beispielhaft dargestellt, wie ein Informationsmodell einer Zustandsmaschine mit zwei Zuständen und einer Transition zwischen Zustand 1 und Zustand 2 modelliert ist. Die Transition wird durch die Methode ChangeState-Method ausgelöst. Außerdem kann ein Event als Auslöser für eine Transition verwendet werden.

Letzten Endes bietet das Informationsmodell das Werkzeug, um endliche Zustandsmaschinen zu modellieren.

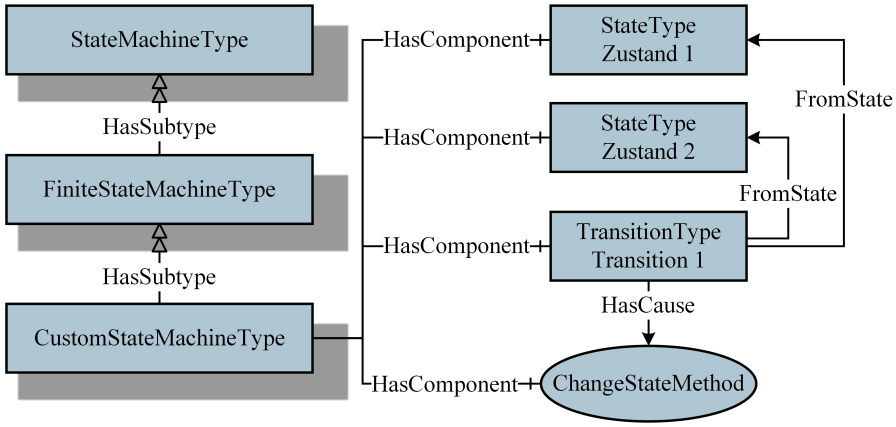


Abbildung 3.16: Informationsmodell einer finiten Zustandsmaschine (Norm OPC UA Part 5).

Im OPC UA PubSub-Standard ist bereits eine allgemeine Zustandsmaschine für die Entitäten der Publisher und Subscriber enthalten. Die Zustandsmaschine besteht aus vier Zuständen – Paused, Disabled, Operational und Error – sowie mehreren Transitionen. Die OPC UA PubSub-Zustandsmaschine ist in Abbildung 3.17 abgebildet. Die Zustandsmaschine ist sehr allgemein und abstrakt gehalten.

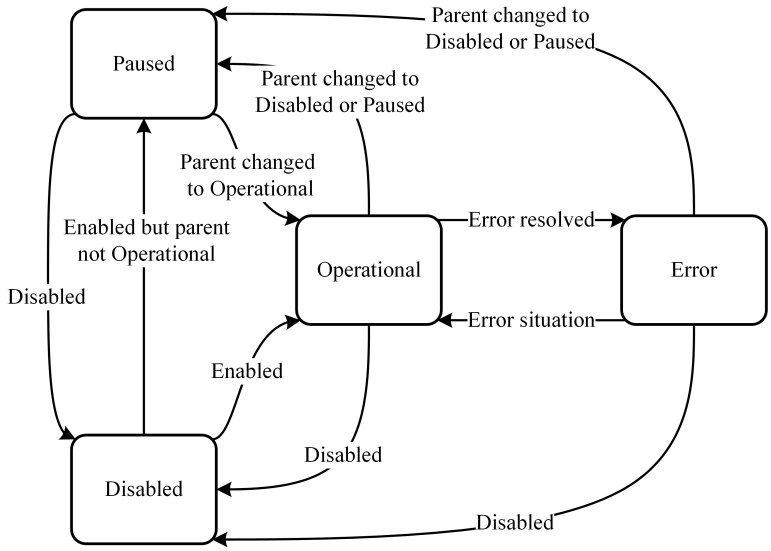


Abbildung 3.17: Zustandsmaschine von OPC UA PubSub (Norm OPC UA Part 14).

3.4.3 Vergleich und Bewertung bisheriger Lösungen von OPC UA

Die Architekturen von OPC UA unterscheiden sich in ihren Charakteristiken in vielen Punkten. In Abbildung 3.18 wird die Eignung von den OPC UA-Architekturen hinsichtlich der Echtzeitfähigkeit, der Überwachung des Datenaustauschs und der Skalierung für C2C-Anwendungsfälle bewertet.

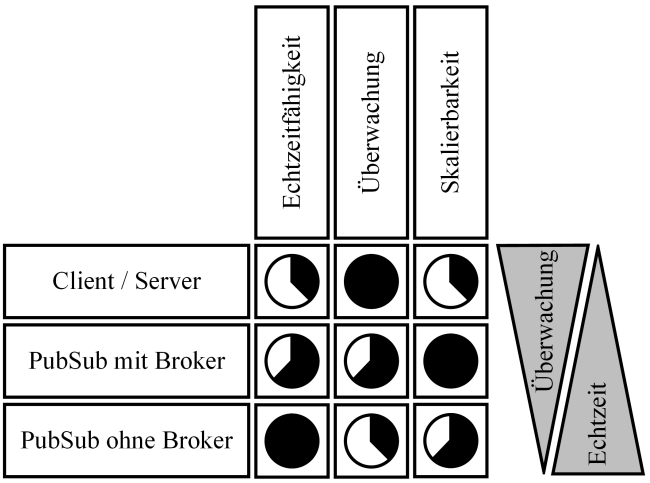


Abbildung 3.18: Bewertung der OPC UA-Architekturen auf ihre Eignung.

Die Client/Server-Architektur basiert auf einer gekoppelten Kommunikation zwischen Client und Server. Auf Grund der Kopplung wird erkannt, wenn es zu Unregelmäßigkeiten beim Datenaustausch kommt. Die Fehlerbehandlung, z. B. ein Retransmit, ist jedoch nicht für C2C-Anwendungsfälle geeignet. Die Folge ist, dass die Client/Server-Architektur im Punkt Echtzeitfähigkeit der Publish/Subscribe-Architektur unterlegen ist. Da Client/Server-Archi-tekturen immer eine Kardinalität von 1-zu-1 verwenden, skalieren Client/Server-Lösungen nicht gut.

Die Publish/Subscribe-Architektur mit einem Broker zwischen Publisher und Subscriber ist auch nicht ideal im Punkt Echtzeit. Der Broker fügt weitere Latenz in die Kommunikation ein. Die Skalierung von OPC UA PubSub mit einem Broker ist sehr gut, da beliebig viele Entitäten ihr Interesse an den Daten beim Broker anmelden können. Dadurch werden sämtliche Kardinalitäten unterstützt. Die Überwachung hängt von der Wahl des Brokers ab. Wird z. B. ein MQTT-Broker verwendet, so können die QoS von MQTT genutzt werden. Bei der Übertragung der Nachrichten unterscheidet MQTT zwischen drei QoS. Je nach QoS ändert sich der Überwachungsalgorithmus.

OPC UA PubSub ohne Broker ist im Vergleich zu den beiden anderen Lösungen die echtzeitfähigste Lösung, da sie keinen Broker oder TCP/IP-Overhead hat. Es werden Kardinalitäten von 1-zu-N unterstützt. Lediglich bei der Überwachung des Datenaustauschs unterliegt die Lösung den anderen Lösungen. Bislang ist es dem Publisher nicht möglich, zu überprüfen, ob seine gesendeten Nachrichten bei den Subscribern ankommen. Hinsichtlich der Überwachung von OPC UA PubSub ohne Broker sind keine Veröffentlichungen bekannt.

3.5 OPC Field Level Communication

Die Vision der OPC FLC Initiative beschreibt eine offene und einheitliche Industrial Internet of Things (IIoT)-Lösung zur Kommunikation zwischen Sensoren, Aktoren, Steuerungen und Clouds, die dabei die Anforderungen heutiger Automatisierungstechnik berücksichtigt. OPC FLC basiert auf den Interoperabilitäts-Standards OPC UA und TSN.

TSN bildet dabei die Funktionalitäten der Schichten 1 und 2 des OSI-Modells und OPC UA die darüber liegenden Schichten ab. Je nach ausgewählter OPC UA-Architektur, werden unterschiedliche Protokolle und Kodierungen der Daten verwendet.

OPC UA PubSub bietet unterschiedliche Middleware-Ansätze, Transportprotokolle und Message Mappings an. OPC FLC definiert deshalb ein Kommunikationsprofil für die Feldebene und legt dadurch fest, welcher Middleware-Ansatz, welches Transportprotokoll und welches Message Mapping verwendet wird. In Abbildung 3.19 wird das Kommunikationsprofil einer verteilten OPC FLC-Applikation dargestellt. Das Kommunikationsprofil beruht auf dem ausgewählten brokerlosen Middleware-Ansatz basierend auf einer Layer-2-Ethernet-Kommunikation und dem UADP Message Mapping.

Die Entitäten Publisher, Subscriber, Writer und Reader sind abstrakte Entitäten, die unterschiedliche Funktionen besitzen und Aufgaben wahrnehmen. Sie sind Bestandteil von Schicht 7 und stellen den Zugriffspunkt auf die Kommunikation für die Applikation dar. Die Applikation kann über Methodenaufrufe die Zustandsmaschinen der Entitäten beeinflussen. In der Darstellungsschicht wird die NetworkMessage enkodiert und als Payload den unteren Schichten zur Verfügung gestellt.

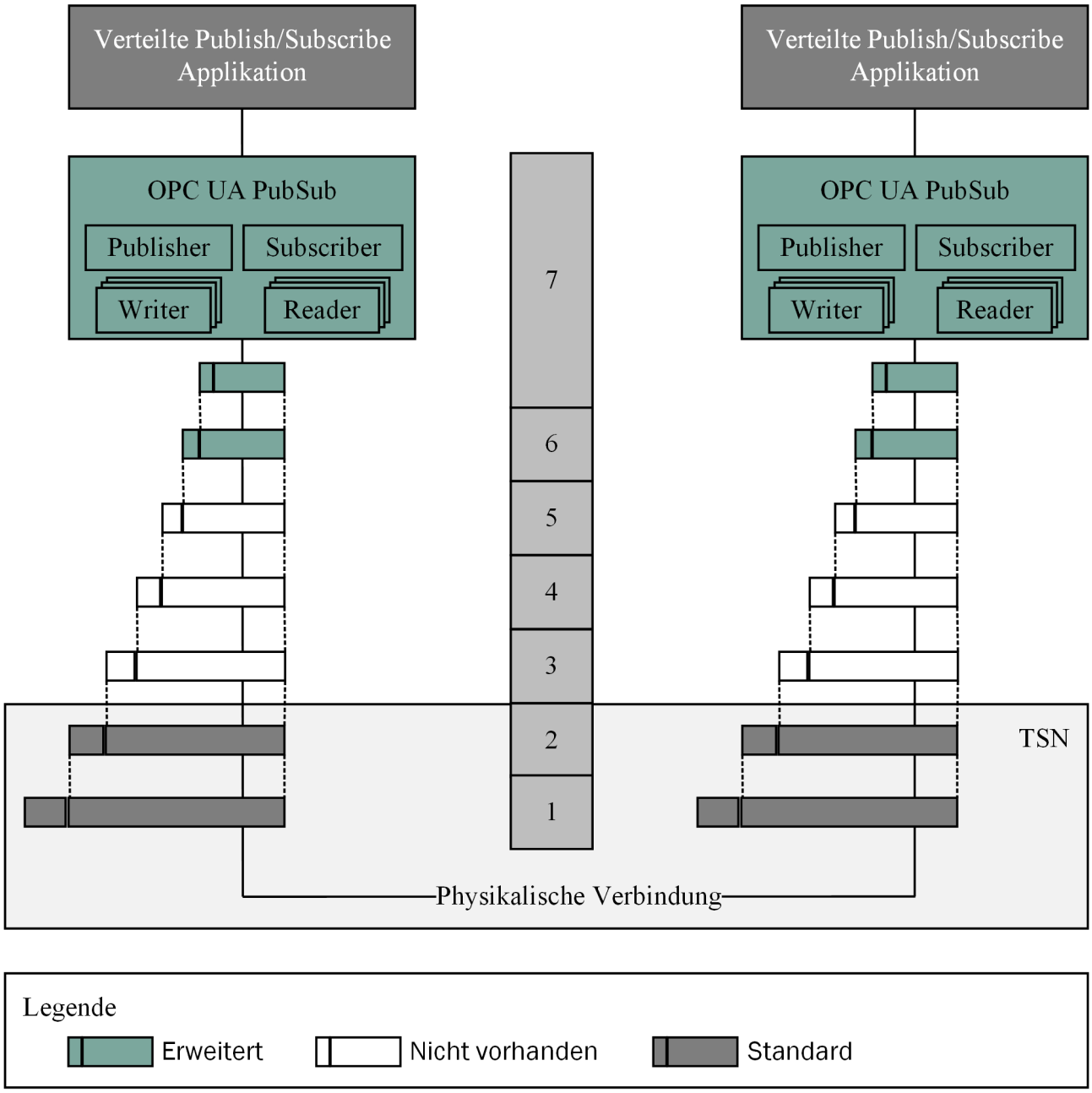


Abbildung 3.19: OPC UA PubSub-Applikation dargestellt im OSI-Modell.

Bisher existiert jedoch nur wenig Literatur über die Kombination von OPC UA und TSN. Hier (Gogolev et al. 2018) wird eine OPC UA Client/Server-Applikation implementiert, um eine Paketumlaufzeitmessung durchzuführen. Bei der Paketumlaufzeit handelt es sich um die Dauer, die notwendig ist, einmal im Kreis zu kommunizieren. Um den Datenstrom zu priorisieren, wird außerdem den versendeten Nachrichten ein VLAN-Tag hinzugefügt. Dadurch werden Paketumlaufzeiten von 2 ms realisiert. Wie bereits von den Autoren erwähnt, ist die Vermutung mit OPC UA PubSub noch kleinere Paketumlaufzeiten zu erreichen.

3.6 Linux als Real-Time-Betriebssystem

Das Standard-Linux Betriebssystem ist ein General Purpose Operating System (GPOS), das für einen maximalen Durchschnitts-Durchsatz entwickelt ist. Hingegen ist das Ziel eines Real-Time Operating Systems (RTOS), Garantien einer Realzeitverarbeitung zu gewährleisten. Die Ziele von GPOS und RTOS stehen im Konflikt zueinander (Reghenzani et al. 2019). Die Realzeitverarbeitung ist nach (Norm DIN 44300-9) „eine Verarbeitungsart, bei der Programme zur Verarbeitung anfallender Daten ständig ablaufbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind“. Ob ein Real-Time (RT)-Betriebssystem notwendig ist, kommt auf die Applikation an. Prinzipiell ist RT in drei Kategorien eingeteilt. Die Kategorien beschreiben, wie die Applikationen auf das Fehlen eines Verarbeitungsergebnisses innerhalb einer Zeitspanne reagieren (Reghenzani et al. 2019; Vaduva et al. 2016; Buttazzo et al. 2006; Kopetz 2011).

- **Hard RT**: Wenn das Verarbeitungsergebnis nicht in einer vorgegebenen Zeitspanne vorliegen, kommt es zu einem Systemfehler.
- **Firm RT**: Das Verpassen einer Zeitspanne ist akzeptabel, wirkt sich aber auf die Qualität der Applikation aus. Ein verspätetes Verarbeitungsergebnis ist außerdem nicht mehr verwendbar.
- **Soft RT**: Das Einhalten der Zeitspanne ist keine gesetzte Voraussetzung. Im Fall einer Verspätung verschlechtert sich die Qualität der Applikation.

Es existieren mehrere Gründe, warum Standard-Linux als RTOS ungeeignet ist (Vaduva et al. 2016):

- **Paging**: Das Paging ermöglicht eine virtuelle Speicherverwaltung. Im Fall eines Seitenfehlers kommt es hier jedoch zu unbegrenzten Verzögerungen.
- **Synchronisierung**: Standardmäßig ist der Linux Kernel nicht unterbrechbar. Ein Prozess, der im Kernel-Kontext aktiv ist, kann durch ein anderes Event nicht unterbrochen werden. Die Verarbeitung des Events muss warten, bis der Prozess im Kernel-Kontext abgeschlossen ist.
- **Batchverarbeitung**: Eine Operation kann zusammen mit anderen Operationen verarbeitet werden. Ein einfaches Beispiel ist hier die Freigabe von Seiten. Es werden mehrere Seiten zur Freigabe übergeben, sodass das Linux-Betriebssystem nicht jede Seite einzeln freigeben muss. Der Zeitpunkt der Bearbeitung des Batchs ist dadurch jedoch nicht festlegbar.

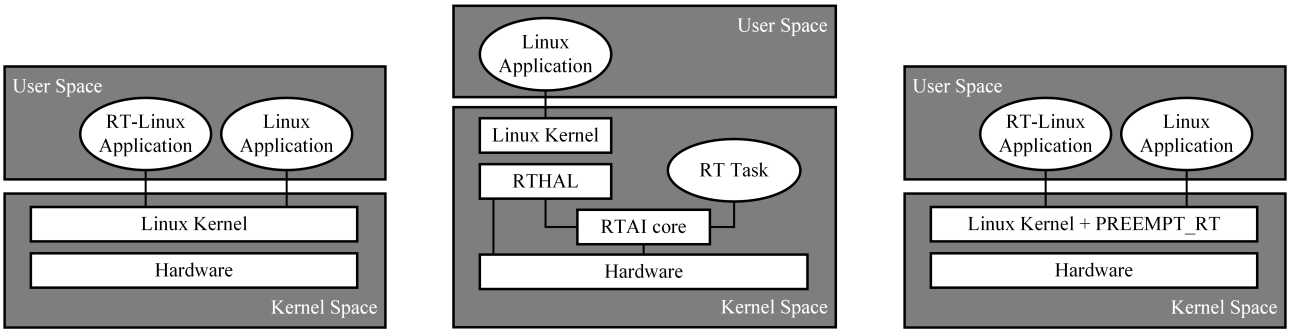
- **Input/Output Anordnung**: Die I/O-Anordnung kann angepasst werden, um die Hardware effizienter einzusetzen. In diesem Kontext bedeutet eine höhere Effizienz mehr Datendurchsatz.
- **Fairness**: Die Standard-Linux Scheduler arbeiten nach dem Fairness-Prinzip. Dadurch bekommen Prozesse trotz niedriger Priorität eine gewisse Bearbeitungszeit.

3.6.1 Systemarchitekturen

Die Standard-Systemarchitektur von Linux ist in Abbildung 3.20(a) dargestellt. Um die Echtzeitfähigkeit von Linux zu verbessern, gibt es unterschiedliche Ansätze. Eine Variante ist die Verwendung eines zusätzlichen Co-Kernels. Co-Kernel werden auch als Pico-Kernel, Nano-Kernel und Dual-Kernel bezeichnet. Co-Kernel existieren parallel zum GPOS-Kernel und verwalten die RT-Applikationen. Die bekanntesten Open-Source Co-Kernel sind Real-Time Application Interface (RTAI)[1], RTLinux[2] und Xenomai[3] (Reghenzani et al. 2019). In Abbildung 3.20(b) ist die Architektur von RTAI dargestellt. Das Real-Time Hardware Abstraction Layer (RTHAL) ist die Schnittstelle zwischen dem GPOS-Linux-Kernel und der Hardware (Dozio et al. 2003).

Eine andere Variante ist die Anpassung des GPOS-Kernels mit sogenannten PREEMPT_RT-Patches[4]. Die Anwendung solcher Patches ist eine verbreitete Lösung, um die Echtzeitfähigkeit von Linux-Betriebssystemen zu verbessern. Ingo Molnár startete die Arbeit an den Patches im Jahr 2005 mit dem Ziel, Linux berechenbarer zu machen. Dies geschieht oftmals auf Kosten des Durchsatzes (Buttazzo et al. 2006). Der große Vorteil der Patches ist, dass sie wirklich das Standard-Linux anpassen und nicht wie die Co-Kernel-Lösungen auf einen zusätzlichen Kernel setzen. Folglich lassen sich RT-Applikationen sehr ähnlich zu Non-RT-Applikationen implementieren (Reghenzani et al. 2019). In Abbildung 3.20(c) ist die Linux-Architektur mit einem PREEMPT_RT-Patch dargestellt.

[1] https://www.rtai.org/
[2] https://www.windriver.com/products/linux/
[3] https://xenomai.org/
[4] http://cdn.kernel.org/pub/linux/kernel/projects/rt/



(a) *Linux-Architektur.* **(b)** *RTAI-Architektur.* **(c)** *PREEMPT_RT-Patch.*

Abbildung 3.20: Unterschiedliche RTOS-Architekturen.

3.6.2 Scheduling Policy

Ein Scheduler ist eine Kernel-Komponente, die entscheidet, welches ausführbare Thread als Nächstes von der Central Processing Unit (CPU) bearbeitet wird. Eine Möglichkeit der Optimierung besteht darin, eine andere Scheduling Policy für die Auswahl des nächsten Threads zu verwenden. Die Scheduling Policies sind in zwei Kategorien – Normal und RT – eingeteilt. Die Priorität der RT-Policies ist immer höher als die Priorität der normalen Policies. Folgende Policies sind in Linux vorhanden:

Normale Schulding Policies:

- SCHED_OTHER
- SCHED_BATCH
- SCHED_IDLE

RT-Scheduling Policies:

- SCHED_FIFO
- SCHED_RR
- SCHED_DEADLINE

Standardmäßig verwendet Linux die Scheduling Policy SCHED_OTHER für Threads, die keine Echtzeitmechanismen benötigen. Die Policy SCHED_FIFO arbeitet nach dem First-in-First-out Prinzip. Bei SCHED_FIFO wird jedem Thread eine Priorität zwischen 1 bis 99 zugewiesen. Threads dieser Policy sind in der Lage, Threads mit normaler Scheduling Policy zu unterbrechen. Es kann passieren, dass durch Threads mit einer SCHED_FIFO-Policy die normalen Threads verdrängt werden. Deshalb wurde der SCHED_RR eingeführt, der auch Bestandteil der RT-Scheduling-Policies ist. Bei SCHED_RR nutzt der Scheduler ein Prinzip der Fairness, sodass normale Threads, die lange inaktiv sind, auch verarbeitet werden. Seit der Kernel Version 3.14 verfügt der Linux Kernel über eine neue Scheduling Policy – der SCHED_DEADLINE. Diese Policy nutzt Algorithmen wie Global Earliest Deadline First (GEDF) in Verbindung mit einem Constant Bandwidth Server, was einem Zeitschlitzverfahren auf der Scheduler-Ebene

entspricht. Die Policy SCHED_DEADLINE hat die höchste Priorität aller Scheduling Policies (Linux Manual 18. 03. 2020).

3.6.3 Linux Kernel Patch

Linux ist standardmäßig als GPOS ausgelegt. Um echtzeitfähiger zu werden, ändert der PREEMPT_RT-Patch folgende Punkte im Linux Kernel (Vaduva et al. 2016):

- Die kritischen Sektionen des Kernels werden durch unterbrechbare *rwlock_t* und *spinlock_t* geschützt.
- Der Mechanismus zum Sperren des Kernels wird mit der Verwendung von sogenannten *rtmutexes* unterbrechbar.
- Die Prioritätsinversion und Prioritätsvererbung ist für *mutexes*, *spinlocks* und *rw_semaphores* implementiert.
- Es werden hochauflösende Zeitgeber bereitgestellt.
- Die Soft Interrupt Handlers werden als Kernel Threads behandelt.

3.6.4 Isolierte CPU-Kerne

Eine weitere Möglichkeit die Worst-Case Execution Time (WCET) zu verbessern, ist die Verwendung von isolierten CPU-Kernen (Brosky et al. 2003). Folgendes Beispiel: Ein System verfügt über eine CPU mit vier Kernen. Standardmäßig würde der Scheduler alle Prozesse auf diesen vier Kernen verteilen. Dadurch können jedoch mehrere negative Effekte auftreten. Es ist möglich, dass Interrupts der nicht-echtzeitkritischen Prozesse die echtzeitkritischen Prozesse negativ beeinflussen. Außerdem ist es möglich, dass der verantwortliche Scheduler die echtzeitkritische Applikation unaufhörlich einem anderen CPU-Kern zuweist. Ein Wechsel des CPU-Kerns führt zu einer Verschlechterung der WCET. Ab Kernel Version 2.5.8 bietet Linux die Möglichkeit, Prozesse auf einen oder mehrere CPU-Kerne zu binden. Dazu wird die CPU-Affinität des Prozesses entsprechend zugewiesen. Um dabei den Prozess vor Einflüssen anderer Prozesse zu schützen, wird die CPU noch isoliert. Folglich können die Scheduler keine weiteren Prozesse diesem CPU-Kern zuweisen. Das gleiche Vorgehen ist für die Interrupt Request (IRQ)-Handlers der Echtzeitprozesse möglich.

3.6.5 Evaluierung der Echtzeitfähigkeit von Linux-Systemen

Um die Echtzeitfähigkeit in einem Linux-System zu bewerten, existieren unterschiedliche Metriken. In Tabelle 3.4 sind übliche Metriken aus der Literatur zur Evaluierung der Echtzeitperformance aufgelistet. Außerdem wird eine Übersicht über die Methoden gegeben, wie in der Literatur die Echtzeitfähigkeit von Linux-Systemen evaluiert wird. Dazu werden drei Kategorien definiert. Unter *Proprietäre Werkzeuge* wird eine von den Autoren eigens implementierte Methode zur Evaluierung der Metrik verstanden.

Unter *Sonstige Werkzeuge* werden Werkzeuge verstanden, die zwar öffentlich zugänglich sind, aber nur einmal in der genannten Literatur verwendet werden. Cyclictest wird explizit in der Tabelle genannt, da es häufig bei Untersuchungen in der Wissenschaft verwendet wird und demnach vergleichbare Daten produziert. Das Werkzeug Cyclictest[5] ist eine Open-Source-Implementierung zur Evaluierung von verschiedener Latenz.

Vor allem die Interrupt-, Scheduling- und User/Kernel-Latenz sind von großer Relevanz bei der Bewertung der Echtzeitfähigkeit. Da die proprietären Werkzeuge oft nicht frei zugänglich sind und oft nur vom Autor selber verwendet werden, ist ein umfassender Vergleich der produzierten Daten meist nur schwer möglich, da sich oft die Hardware-Architektur stark unterscheidet.

Im Rahmen dieser Arbeit wird deshalb die Bewertung der Echtzeitfähigkeit der Evaluierungsplattform unter Einbeziehung des Werkzeugs Cyclictest geschehen, da OSADL[6] eine große Menge an Daten zur Verfügung stellt, die zum Vergleich verwendet werden können. Die zur Verfügung gestellten Daten wurden unter Verwendung unterschiedlicher Hardware-Architekturen erhoben.

[5] `https://wiki.linuxfoundation.org/realtime/documentation/howto/tools/cyclictest/faq/`
[6] `https://www.osadl.org/Home.1.0.html`

Tabelle 3.4: Mögliche Metriken und Methoden zur Evaluierung der Echtzeitfähigkeit eines OS aus der Literatur, angelehnt an (Reghenzani et al. 2019).

Metrik	Feuerer 2007	Mossige et al. 2007	Arthur et al. 2007	Koolwal 2009	Betz et al. 2009	Emde 2010	Brown et al. 2010	Marieska et al. 2011	Litayem et al. 2011	Cerqueira et al. 2013	Fayyad-Kazan et al. 2014	Garre et al. 2014	Chalas 2015	Dantam et al. 2015	Murikipudi et al. 2015	Yanyan et al. 2018
Deadlock Break Time								X								
GPIO-Latenz	X	X	X				X								X	
GPIO-Jitter	X	X	X				X									
Interrupt-Latenz		X		X	X	X		X	X	X	X	X	X		X	
IPC-Leistung	X											X		X		
Netzwerklatenz		X		X										X		
Scheduling-Latenz				X	X			X	X	X	X	X	X			
Semaphore A/R Zeit											X					
Systemdurchsatz			X			X			X		X					
Zeitgeberlatenz						X										
User/Kernel-Latenz		X														X
Verwendete Werkzeuge																
Proprietäre Werkzeuge	X	X			X		X	X							X	X
Sonstige Werkzeuge				X		X			X		X	X				
Cyclictest				X					X	X			X			

3.7 Defizite

Dem Publisher ist nicht bekannt, ob seine Nachrichten bei den Subscribern ankommen, da OPC UA PubSub auf einer entkoppelten Kommunikation basiert und bisher das Protokoll keinen Mechanismus bietet, den Datenaustausch zwischen Publisher und Subscriber zu über-

wachen. Bei vielen Anwendungsfällen ist jedoch diese Information für den Publisher essentiell, da je nach Kommunikationszustand die überlagerte Applikation auf eine fehlerhafte Verbindung reagieren muss. Zur Erfüllung der Anforderungen des C2C-Anwendungsfalls muss zum einen die standardmäßige OPC UA PubSub-Kommunikation als auch die Überwachungserweiterung performant genug sein. Auf Seiten des Betriebssystems ist die Leistung von OPC UA PubSub stark von der Systemarchitektur abhängig. Auf Seiten des Netzwerks bietet TSN die notwendigen Mechanismen, die Echtzeitfähigkeit bei der Übertragung zu gewährleisten.

Wie in diesem Kapitel gezeigt, sind öffentliche Benchmarking-Werkzeuge verfügbar, die sowohl die Echtzeitfähigkeit von Linux als auch die Leistung von TSN analysieren. Bisher gibt es jedoch noch keine Literatur oder öffentliche Werkzeuge, welche die Verarbeitungszeiten und Übertragungszeiten von OPC UA PubSub untersuchen. Infolgedessen sind die Auswirkungen der aufgezeigten Systemdesign-Einflussgrößen, z. B. PREEMPT_RT-Patch oder CPU-Isolation, auf eine OPC UA PubSub-Kommunikation unbekannt.

Es fehlt bisher eine Leistungsmetrik, um die Relation zwischen den Systemdesign-Einflussgrößen und der OPC UA-Kommunikation herzustellen. Eine solche Leistungsmetrik ist eine Grundvoraussetzung zur Konzeption eines überwachten Datenaustauschs auf Basis von OPC UA PubSub und TSN.

3.8 Zielsetzung und Vorgehensweise

Das Ziel ist es, eine Evaluierungsplattform auf Basis von OPC UA PubSub und TSN zu realisieren und es für einen überwachten Datenaustausch in der Feldebene einzusetzen. Zunächst wird überprüft, ob die allgemeinen Anforderungen der Anwendungsfälle aus der Feldebene, die im Kapitel 2 beschrieben sind, erfüllt werden.

Dazu wird eine Methode definiert, mit der die verwendete Kommunikation, basierend auf OPC UA PubSub und TSN, evaluiert werden kann. In Abbildung 3.21 ist das grobe Konzept eines überwachten Datenaustauschs dargestellt. Die Zustandsmaschine muss für den überwachten Datenaustausch erweitert werden, damit die Zustandsmaschine den Publisher und Subscriber kontrollieren kann.

Um die Zustandsmaschine in einem OPC UA-Server abzubilden, ist es notwendig, ein Informationsmodell zu definieren. Um einheitlich den Status der Zustandsmaschine zu übertragen, ist eine Erweiterung des UADP-Protokolls unabdingbar.

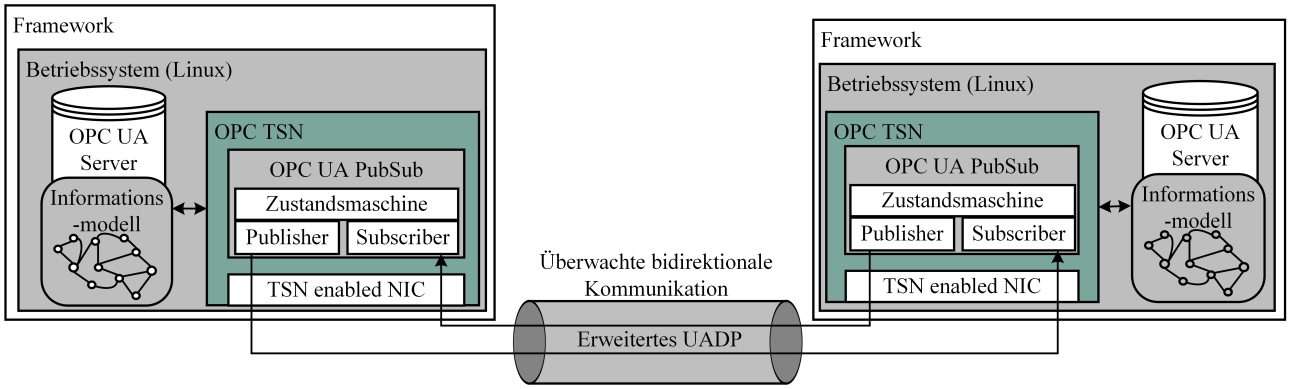


Abbildung 3.21: Tätigkeitsfelder für eine überwachte bidirektionale Verbindung.

Die Vorgehensweise im Rahmen dieser Arbeit ist in Abbildung 3.22 dargestellt und gliedert sich in folgende Schritte:

1. **Konzeption einer Evaluierungsplattform:** Ausgehend von den Entscheidungsmöglichkeiten zur Optimierung eines Linux-Betriebssystems im Stand der Technik, werden in Kapitel 4 zunächst einige grundlegende Entscheidungen für die Evaluierungsplattform getroffen und begründet. Des Weiteren beschreibt die Evaluierungsplattform alle nötigen OPC UA- und TSN-Komponenten für einen bidirektionalen Datenaustausch als auch deren Interaktion miteinander.

2. **Erstellung einer Leistungsmetrik:** Basierend auf der konzipierten Evaluierungsplattform wird in Kapitel 5, mit Hilfe von verbreiteten Benchmarking-Werkzeugen, die allgemeine Echtzeitfähigkeit der Evaluierungsplattform analysiert. Trotz hoher Relevanz von OPC UA PubSub gibt es bis dato wenig Literatur über Untersuchungen zur Leistung der Kombination aus OPC UA PubSub und TSN. Deshalb wird eine eigene experimentelle Methode entworfen, um eine Leistungsmetrik für OPC UA PubSub und TSN zu erstellen und die Einflüsse der Evaluierungsplattform auf die Prozesszeiten und Durchlaufzeiten zu ermitteln.

3. **Konzeption einer Überwachung für OPC UA PubSub:** OPC UA PubSub basiert auf einer entkoppelten Kommunikation und bietet bisher noch keine Überwachung des Datenaustauschs auf Seiten des Publishers. Deshalb wird in Kapitel 6 eine Überwachung des Datenaustauschs für OPC UA PubSub konzipiert. Dazu werden zunächst zwei Busprotokolle der Klasse 3 – PROFINET IRT und Sercos III – hinsichtlich ihrer Überwachung abstrahiert. Es werden alle Funktionen zur Überwachung des Datenaustauschs identifiziert und in einem weiteren Schritt generalisiert. Anhand der Generalisierung wird ein Konzept abgeleitet, das einen überwachten Datenaustausch, basierend auf OPC UA PubSub, zur Verfügung stellt.

4. **Realisierung der Überwachung des Datenaustauschs:** Abschließend wird in Kapitel 7 die Umsetzbarkeit des überwachten Datenaustauschs gezeigt. Dazu wird eine Achskopplung über eine C2C-Applikation umgesetzt. In Kapitel 8 schließen eine Zusammenfassung der Ergebnisse und ein Ausblick die Arbeit ab.

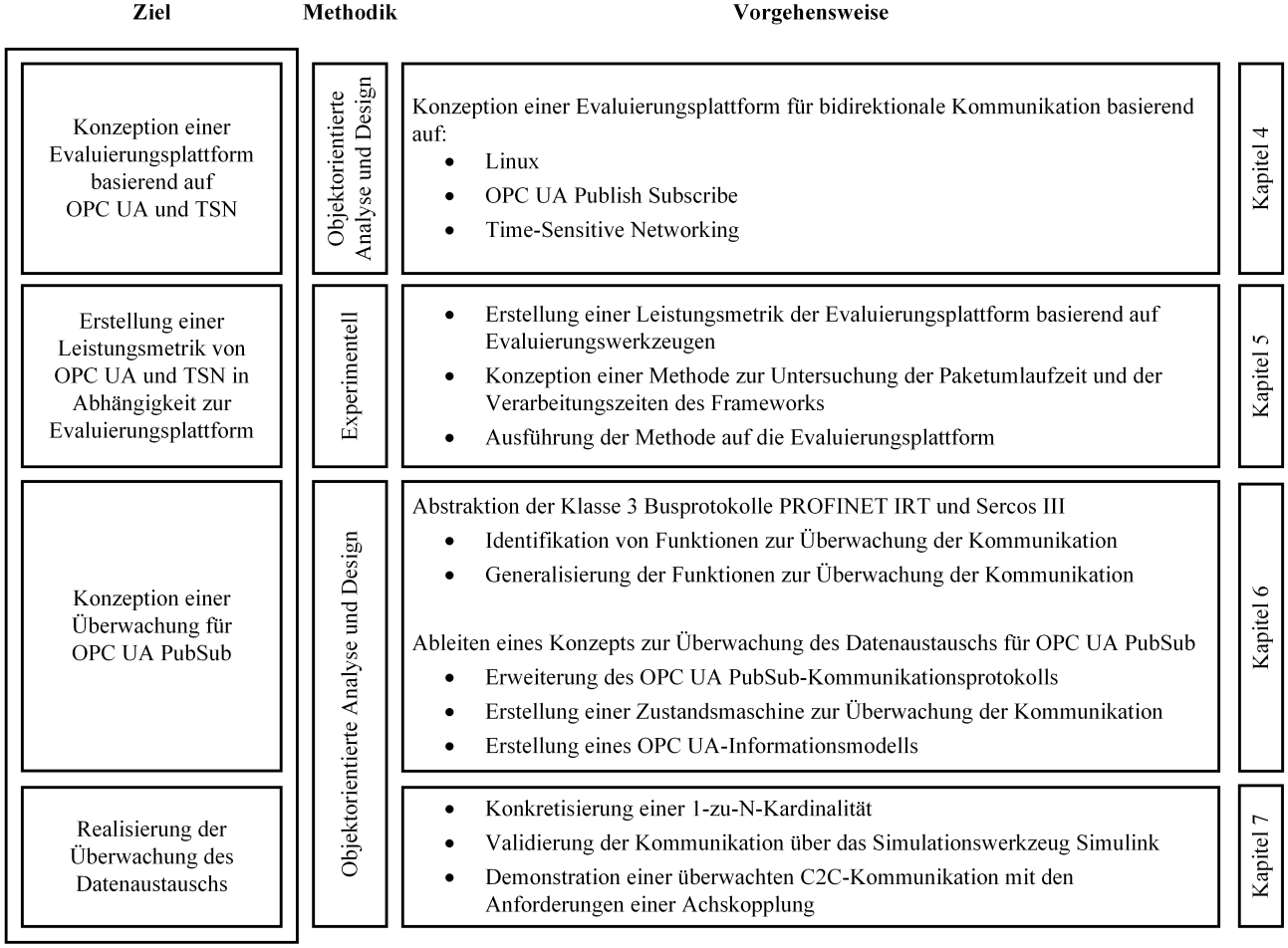


Abbildung 3.22: Zielsetzung, Methodik und Vorgehensweise der Arbeit.

4 Konzeption einer Evaluierungsplattform für einen bidirektionalen Datenaustausch

In diesem Kapitel wird zunächst die Grundlage für die nächsten Kapitel geschaffen, indem eine entsprechende Hardware für die Evaluierungsplattform ausgewählt wird. Die Auswahlmöglichkeiten sind sehr eingeschränkt, da zu diesem Zeitpunkt noch nicht viele TSN-fähige Lösungen auf dem Markt verfügbar sind. Das gilt zum einen für die Hardware als auch für die notwendigen Software-Komponenten, wie z. B. die TSN-Implementierungen oder die OPC UA PubSub-Komponenten. Es wurde das Xilinx Zynq UltraScale+ MPSoC[7] aus folgenden Gründen ausgewählt:

FPGA-basierte Kommunikationslösung: Xilinx bietet eine Field Programmable Gate Array (FPGA)-basierte Kommunikationslösung an, die einige der TSN-Funktionalitäten bietet.

Ethernet-Ports: Standardmäßig hat das Board nur einen Ethernet-Port. Über eine Einsteckkarte können aber noch weitere vier Ports hinzugefügt werden. Die Ports verfügen über einen Datendurchsatz von bis zu 10/100/1000 Mbps.

Switch-Funktionalität: Je nach Hardware-Konfiguration kann das Board auch als Endpoint oder Switched-Endpoint eingesetzt werden. Wenn das Board als Switched-Endpoint eingesetzt wird, bietet es die Funktionalitäten eines Switches, wie beispielsweise die Weiterleitung von Datenverkehr an einen anderen Ethernet-Port. Das ist z. B. für einen Linienbetrieb auf der Feldebene wichtig, wenn sehr viele Geräte verwendet werden. Durch den Linienbetrieb kann zusätzliche Netzwerkinfrastruktur, beispielsweise weitere Switches, eingespart werden.

[7] `https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf/`

In Abbildung 4.1 ist das Grundgerüst des Designs dargestellt. Es besteht aus einem Processing System und einer Programmable Logic. Das Processing System nutzt eine Application Processing Unit (APU) mit einem Quad Core Arm Cortex-A53. Die APU besitzt vier CPU-Kerne. Die APU betreibt ein Linux-Betriebssystem mit der Kernel Version 4.14. Linux teilt sich in einen User Space und Kernel Space auf. Im User Space sind mehrere Werkzeuge zur Synchronisation der Zeit und zur Evaluierung des Systems vorhanden. Außerdem führt die APU im User Space jene Applikation aus, welche die OPC UA-Funktionalitäten eines Servers, Publishers und Subscribers zur Verfügung stellt.

Für die allgemeine OPC UA-Funktionalität wird ein kommerzielles SDK verwendet. Auf Grund fehlender OPC UA PubSub-SDKs, müssen jedoch die Publisher und Subscriber selbst implementiert werden. Die Publisher und Subscriber nutzen TSN-Talker und TSN-Listener zum Senden und Empfangen der Nachrichten. Die TSN-Talker und TSN-Listener können über ein Standard-Linux Application Programming Interface (API) auf den Kernel Space und dementsprechend auf den Netzwerk-Stack zugreifen.

Die TSN-Talker und TSN-Listener Applikationen basieren auf dem Xilinx IP-Core Evaluationskit. Das Processing System ist über ein Interconnect, als logische Verbindung, mit der Programmable Logic verbunden. Standardmäßig sind im TSN-Endpoint drei verschiedene TSN-Streams – Scheduled, Reserved und Best-Effort – konfiguriert. Die drei Direct Memory Access (DMA)s ermöglichen den Zugriff auf die Queues der Streams. Am TSN-Endpoint sind zwei von den vier zusätzlichen Ethernet-Ports angeschlossen.

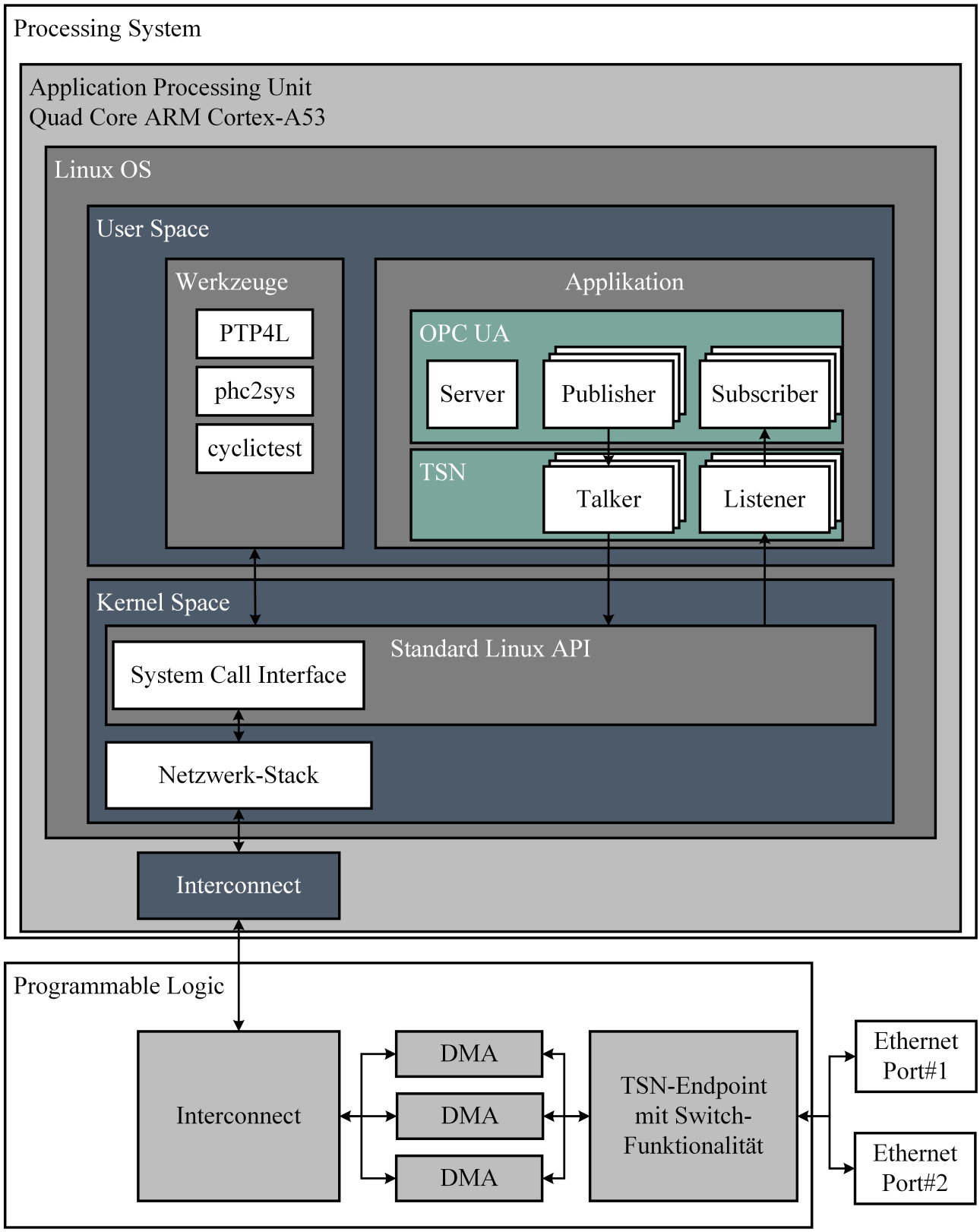


Abbildung 4.1: Design der Evaluierungsplattform.

5 Erstellung einer Leistungsmetrik von OPC UA PubSub und TSN

Zunächst wird die Evaluierungsplattform aus Kapitel 4 auf ihre Echtzeitfähigkeit untersucht. Dazu wird das öffentlich zugängliche Werkzeuge Cyclictest verwendet. Danach wird eine eigene Methode konzipiert und auf die Evaluierungsplattform angewendet. Das Ziel dieser Methode ist es, die Verarbeitungszeiten von OPC UA PubSub und die Paketumlaufzeiten in Abhängigkeit des Systemdesigns zu messen.

5.1 Anwendung des Evaluierungswerkzeugs Cyclictest

Im Kapitel 3 Stand der Technik wird eine Literaturanalyse zur Auswahl geeigneter Evaluierungswerkzeuge durchgeführt. Es wird das Werkzeug Cyclictest zur Messung der Wake-Up-Latenz verwendet.

Dazu startet das Werkzeug Cyclictest über einen Master-Thread mit der Scheduling Policy SCHED_OTHER mehrere definierte Echtzeit-Threads mit Scheduling Policy SCHED_FIFO. Die Echtzeit-Threads werden zyklisch über einen Zeitgeber aufgeweckt. Die Differenz zwischen dem programmierten Zeitpunkt und dem tatsächlichen Zeitpunkt, an dem der Thread aufwacht, wird als Wake-Up-Latenz zurückgegeben.

Folgender Befehl wird dafür ausgeführt:

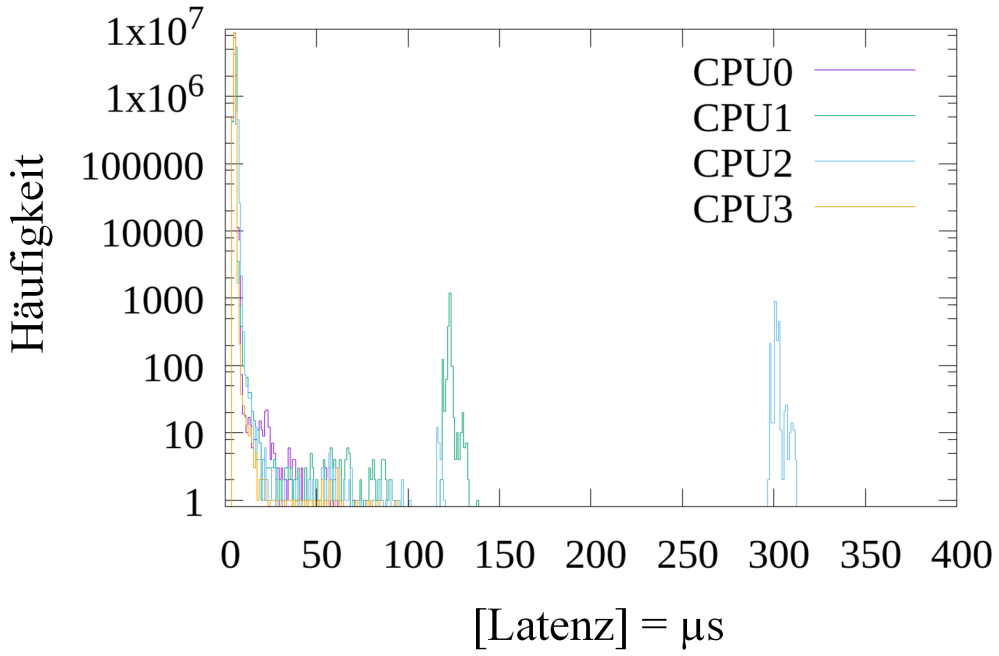
```
$ cyclictest -l10000000 -m -Sp90 -i200 -h400 -q

Mit:
-l: Anzahl der Zyklen
-m: Mlockall
-S: Standard-Testing
-p: Scheduler-Priorität
-i: Intervalldauer
-h: Erstellung eines Histogramms
-q: Zusammenfassung am Ende des Tests
```

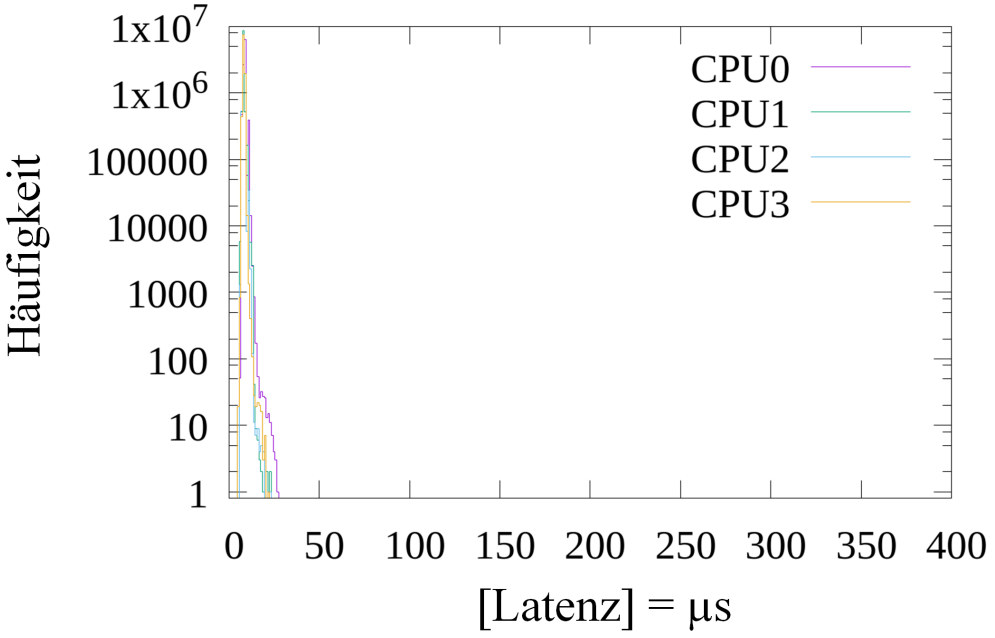
Die Tabelle 5.1 listet alle gemessenen Min.- und Max.-Werte, sowie die Avg.-Werte der einzelnen CPU-Kerne auf. In Abbildung 5.1 sind die zwei Histogramme der Messreihen mit und ohne Linux Kernel PREEMPT_RT-Patch abgebildet. Zwei wichtige Trends sind hier zu erkennen. Erstens verschlechtert sich die Wake-Up-Durchschnittslatenz durch die Verwendung des PREEMPT_RT-Patches. Zweitens verbessert sich die maximal aufgetretene Wake-Up-Latenz. Durch eine geringere Durchschnittslatenz lässt sich ein höherer Durchsatz erreichen. Das ist meistens das Ziel von GPOS-Betriebssystemen. Durch den PREEMPT_RT-Patch verbessert sich durch die niedrigere maximale Latenz das Echtzeitverhalten auf Kosten des Durchsatzes.

Tabelle 5.1: Messergebnisse des Cyclictests.

	Standard-Kernel				PREEMPT_RT Patch			
CPU Kern	1	2	3	4	1	2	3	4
Min. [µs]	4	4	4	4	6	6	6	5
Avg. [µs]	5	5	5	5	8	8	8	8
Max. [µs]	72	138	312	94	27	23	23	22



(a) *Standard Linux-Kernel.*



(b) *Kernel mit PREEMPT_RT-Patch.*

Abbildung 5.1: Histogramme der Ergebnisse des Cyclictests.

5.2 Konzeption einer Methode zur Untersuchung der Evaluierungsplattform

Wie im Stand der Technik gezeigt, gibt es mehrere Möglichkeiten die Echtzeitfähigkeit von Linux zu verbessern. Diese Möglichkeiten werden nun als Einflussgrößen in der Methode verwendet. In Abbildung 5.2 sind zum einen alle Einflussgrößen (z. B. UADP-Konfiguration und Compiler-Optimierung) und zum anderen alle erhobenen Daten (z. B. Verarbeitungszeiten) dargestellt.

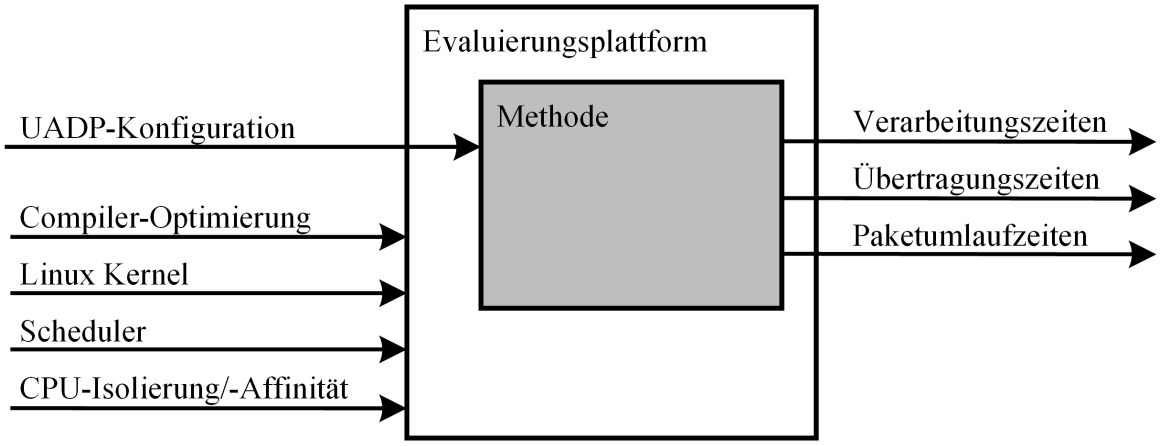


Abbildung 5.2: Einflussgrößen des Experiments.

Bei der Compiler-Optimierung handelt es sich um Compiler-Flaggen. Über diese Optimierung wird die Verarbeitungszeit positiv beeinflusst. Es gibt sehr viele unterschiedliche Optimierungen. Oftmals wird eine Sammlung von Optimierungen in sogenannte Levels eingestuft. In dieser Arbeit wird zwischen Level 0 und Level 2 unterschieden. Level 0 bedeutet, dass die Compiler-Optimierung deaktiviert ist. Die Compiler-Optimierung Level 2 ist eine Sammlung von Standard-Optimierungen.

Beim Linux Kernel als eine Einflussgröße wird zwischen zwei Konfigurationen ausgewählt. Zur Auswahl stehen ein Standard-Linux Kernel mit der Version 4.14 oder der gleiche Linux Kernel mit einem angewendeten PREEMPT_RT-Patch. Beim PREEMPT_RT-Patch wird die Konfiguration *Fully Preemptible Kernel* ausgewählt.

Bei der Wahl der Scheduler wird entweder der Default-Scheduler SCHED_OTHER oder der Real-Time (RT)-Scheduler SCHED_FIFO ausgewählt. Bei der Messreihe mit dem Scheduler SCHED_FIFO wird der Applikation eine Priorität von 90 zugewiesen.

Hinsichtlich CPU-Affinität werden zwei Konfigurationen getestet. Bei der ersten Konfiguration wird kein CPU-Kern isoliert und die CPU-Affinität für die Applikation wird nicht explizit

gesetzt. Infolgedessen kann die Applikation auf allen vier CPU-Kernen ausgeführt werden. Bei der zweiten Konfiguration wird der vierte CPU-Kern isoliert und die CPU-Affinität so gesetzt, dass der vierte CPU-Kern ausschließlich für die Applikation verwendet wird.

Die Grundidee der Methode ist, eine Nachricht zwischen zwei Geräten hin- und zurückzusenden und dabei mehrere Zeitstempel aufzuzeichnen. Dadurch können Rückschlüsse über die Verarbeitungs- und Übertragungszeiten gezogen werden. In Abbildung 5.3 ist der Messvorgang detailliert dargestellt.

Damit die Zeitstempel zwischen den Boards miteinander vergleichbar sind, werden die Precision Time Protocol (PTP)-Hardware Clocks miteinander synchronisiert. Dazu wird ein Open-Source-Werkzeug namens PTP4L[8] verwendet. PTP4L ist eine Implementierung des IEEE-Standards 1588. PTP4l unterstützt außerdem IEEE 802.1AS-2011 als Endstation, Boundary Clock, Ordinary Clock und Transition Clock.

Insgesamt werden 17 Zeitstempel T_1 bis T_{17} aufgezeichnet. Die Zeitstempel im User Space werden dabei über die Funktion clock_gettime() aufgezeichnet. Die Zeitstempel aus dem Kernel Space (T_4, T_5, T_{13} und T_{14}) werden hingegen automatisch beim Bearbeiten des SFDs im Rahmen des Sende- und Empfangsvorgangs aufgezeichnet.

Beim Initialisieren liest der Publisher zunächst eine Konfigurationsdatei aus und enkodiert, entsprechend der Konfiguration, eine NetworkMessage. Da das Enkodieren nur in der Initialisierungsphase stattfindet, ist es in Abbildung 5.3 nicht dargestellt. Nichtsdestotrotz wird die Dauer der Enkodierung ebenfalls gemessen. Über die Konfigurationsdatei kann z. B. Einfluss auf die Größe der NetworkMessage oder des Header-Layouts genommen werden. Sobald die Initialisierungsphase abgeschlossen ist, kann die Messung der Paketumlaufzeit gestartet werden.

Die Funktion *Update()* aktualisiert alle dynamischen Felder der NetworkMessage. Dynamische Felder sind z. B. Sequenznummern und bisher aufgezeichnete Zeitstempel. Der Publisher kopiert die aufgezeichneten Zeitstempel T_1 bis T_3 in die NetworkMessage und übergibt die NetworkMessage an die Funktion *Send()*.

Die Funktion *Send()* kopiert den vorbereiteten Frame in die Qbv-Queue. Sobald das Qbv-Gate offen ist, wird der Frame an das Board 2 gesendet. Der Publisher nutzt dafür die TSN-Implemen-tierung von Xilinx.

Die Funktion *Receive()* empfängt den Frame und gibt die Payload, was hier der NetworkMessage entspricht, an die *Decode()*-Funktion weiter.

[8] `https://linux.die.net/man/8/ptp4l/`

Die Funktion *Decode()* dekodiert die NetworkMessage. Nach dem Dekodieren liegen alle in der NetworkMessage enthaltenen Zeitstempel vor, die dann an die nächste Funktion übergeben werden.

Die Funktionen *CopyToBuffer()* und *CopyFromBuffer()* tauschen die bisher aufgezeichneten Zeitstempel zwischen der Publisher- und Subscriber-Applikation aus. Bei diesem Buffer handelt es sich um einen Ring-Buffer. Die Funktion *CopyFromBuffer()* ist außerdem eine blockierende Funktion, die auf ein neues Element im Buffer wartet. Das heißt, der Publisher von Board 2 sendet nicht wie der Publisher von Board 1 zyklisch seine Frames, sondern nur dann, wenn er über den Buffer ein Event für ein neu verfügbares Buffer-Element bekommt.

Beim Erreichen des Zustandes *Ende* werden alle aufgezeichneten Zeitstempel des Zyklus gespeichert. Sobald die Messung abgeschlossen ist, wird eine Textdatei mit allen Zeitstempeln erstellt, die zur weiteren Auswertung der Daten verwendet werden kann.

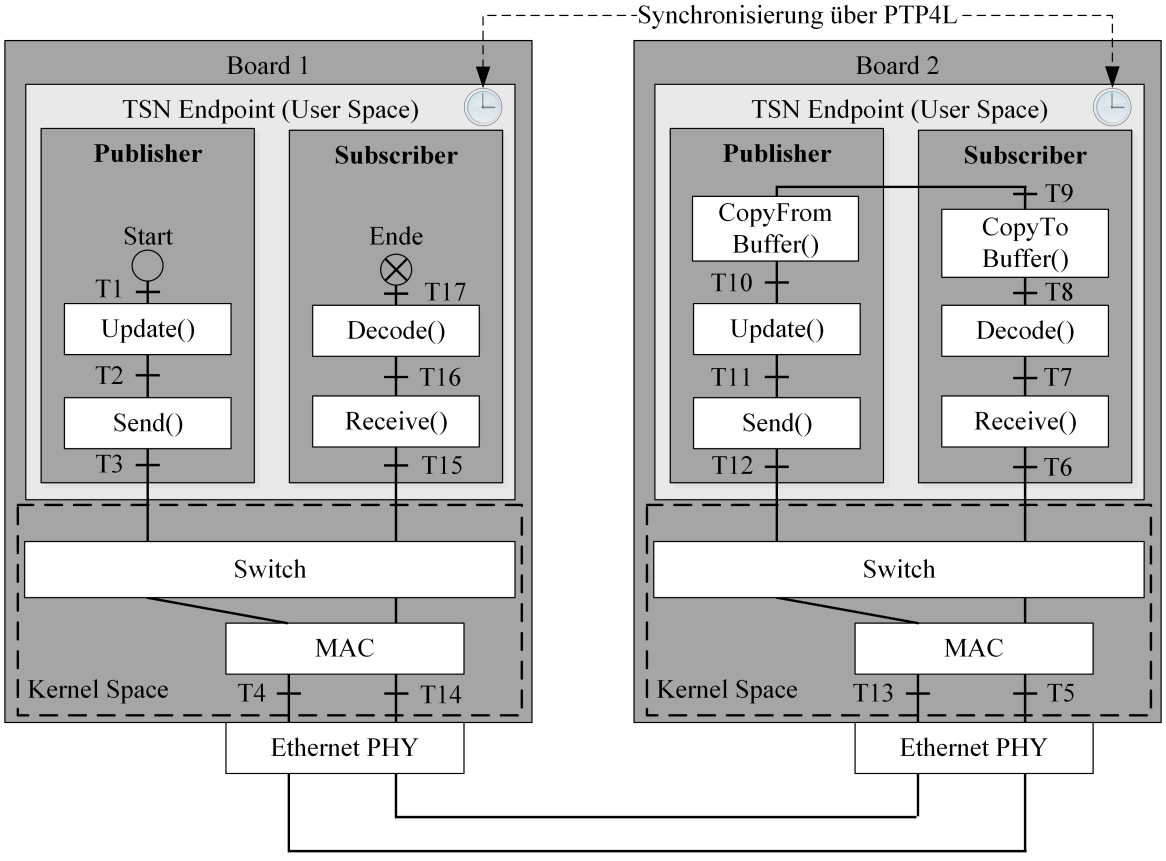


Abbildung 5.3: Design der konzipierten Methode (Eckhardt et al. 2019a; Eckhardt et al. 2019b).

5.3 Anwendung der Methode

Die zuvor konzipierte Methode wird nachfolgend auf die Evaluierungsplattform angewendet.

5.3.1 Messreihen und allgemeine Konfiguration

Die Messreihen setzen sich aus der unterschiedlichen Kombination aus Compiler-Optimierung, Linux Kernel, CPU-Isolation und UADP-Layout zusammen. In der ersten Serie mit den Messreihen 1 bis 4 ist ein CPU-Kern für die Applikation isoliert und die UADP-Größe konstant. Es werden ausschließlich die Compiler-Optimierung und der Linux Kernel variiert. Die zweite Serie mit den Messreihen 5 bis 8 ist identisch mit der ersten Serie dieses Mal jedoch mit keinem isolierten CPU-Kern für die Applikation. Anhand der Ergebnisse der ersten beiden Messreihen kann eine optimale Konfiguration der Compiler-Optimierung, der Linux Kernel und der CPU-Kern Isolierung ausgewählt werden. In der dritten Serie mit den Messreihen 9 bis 13 soll diese optimale Konfiguration verwendet werden, um die zuvor konstant gehaltene UADP-Größe zu untersuchen. Deshalb bleiben in der dritten Messreihe die Compiler-Optimierung, der Linux Kernel und die CPU-Kern-Isolierung konstant und die UADP-Größe variable. In der Tabelle 5.2 sind alle Messreihen des Experiments zusammengefasst.

Tabelle 5.2: Messreihen des Experiments.

Messreihe Nr.	Compiler-Optimierung	Linux Kernel	Isolierung	UADP-Größe
1	0	Standard	Ja	1500
2	2	Standard	Ja	1500
3	0	RT	Ja	1500
4	2	RT	Ja	1500
5	0	Standard	Nein	1500
6	2	Standard	Nein	1500
7	0	RT	Nein	1500
8	2	RT	Nein	1500
9	2	RT	Nein	250
10	2	RT	Nein	400
11	2	RT	Nein	800
12	2	RT	Nein	1200
13	2	RT	Nein	1500

5.3.1.1 Verwendetes UADP Header Layout

In Abbildung 5.4 ist das verwendete UADP Header Layout dargestellt. Das Layout orientiert sich dabei an dem statischen UADP Header Layout aus Kapitel 3.4.2.4. Die erfassten Zeitstempel T_1 bis T_{17} werden als uint64 Datentyp in die DataSetFields geschrieben. Die daraus resultierende minimale Größe der NetworkMessage ist aufgrund der 17 enthaltenen Zeitstempel 250 Bytes lang. Um einen Frame mit einer Länge von 1500 Bytes zu versenden, werden weitere DataSetFields vom Typ uint64 angehängt.

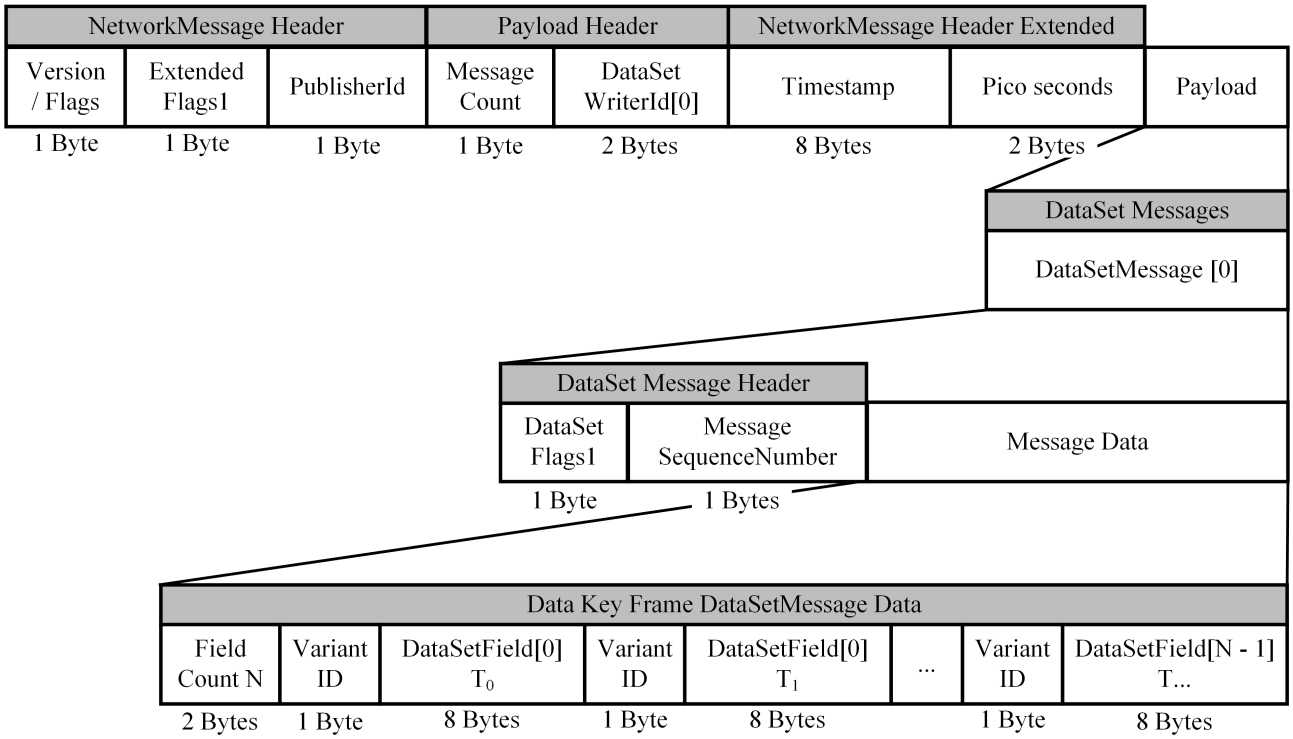


Abbildung 5.4: Verwendetes UADP Header Layout.

5.3.1.2 Verwendete Time-Sensitive Networking Konfiguration

Zum Senden der Frames wird die TSN-Implementierung von Xilinx verwendet. Die Implementierung verfügt über eine Gate Control List nach IEEE 802.1 Qbv zur Konfiguration der Öffnungszeiten der Qbv-Gates. Für eine Paketumlaufzeitmessung ist es jedoch fatal, das verwendete Gate für eine gewisse Zeit zu schließen, da der Ablauf der Messung eventgesteuert und nicht zeitgesteuert ist. Deshalb wird das Gate durchgängig geöffnet, um keine zusätzliche Verzögerung der Nachricht zu verursachen. Um die PTP-Uhren zu synchronisieren, wird im Hintergrund das Werkzeug PTP4L ausgeführt. Erst wenn die Genauigkeit der Synchronisierung im unteren zweistelligen Bereich < 20 ns ist, startet die Paketumlaufzeitmessung.

5.3.2 Ergebnisse

In Abbildung 5.5 sind die Minima, die Mittelwerte und die Maxima der Paketumlaufzeiten der Messreihen 1 bis 8 dargestellt. Es lassen sich mehrere Trends in Abbildung 5.5 erkennen. Erstens, die Compiler-Optimierung hat in allen Fällen einen positiven Einfluss auf die Reduzierung der Paketumlaufzeiten. Durch die Compiler-Optimierung kann das Minimum, der Durchschnitt und das Maximum in allen Fällen reduziert werden. Zweitens, Linux-Betriebssysteme ohne PREEMPT_RT-Patch profitieren von einer Isolierung der CPU-Kerne, indem das Maximum reduziert wird. Die durchschnittliche Paketumlaufzeit nimmt aber geringfügig zu. Drittens, Linux-Betriebssysteme mit PREEMPT_RT-Patch profitieren nicht von einer Isolierung der CPU-Kerne.

Die Messreihe mit dem niedrigsten Minimum und geringsten Durchschnitt ist die Messreihe 6 – keine Isolierung, kein PREEMPT_RT-Patch aber Compiler-Optimierung – mit 118 µs und 128 µs. Das niedrigste Maximum hat die Messreihe 8 – keine Isolierung, PREEMPT_RT-Patch und Compiler-Optimierung – mit 252 µs.

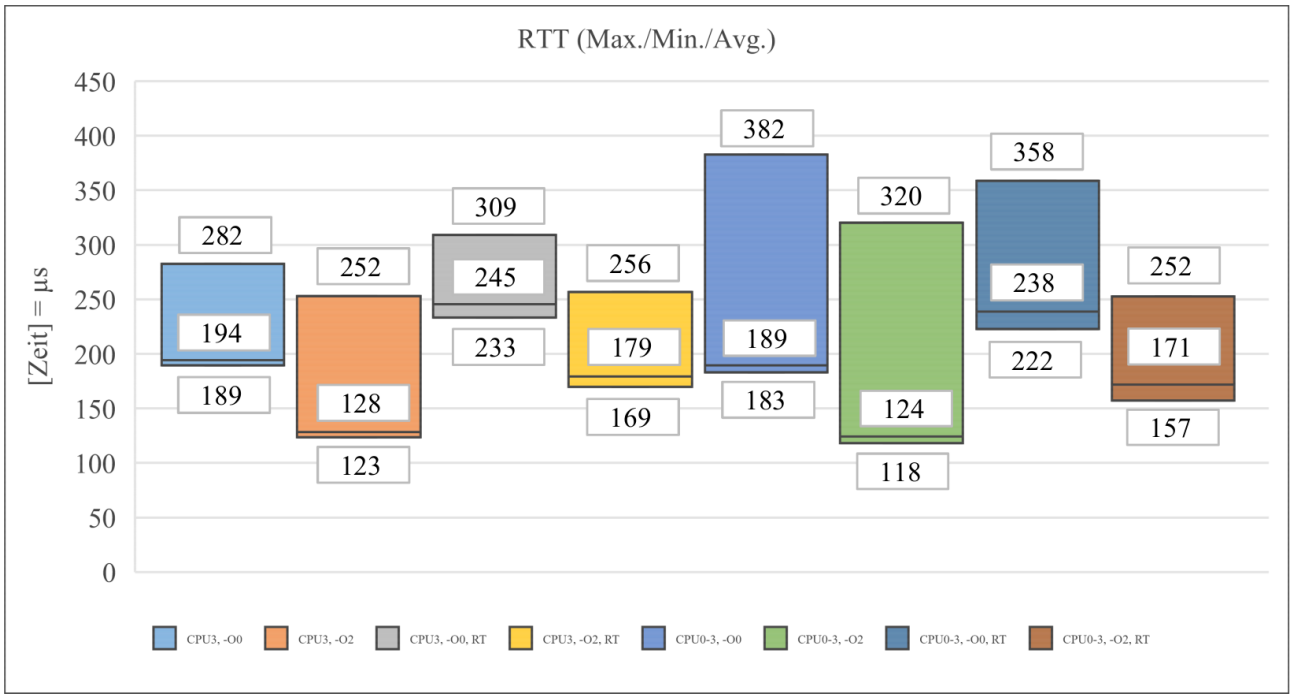


Abbildung 5.5: Min./Avg./Max.-Verteilung der Messreihen 1 - 8.

Als Nächstes wird in Abbildung 5.6 die Verteilung der Paketumlaufzeiten für die Messreihen mit Compiler-Optimierung näher betrachtet. Es ist von Interesse, ob es sich bei den Maxima um Ausreißer handelt. Dies ist jedoch nicht der Fall, da alle Maxima einen unmittelbaren Nachbarn mit einem Abstand kleiner 15 µs haben.

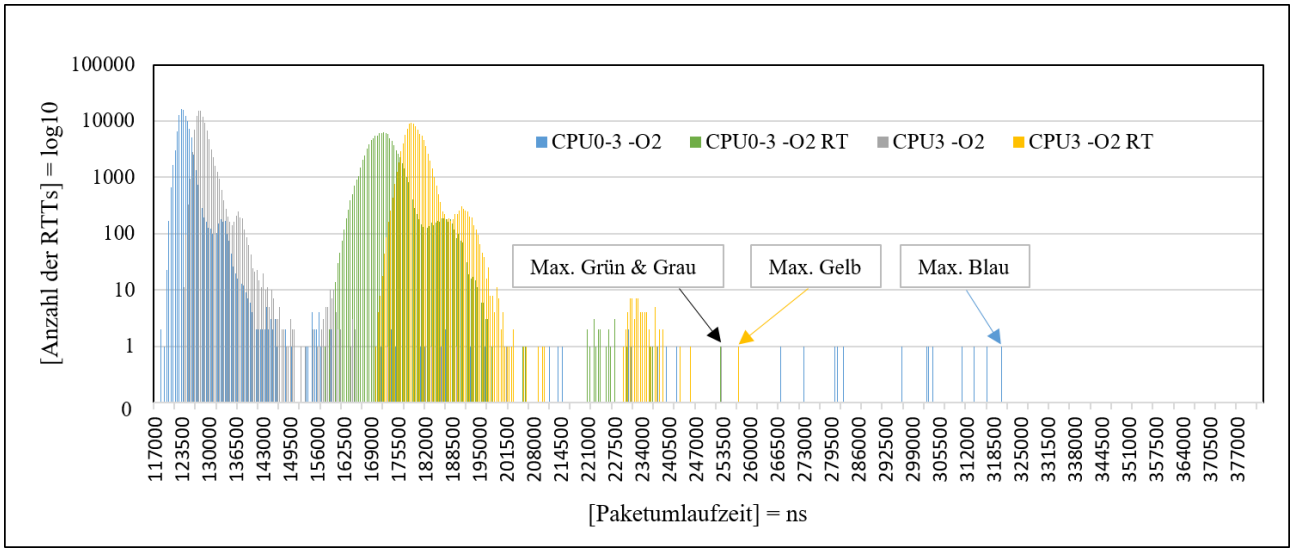
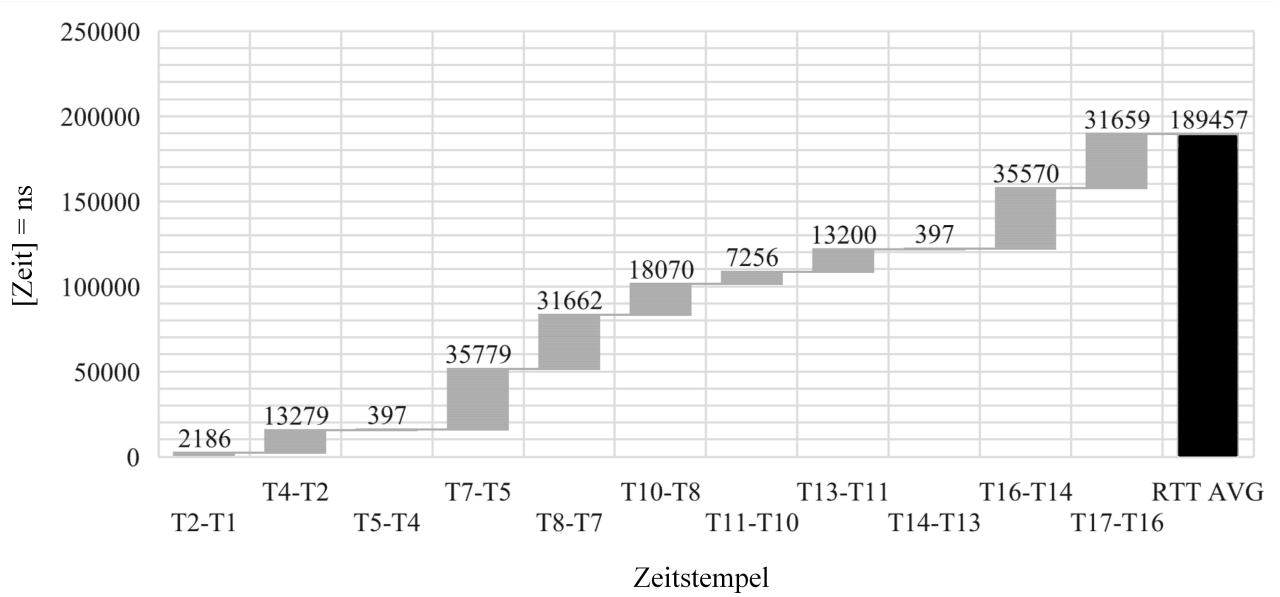


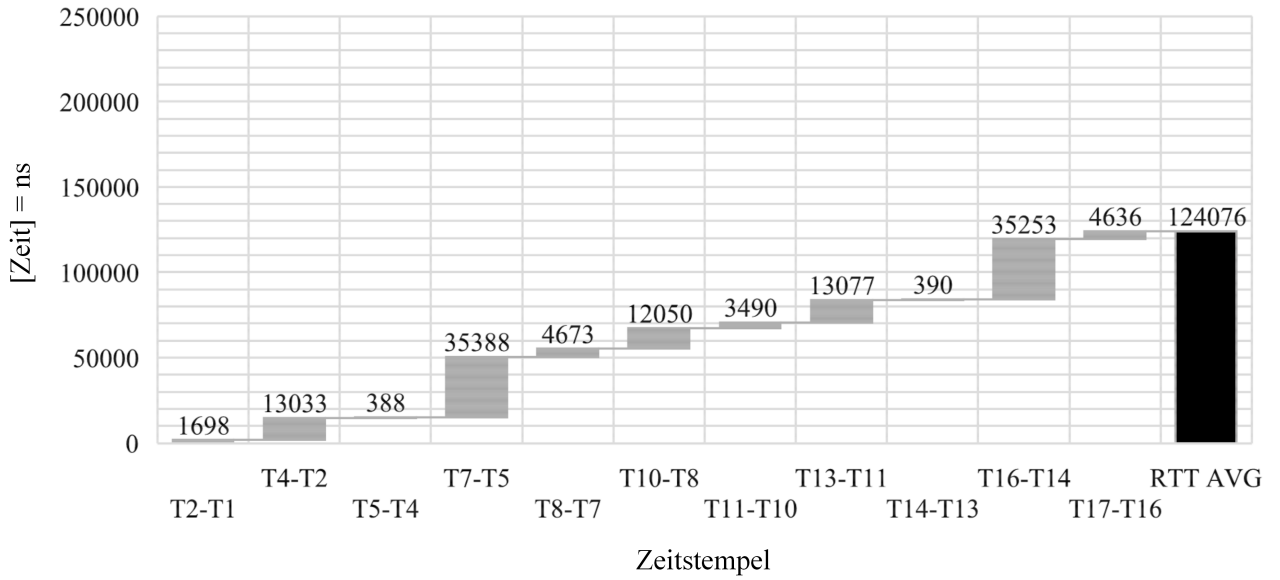
Abbildung 5.6: Histogramm der Messreihen – 2, 4, 6 und 8 – ohne CPU-Kern-Isolierung.

In den folgenden Abbildungen 5.7(a) bis 5.8(b) wird die Paketumlaufzeit in ihre Bestandteile aufgeteilt. Beim Vergleich der Abbildung 5.7(a) mit der Abbildung 5.7(b) und der Abbildung 5.8(a) mit der Abbildung 5.8(b) wird deutlich, dass die Compiler-Optimierung vor allem die Dekodierung, T_8 - T_7 und T_{17} - T_{16}, verbessert. Ein weiterer Vergleich zwischen der Abbildung 5.7(a) und der Abbildung 5.8(a), sowie der Abbildung 5.7(b) und der Abbildung 5.8(b) zeigt noch deutlicher die Auswirkungen des PREEMPT_RT-Patches. Es ist klar ersichtlich, dass die Verarbeitungszeiten allgemein größer werden. Infolgedessen verlängern sich auch die durchschnittlichen Paketumlaufzeiten. Es ist also nicht ein spezieller Teil der Verarbeitung für die Verlängerung der Paketumlaufzeiten verantwortlich, sondern vielmehr eine insgesamt langsamere Verarbeitung. Betrachtet man die Abbildung 5.7(b) und Abbildung 5.8(b) so sieht man, dass die Blöcke des Wasserfalldiagramms am größten bei den Deltas der Zeitstempel T_7 - T_5 und T_{16} - T_{14} sind. Zwei Dinge passieren hauptsächlich zwischen dem Zeitpunkt T_5 und dem Zeitpunkt T_7 und zwar ein Kontextwechsel und die komplette Verarbeitung der 1500 Bytes im Network Interface Card (NIC). Für die Verarbeitung von 1500 Bytes benötigt die verwendete 1 GBit/s NIC theoretisch 12 µs. Das Delta der Zeitstempel von T_7 - T_5 aus der Abbildung 5.7(b) beträgt gerundet 35 µs. Wird nun der theoretische Wert von 12 µs subtrahiert, bleiben 23 µs für den Kontextwechsel übrig. Das Delta der Zeitstempel von T_7 - T_5 aus der Abbildung 5.8(b) liegt mit ungefähr 50 µs höher als bei der Abbildung 5.8(b). Wenn der theoretische Wert von 12 µs davon subtrahiert wird, beträgt die Dauer des Kontextwechsel 38 µs.

Es ist das Ziel über die Messreihen 9 bis 13 die Paketumlaufzeit in Abhängigkeit der Frame-Größe zu bestimmen. In Abbildung 5.9 sind der maximale, der durchschnittliche und der mini-

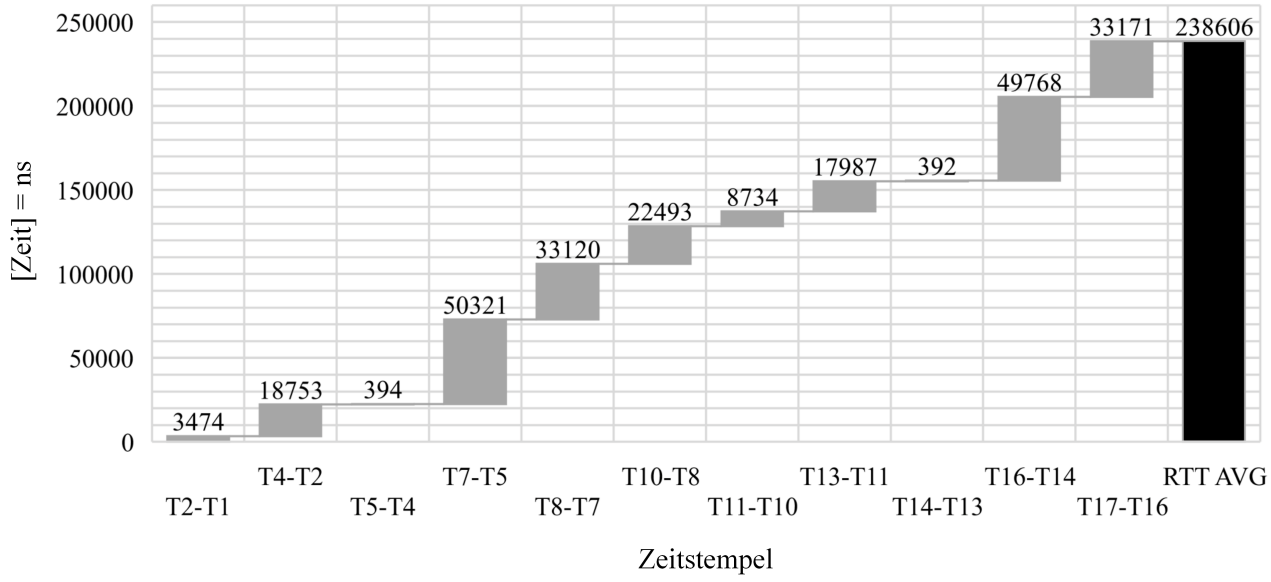


(a) *Standard Kernel, -O0.*

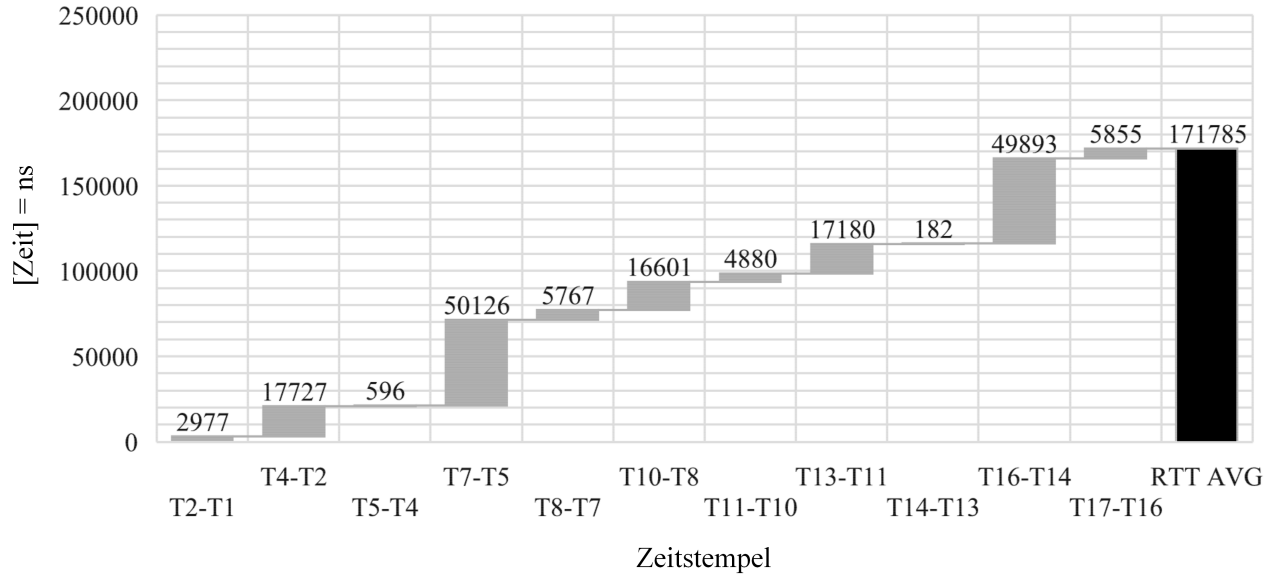


(b) *Standard Kernel, -O2.*

Abbildung 5.7: Durchschnittliche Paketumlaufzeiten ohne PREEMPT_RT-Patch.



(a) *RT-Patch, -O0.*



(b) *RT-Patch, -O2.*

Abbildung 5.8: Durchschnittliche Paketumlaufzeiten mit PREEMPT_RT-Patch.

male Verlauf der Paketumlaufzeit dargestellt. Es wird deutlich, dass die Verläufe im gemessenen Bereich zwischen 250 Bytes und 1500 Bytes linear steigen.

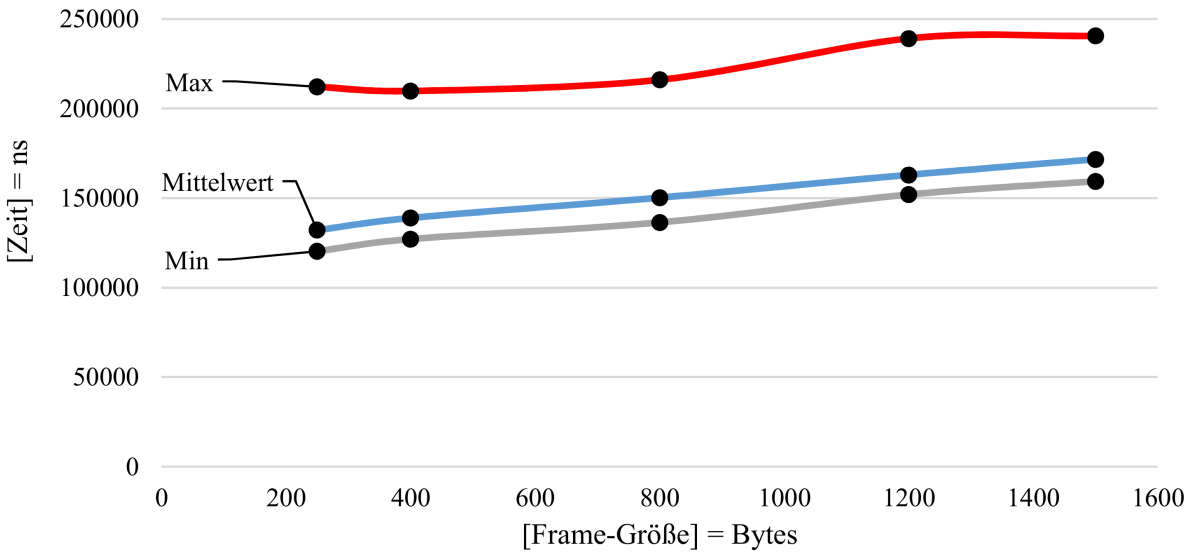


Abbildung 5.9: Paketumlaufzeit in Abhängigkeit der Frame-Größe.

Außerdem wird die Dauer der Enkodierung und Dekodierung in Abhängigkeit der Frame-Größe bestimmt. In Abbildung 5.10 sind die Verläufe der Messreihen 9 bis 13 dargestellt. Die Enkodierung der Nachricht dauert prinzipiell länger als die Dekodierung. Die Dauer der Dekodierung und Enkodierung verläuft linear über die Frame-Größe.

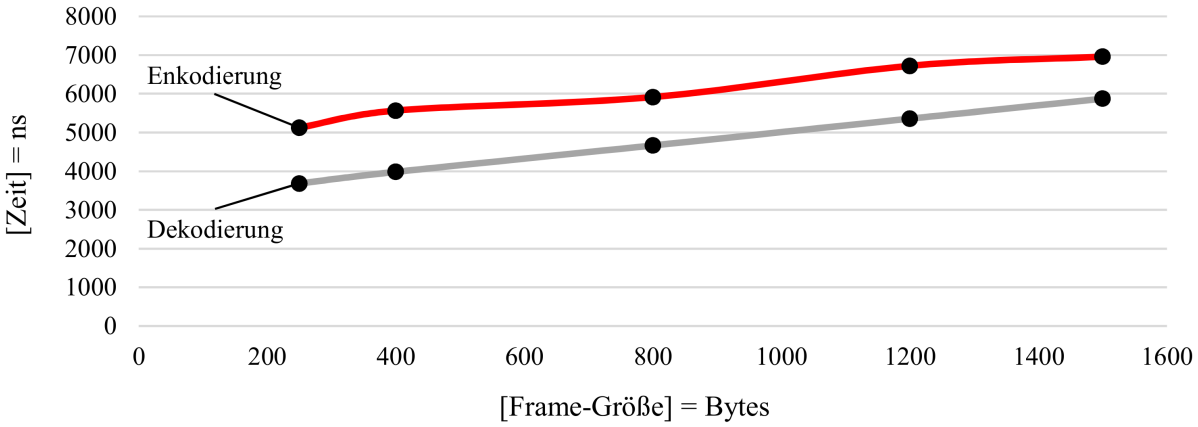


Abbildung 5.10: Dauer der En- und Dekodierung in Abhängigkeit der Frame-Größe.

5.4 Zusammenfassung

Die Messungen ergeben, dass mit dem Testfall 6 (keine CPU-Isolierung, Compiler-Optimierung -O2 und Standard-Kernel) die niedrigsten Min.- und Avg.-Werte mit 118 µs und 128 µs erreicht werden konnten.

Der niedrigste Max.-Wert mit 252 µs konnte anhand der Messreihe 8 (keine CPU-Isolierung, Compiler-Optimierung -O2 und RT-Kernel) erzielt werden.

Durch die Ergebnisse der Messreihen 9 bis 13 wird deutlich, dass die Paketumlaufzeit linear zur Frame-Größe im gemessenen Bereich steigt. Das Gleiche gilt für die Enkodierung und Dekodierung. Die Enkodierung dauert prinzipiell länger als die Dekodierung. Überraschend war es, dass die CPU-Isolierung für die Evaluierungsplattform mit PREEMPT_RT-Patch keine Verbesserung gebracht hat. Die Evaluierungsplattform mit Standard-Kernel konnte jedoch von der CPU-Isolierung profitieren.

Die erzielten Ergebnisse erfüllen die Anforderungen des C2C-Anwendungsfalls bei dem typischerweise eine Zykluszeit von bis zu 1 ms vorausgesetzt ist. Außerdem wird deutlich, dass die Enkodierung und Dekodierung nur einen geringen Teil der Paketumlaufzeit ausmachen. Die logische Schlussfolgerung ist, dass eine Erweiterung der UADP-Struktur, um die notwendigen Felder zur Überwachung des Datenaustauschs, vertretbar ist.

6 Entwicklung einer Überwachung des Datenaustauschs für OPC UA PubSub

Um eine geeignete Überwachung des Datenaustauschs bei OPC UA PubSub zu finden, werden zunächst in diesem Kapitel die Busprotokolle der Klasse 3, PROFINET IRT und Sercos III, untersucht und abstrahiert. Nachfolgend wird PROFINET IRT als PROFINET bezeichnet. Dabei wird genau herausgearbeitet, was bei den Busprotokollen überwacht wird und wie es umgesetzt wird. Basierend auf dieser Erkenntnis wird in einem nächsten Schritt eine Überwachung für das OPC UA PubSub-Protokoll abgeleitet. Es wurde PROFINET ausgewählt, da dieses Busprotokoll den größten Marktanteil hat. Sercos III wurde auf Grund der großen Verbreitung bei High-End-Motion-Applikationen ausgewählt.

6.1 Analyse und Abstraktion der Klasse 3 Busprotokolle

Im Kapitel 1 wurden bereits die Klasse 3 Busprotokolle kurz vorgestellt. Jetzt werden die zwei Busprotokolle – PROFINET und Sercos III – der Echtzeitklasse 3 näher betrachtet, miteinander verglichen und ihre Funktionen abstrahiert. Der Fokus liegt auf ihren Funktionen, die zur Überwachung der Kommunikation direkt eingesetzt werden oder Funktionen, die indirekt an der Überwachung der Kommunikation beteiligt sind. Um die Protokolle besser vergleichen zu können, wird die Deutsche Norm DIN 61158 verwendet (Norm DIN EN 61158-1; Norm DIN EN 61158-2; Norm DIN EN 61158-3-19; Norm DIN EN 61158-4-10; Norm DIN EN 61158-4-19; Norm DIN EN 61158-5-10; Norm DIN EN 61158-5-19; Norm DIN EN 61158-6-10; Norm DIN EN 61158-6-19).

6.1.1 Hochlaufphasen der Kommunikation

Die Hochlaufphasen der Kommunikation initialisieren, konfigurieren und bilden die Grundlage für die azyklische und zyklische Übertragung von Daten. PROFINET und Sercos III verfügen über unterschiedliche Hochlaufphasen, die in dem Unterkapitel 6.1.1.1 und Unterkapitel 6.1.1.2 erklärt werden.

6.1.1.1 Hochlaufphasen der PROFINET Geräteklassen Controller/Device

Bei PROFINET unterscheiden sich die Hochlaufphasen der Kommunikation in Abhängigkeit von den Geräteklassen Controller und Device. In Abbildung 6.1 sind die Hochlaufphasen der Context Management Controller sowie der Context Management Devices dargestellt, die im Nachfolgenden beschrieben werden.

Im Zustand POWER-ON werden der Controller und das Device gestartet und initialisiert. Bei den Zuständen *Wait for the IO device* und *Wait for a name and an IP address* werden alle Geräte gesucht und entsprechend mit IP-Adressen versehen.

Um die Kommunikationsparameter auszutauschen, bauen die Controller und Devices eine Verbindung über die Zustände *Wait for connect* und *Start connect* auf.

Nach den Zuständen *Wait for end of parametrization* und *End of parametrization* ist die Parametrierung abgeschlossen. Über die Zustände *Wait for application ready*, *Wait for update from user and create application ready* und *Wait for application ready from IO controller* wird darauf gewartet, dass der Controller und das Device bereit für die zyklische Kommunikation sind. Im gemeinsamen Zustand *Data* werden zyklisch Daten ausgetauscht.

Abstrakt betrachtet lässt sich der Hochlauf in drei Phasen unterteilen. Zunächst werden der Controller und das Device konfiguriert. Es findet azyklische Kommunikation in dieser Phase statt. Danach wird sichergestellt, dass die Applikation bereit ist, um dann im nächsten Schritt die zyklische Kommunikation zu starten.

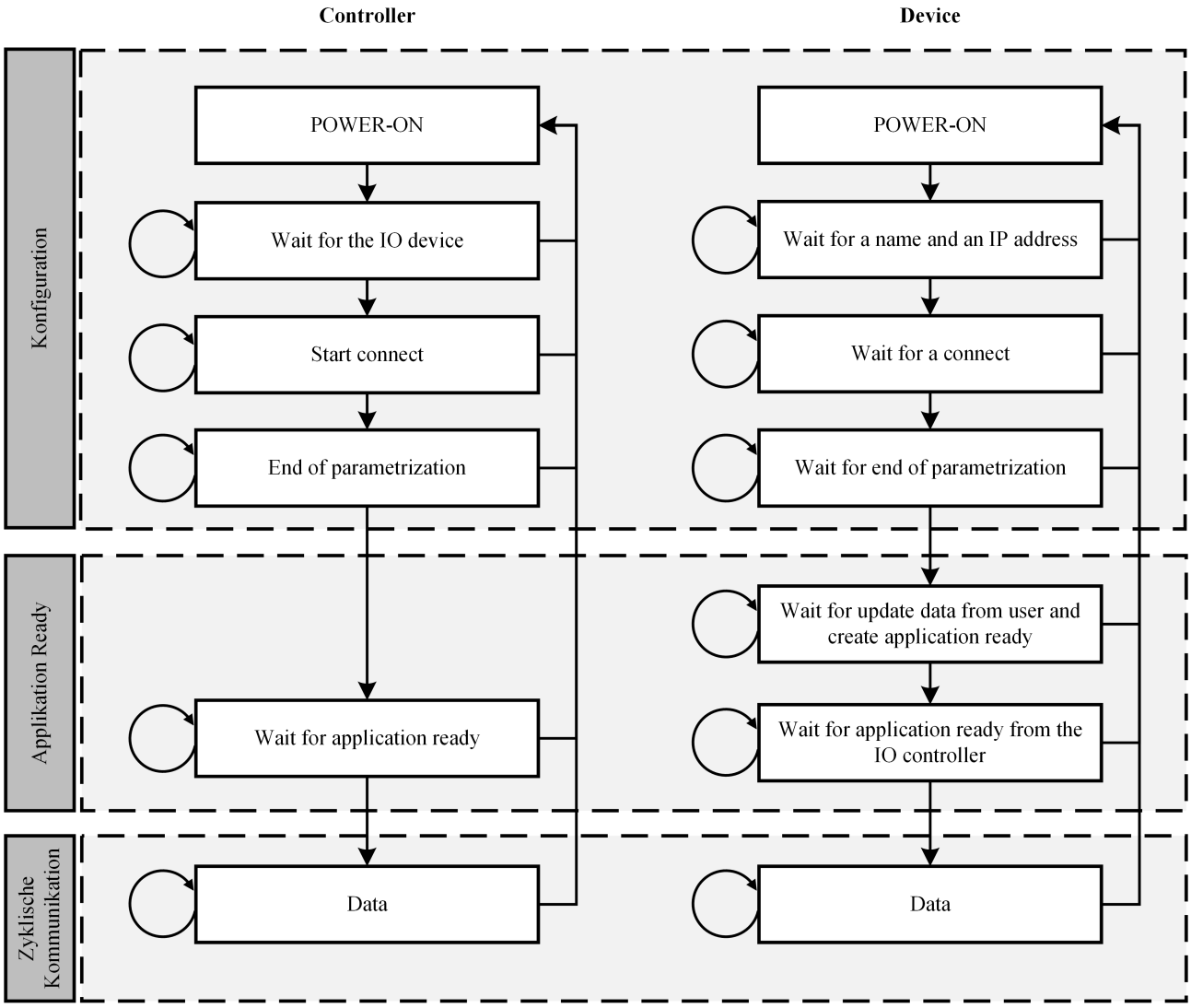


Abbildung 6.1: Hochlauf PROFINET

6.1.1.2 Hochlaufphasen von Sercos III

Sercos III unterteilt den Hochlauf in fünf Phasen 0 bis 4. Die Hochlaufphasen initialisieren und konfigurieren die azyklische und zyklische Kommunikation. In Phase 4 ist die Konfiguration abgeschlossen und das System ist zur Übertragung der Daten betriebsbereit. Die Phase 4 stellt die Basis für den Datenaustausch über das Producer/Consumer-Modell dar. Das heißt, die Ausgangssituation für den Datenaustausch ist eine bereits funktionsfähige zyklische Kommunikation, die bereits überwacht wird. Dadurch ist eine strikte logische Trennung zwischen der Übertragung der Daten und dem Datenaustausch, basierend auf dem Producer/Consumer-Modell, gewährleistet.

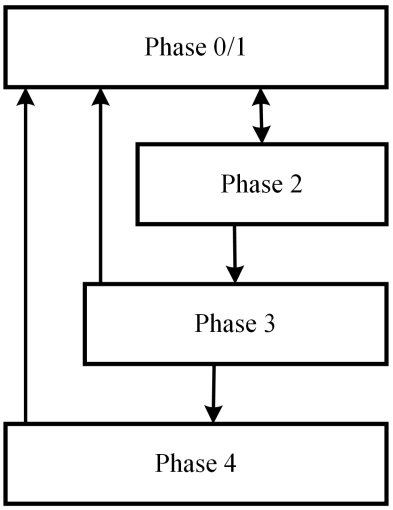


Abbildung 6.2: Hochlauf Sercos III.

Phase 0: In dieser Phase werden alle Sercos-Geräte erkannt.

Phase 1: Nach dem Erkennen der Sercos-Geräte wird zunächst die azyklische Kommunikation der Slaves konfiguriert.

Phase 2: Es wird die zyklische Kommunikation der Slaves konfiguriert. Dazu werden Parametereinstellungen in den Slaves über die azyklische Kommunikation vorgenommen.

Phase 3: In dieser Phase werden weitere Einstellungen an den Slaves vorgenommen und die zyklische Kommunikation findet bereits statt. Die übertragenen Daten sind jedoch noch ungültig.

Phase 4: In der letzten Phase ist die Konfiguration abgeschlossen und das Sercos III Netzwerk ist betriebsbereit.

6.1.1.3 Vergleich der Hochlaufphasen von Sercos III und PROFINET

In diesem Kapitel werden die Gemeinsamkeiten und die Unterschiede der Busprotokolle Sercos III und PROFINET beim Hochlauf der Kommunikation zusammengefasst. Abstrahiert betrachtet, konfigurieren beide Busprotokolle im ersten Schritt alle erkannten Geräte. Nach der Konfiguration wird bei PROFINET jedoch noch überprüft, ob die Applikation bei allen Geräten betriebsbereit ist. Dies ist bei Sercos III nicht der Fall und kann in gewissen Fällen

von Nachteil sein. Nachdem alle Geräte betriebsbereit sind, werden bei PROFINET im Zustand *Data* die zyklische Kommunikation ausgeführt und parallel die Zustandsmaschinen des Datenaustausches initialisiert und gestartet. Bei Sercos III werden die Producer/Consumer erst beim Erreichen der Phase 4 ausgeführt. Bei Sercos III ist hier eine striktere Trennung zwischen der Übertragung der Daten und dem Datenaustausch erkennbar, was als ein Vorteil angesehen wird.

6.1.2 Zustandsmaschinen

Um den Datenaustausch und dessen Überwachung der Busprotokolle im Ganzen zu verstehen, werden in diesem Kapitel zunächst die beteiligten Zustandsmaschinen identifiziert und beschrieben. Vor allem ist von Interesse, welche Zustandsmaschinen beteiligt sind und welche Mechanismen die Busprotokolle zur Überwachung verwenden.

6.1.2.1 PROFINET

In Abbildung 6.3 ist die Architektur der Zustandsmaschinen von PROFINET dargestellt. Die FAL Service Protocol Machine (FSPM) verteilt die Service-Anfragen des Anwenders an die entsprechenden untergeordneten Zustandsmaschinen und meldet die Antworten der Zustandsmaschinen an den Anwender zurück.

Die zugrundeliegende Architektur besteht aus vielen Zustandsmaschinen, die interagieren. Die Vielzahl an Zustandsmaschinen erklärt sich dadurch, dass bei PROFINET die Funktionalitäten in einzelne Zustandsmaschinen gegliedert werden.

Zur Vollständigkeit sind hier alle Zustandsmaschinen in der Abbildung dargestellt. Nachfolgend sind aber nur die rot umrandeten Zustandsmaschinen Context Management Protocol Machine Controller (CMCTL), Context Management Client Protocol Machine (CTLSM), Context Management Input Oputput Protocol Machine Controller (CTLIO), Provider Protocol Machine (PPM) und Consumer Protocol Machine (CPM) im Detail beschrieben und erklärt.

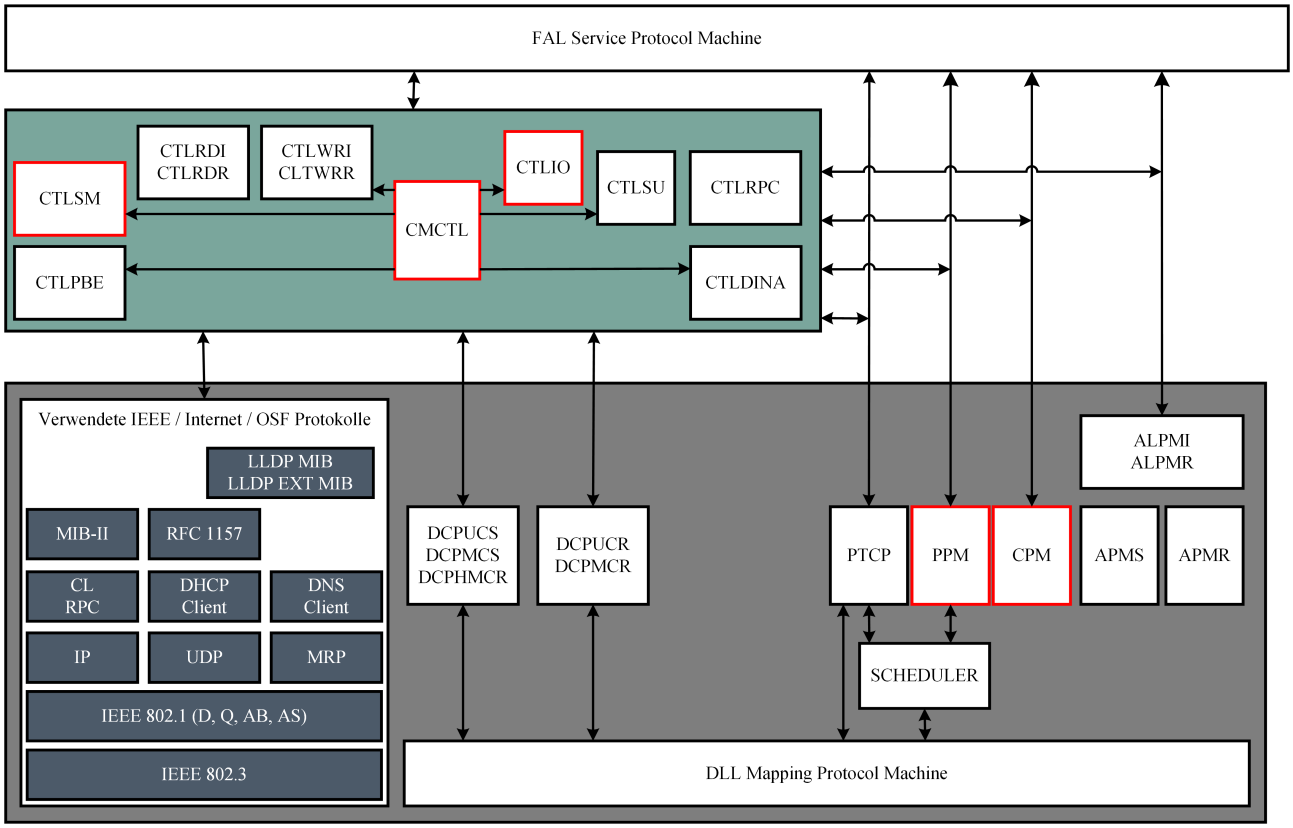
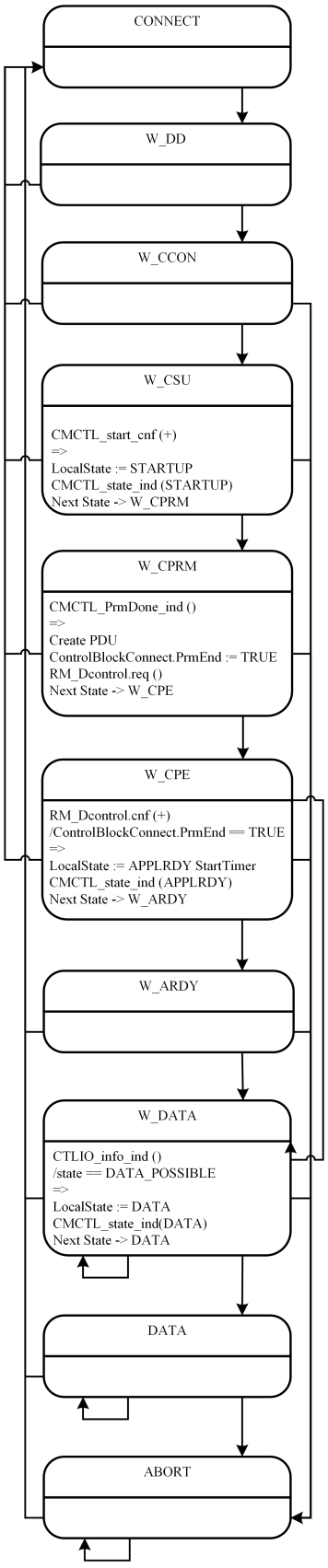


Abbildung 6.3: Zustandsmaschinen von PROFINET, angelehnt an (Norm DIN EN 61158-6-10).

Context Management Protocol Machine Controller ist die Zustandsmaschine, die für die Herstellung einer Verbindung eingesetzt wird. Sie besteht aus zehn Zuständen und 44 Transitionen. Sie ist die Hauptzustandsmaschine und dient als zentraler Punkt für die anderen Zustandsmaschinen, die mit ihren Funktionalitäten den Verbindungsaufbau unterstützen. In Abbildung 6.4 sind die Zustandsmaschine sowie ihre Zustände abgebildet und kurz beschrieben.



Zustände

CONNECT: Die Zustandsmaschine wartet auf einen CONNECT-Service-Aufruf von der Fieldbus Application Layer (FAL).

W_DD: Die Zustandsmaschine wartet auf eine erfolgreiche Ermittlung der Geräte und startet die Application Relationship (AR).

W_CCON: Die Zustandsmaschine wartet auf die Bestätigung der CONNECT-Anfrage.

W_CSU: Die Zustandsmaschine wartet auf die positive Bestätigung CMCTL_start_cnf, um darauf das Event CMCTL_state_ind(STARTUP) auszulösen. Danach wechselt die Zustandsmaschine in den Zustand W_CPRM.

W_CPRM: Die Zustandsmaschine wartet auf das Ende der Parametrisierung, um dann eine PrmEnd-Anfrage zu senden.

W_CPE: Die Zustandsmaschine wartet auf die positive Bestätigung RM_Dcontrol.cnf, um darauf das Event CMCTL_state_ind(APPLRDY) auszulösen. Danach wechselt die Zustandsmaschine in den Zustand W_ARDY.

W_ARDY: Die Zustandsmaschine wartet auf die ApplRdy-Anfrage der Geräte und antwortet mit einer ApplRdy-Antwort.

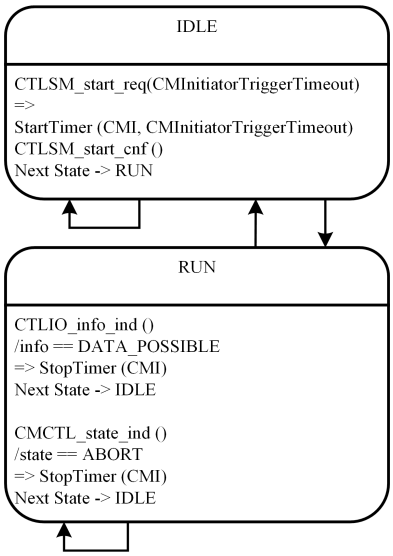
W_DATA: Bei einer CTLIO_info_ind mit der Bedingung, dass DATA_POSSIBLE ist, wird das Event CMCTL_state_ind(DATA) ausgelöst und in den nächsten Zustand DATA gewechselt.

DATA: Es werden Daten über die PPM und CPM ausgetauscht.

ABORT: Es werden bestehende ARs beendet.

Abbildung 6.4: Zustandsmaschine CMCTL.

Context Management Client Protocol Machine ist die Zustandsmaschine, die die Zustände der Zustandsmaschinen zwischen dem Ende der Parametrisierung und dem Start der PPM- und CPM-Verbindungsüberwachung kontrolliert. Sie besteht aus den zwei Zuständen IDLE und RUN sowie 15 Transitionen. In Abbildung 6.5 sind die Zustandsmaschine sowie ihre Zustände abgebildet und kurz beschrieben.



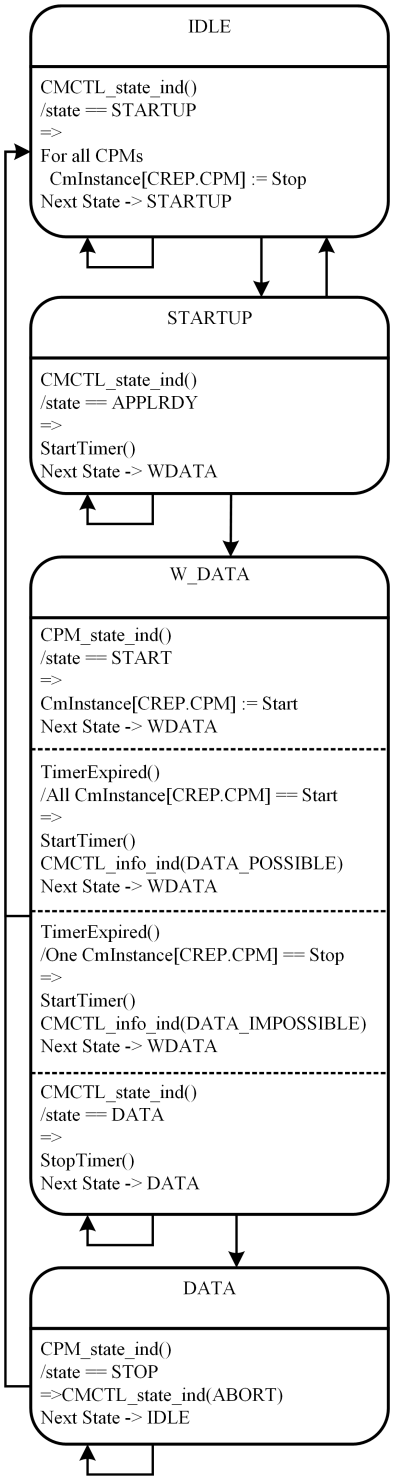
Zustände

IDLE: Die Zustandsmaschine wartet auf ein CTLSM-start_req, um daraufhin in den nächsten Zustand zu wechseln. Es wird außerdem ein Zeitgeber gestartet, der für die Überwachung zur Anwendung kommt.

RUN: Die Zustandsmaschine überwacht die Verbindung.

Abbildung 6.5: Zustandsmaschine CTLSM.

Context Management Input Oputput Protocol Machine Controller ist eine Zustandsmaschine, dargestellt in Abbildung 6.6, die den Datenaustausch der PPM und CPM überwacht. Die Zustandsmaschine besteht aus vier Zuständen IDLE, STARTUP, W_DATA und DATA und zwanzig Transitionen. Unter anderem nutzt die Zustandsmaschine die globale AR-Variable CmInstance, die alle Informationen der ARs enthält.



Zustände

IDLE: In diesem Zustand wartet die Zustands- maschine darauf, dass die CMCTL-Zustandsmaschine in den Zustand STARTUP wechselt.

STARTUP: Die Zustandsmaschine wartet darauf, dass die CMCTL-Zustandsmaschine das Event CMCTL-state_ind(APPLRDY) auslöst.

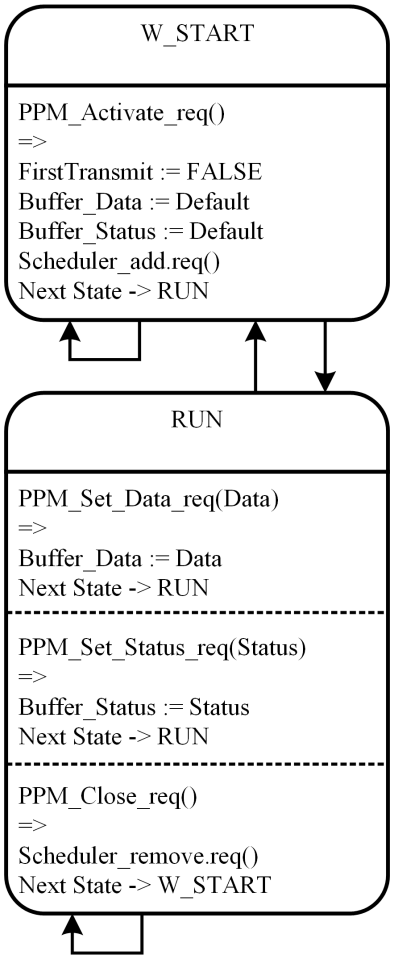
W_DATA: In diesem Zustand wartet die Zustandsmaschine darauf, dass die CPM das CPM_state_ind(Start)-Event auslöst, um daraufhin die sogenannte CmInstance-Variable auf Start zu setzen. Sobald sich alle CmInstance-Variablen auf Start befinden, wird in den nächsten Zustand DATA gewechselt.

DATA: In diesem Zustand wird der Zustand der CPM überwacht, indem auf ein CPM_state_ind(Stop) mit einem Zustandswechsel zu IDLE reagiert wird.

Abbildung 6.6: Zustandsmaschine CTLIO.

Provider Protocol Machine ist die Zustandsmaschine, die für die Kodierung des Frames zuständig ist. Die Zustandsmaschine kopiert die Daten und den Status der Daten entsprechend der Kodierung in den Frame. Der Frame-Handler, der auch Bestandteil der PPM ist, über-

gibt im Takt des Schedulers den Frame an die entsprechende Zustandsmaschine zur weiteren Übertragung. Das heißt die Bereitstellung und die Übertragung des Frames sind hier funktional getrennt. Insgesamt besteht die PPM aus zwei Zuständen und 19 Transitionen. In Abbildung 6.7 sind die Zustandsmaschine und vier ihrer relevantesten Transitionen dargestellt.



Zustände

W_START: In diesem Zustand initialisiert sich der Provider und wartet auf eine CPM-Activate-Anfrage. Durch eine solche Anfrage wird ein Scheduler gestartet und der Provider wechselt den Zustand zu RUN.

RUN: Der Provider ist im Zustand, in dem er auf PPM_Set_Data_req-Anfragen und PPM_Set_Status_req-/PPM_Close_req-Anfragen wartet und reagiert. Es werden die Daten und der DataStatus in den Buffer kopiert und von einem zuvor gestarteten Frame-Handler zyklisch zur weiteren Übertragung an die unteren Zustandsmaschinen weitergegeben.

Abbildung 6.7: Zustandsmaschine PPM.

Consumer Protocol Machine ist die Zustandsmaschine, die für das Dekodieren der Frames zuständig ist. Die Zustandsmaschine ist das Gegenstück zur PPM. Ihre Aufgabe besteht darin, die Daten und den Status der Daten auszulesen und an die entsprechenden Zustandsmaschinen zur weiteren Verarbeitung weiterzuleiten. Sie besteht aus drei Zuständen und 26 Transitionen und nutzt Frame Guards und Zeitgeber zur Überwachung der zyklischen Dekodierung der Frames. In Abbildung 6.8 sind die Zustandsmaschine sowie ihre Zustände abgebildet und kurz beschrieben.

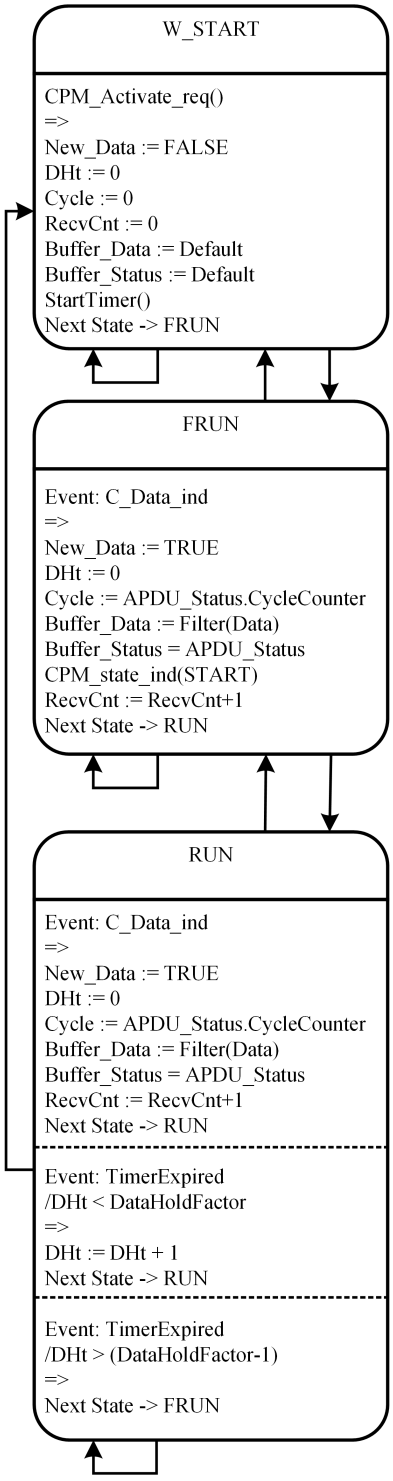


Abbildung 6.8: Zustandsmaschine CPM.

Zustände

W_START: In diesem Zustand wird auf die CPM_Activate-Anfrage gewartet. Die CPM_Activate-Anfrage initialisiert den Zustand mit den Werten aus Abbildung 6.8. Es wird außerdem ein Zeitgeber gestartet, der für die Überwachung des zyklischen Datenaustauschs verwendet wird.

FRUN: In diesem Zustand wird auf das erste Empfangen des Events C_Data_ind gewartet. Das Event C_Data_ind sagt aus, dass neue Daten zur Bearbeitung bereit stehen. Beim Auftreten des Events werden mehrere Daten in die Buffer kopiert. Es wird außerdem in den nächsten Zustand gewechselt.

RUN: Dieser Zustand wartet, wie der vorherige Zustand FRUN, auf das Event C_Data_ind. Wenn dieses Event in einem Zyklus nicht auftritt, läuft der Zeitgeber in eine Zeitüberschreitung und das Event TimerExpired tritt auf, in dem der DataHoldFactor (DHt)-Zähler inkrementell um eins hochgezählt wird.
Sollte die Bedingung DHt > (DataHoldFactor-1) erfüllt sein, wird zurück in den FRUN-Zustand gewechselt.

6.1.2.2 Sercos III

In Abbildung 6.9 sind alle Zustandsmaschinen abgebildet. Sercos III nutzt das Producer und Consumer-Modell, bei dem eine *Connection* zwischen Producer und Consumer konfiguriert und für den Datenaustausch verwendet wird. Außerdem überwachen die Producer und Consumer den Datenaustausch und werden deshalb nachfolgend beschrieben. Die Producer und Consumer sind Bestandteil der Application Relationship Protocol Machine.

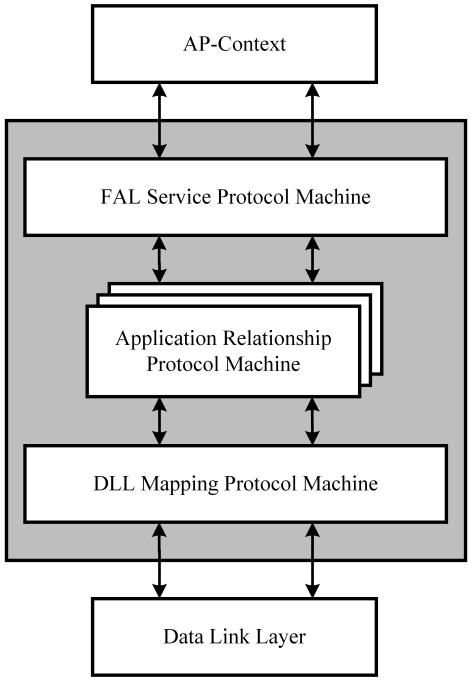


Abbildung 6.9: Zustandsmaschinen von Sercos III.

Application Relationship Protocol Machine – Producer: Der Producer ist für das Schreiben der Application Protocol Data Unit (APDU) verantwortlich. In Abbildung 6.10 ist die Zustandsmaschine des Producers dargestellt. Die Zustandsmaschine besteht aus sechs Zuständen und zwölf Transitionen. Nachfolgend sind die Zustände kurz erklärt:

- Der erste Zustand ist der *Init*-Zustand, in dem zunächst gewartet wird, bis die Hochlaufphase 4 erreicht ist.
- Im Zustand *Prepare* wird zunächst die *Connection* initialisiert. Das bedeutet, dass die Variable *Pcnt* und die Flaggen *C-CON.FlowControl* und *C-CON.ProducerReady* zurückgesetzt werden.
- Im nächsten Zustand *Ready* ist der Producer betriebsbereit. Die Flagge *C-CON.ProducerReady* wird auf 1 gesetzt.

- Im Zustand *Producing* werden gültige Daten der Applikation produziert und die Variable *C-CON.Counter* wird jeweils inkrementell um eins erhöht. Außerdem wird die Flagge *C-CON.NewData* auf True gesetzt.
- Im Zustand *Waiting* wartet der Producer auf neue Daten der Applikation. In jedem Zyklus wird die lokale Variable *Pcnt* inkrementell um eins erhöht.
- Im letzten Zustand *Stopping* wird die Flagge *C-CON.FlowControl* auf True gesetzt und die Daten der Applikation werden ungültig.

Ein wichtiger Punkt für die Zustandsmaschine des Producers ist der Ausgangspunkt der Kommunikation. Erst wenn die Hochlaufphase 4 erreicht ist, wechselt die Zustandsmaschine vom *Init*-Zustand in den *Prepare*-Zustand. Dadurch kann der Producer schon von einer funktionierenden zyklischen Kommunikation ausgehen.

Sobald der Producer initialisiert ist, kann er über die Transition *Start-Connection* in den nächsten Zustand *Ready* geschaltet werden. Im Zustand *Ready* setzt der Producer die Flagge *C-CON.ProducerReady* auf True.

Sobald der Producer nun Daten zur Verarbeitung zugewiesen bekommt, wechselt der Producer in den Zustand *Producing*. Im Zustand *Producing* fügt der Producer die Daten in die *Connection* des Frames ein. Solange der Producer in jedem Zyklus neue Daten erhält, bleibt der Producer im Zustand *Producing*.

Wenn der Producer jedoch keine Daten erhält, wechselt er in den Zustand *Waiting* und wartet auf neue Daten. In diesem Zustand verharrt der Producer bis neue Daten vorhanden sind, worauf der Producer wieder in den Zustand *Producing* wechselt.

Der Producer hat keinen Fehlerzustand, sondern ausschließlich den Zustand *Stopping*. Der Producer erreicht von den Zuständen *Ready*, *Producing* und *Waiting* den Zustand *Stopping* über die Transition *Stop-Connection*. Im Zustand *Stopping* ist die Flagge *C-CON.FlowControl* auf True gesetzt, worauf die Daten auf der Seite des Consumers nicht weiter verarbeitet werden. Aus dem Zustand *Stopping* ist es nur möglich, über die Transition *Reset-Connection* in den Zustand *Prepare* zu wechseln.

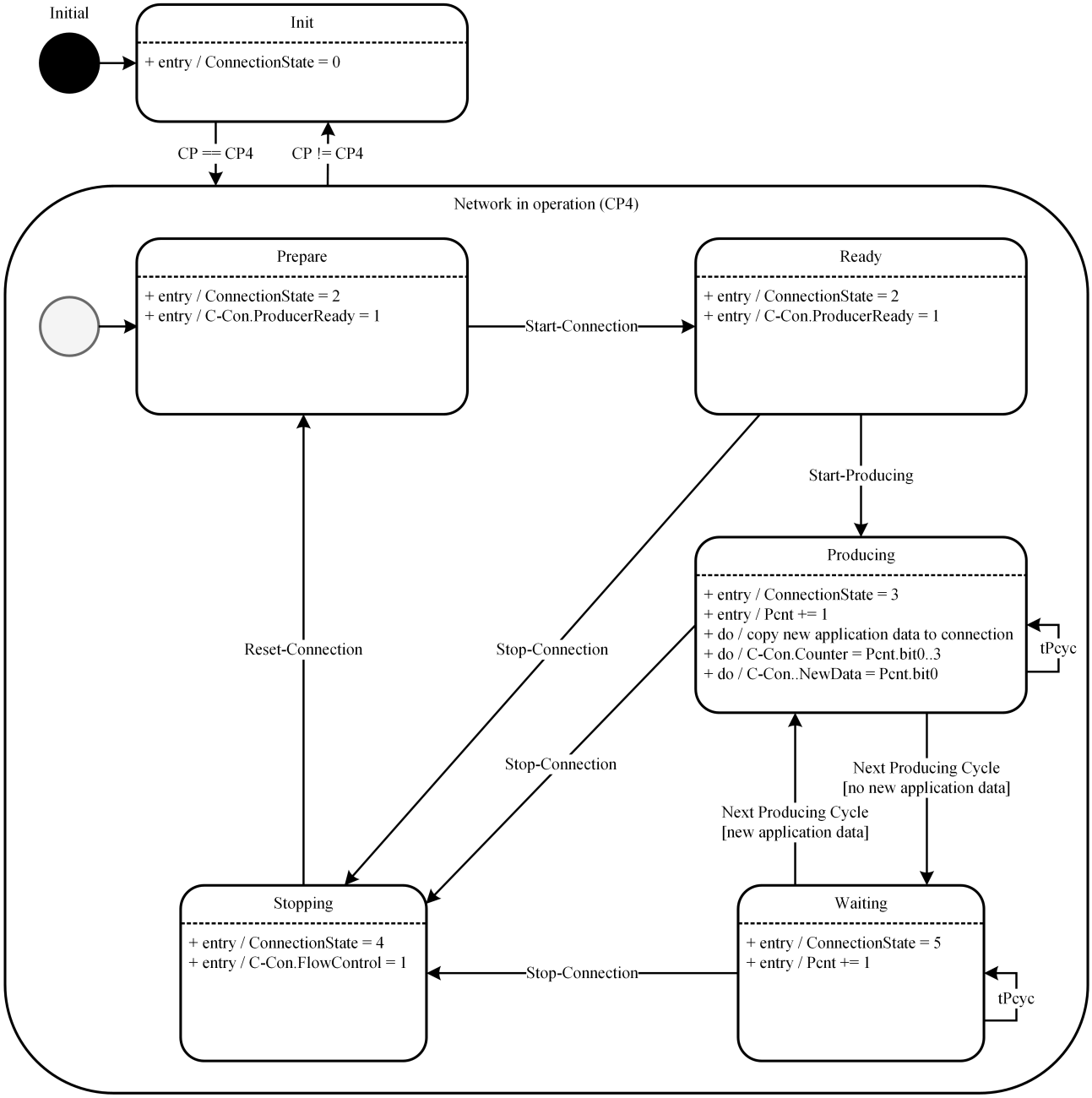


Abbildung 6.10: Zustandsmaschine des Producers.

Application Relationship Protocol Machine – Consumer: Der Consumer ist für das Evaluieren der APDU verantwortlich. In Abbildung 6.11 ist die Zustandsmaschine des Consumers dargestellt. Die Zustandsmaschine besteht aus sieben Zuständen und 17 Transitionen. Nachfolgend sind die Zustände kurz erklärt.

- Zunächst wartet der Consumer in der *Init*-Phase, bis die Hochlaufphase 4 erreicht ist.

- Im Zustand *Prepare* bereitet sich die Zustandsmaschine des Consumers für den Betrieb vor. Der Consumer setzt die Flagge *S-DEV.ConnectionError* auf False und überprüft die Flagge *C-CON.ProducerReady*.
- Im Zustand *Waiting* wartet der Consumer darauf, dass die Flagge *C-CON.ProducerReady* vom Producer auf True gesetzt wird. Dafür überprüft der Consumer mehrere Felder der *C-CON* und reagiert entsprechend mit Transitionen darauf. Außerdem setzt der Consumer die Variablen *Ccnt*, *CnewData* und einen Fehlerzähler zurück.
- Im Zustand *Consuming* verarbeitet der Consumer die Daten der Applikation und überprüft weiterhin die Felder der *C-CON*.
- Der Zustand *Warning* wird erreicht, wenn mindestens in einem Zyklus, keine neuen Daten beim Consumer angekommen sind. Der Consumer erhöht in jedem Zyklus einen Zähler. Außerdem überprüft der Consumer weiterhin die Felder der *C-CON*.
- Im Zustand *Stopped* hält der Consumer an. Der Consumer überprüft weiterhin die Flaggen *C-CON.ProducerReady* und *C-CON.FlowControl*.
- Im Zustand *Error* ist der Consumer im Fehlerzustand und setzt die Flagge *S-DEV.ConnectionError* auf True.

Wie auch schon beim Producer ist die Zustandsmaschine des Consumers zunächst im *Init*-Zustand. Erst wenn die Hochlaufphase 4 erreicht ist, wechselt der Consumer in den Zustand *Prepare*. Sobald die empfangene Flagge *C-CON.ProducerReady* auf True steht, wechselt der Consumer in den Zustand *Waiting*. Über die weitere Transition *Restart-Prepare* ist es dem Consumer möglich, vom Zustand *Waiting* zurück in den Zustand *Prepare* zu wechseln. Der Consumer wechselt vom Zustand *Waiting* zum Zustand *Consuming*, sobald die Flaggen *C-CON.ProducerReady* und *C-CON.FlowControl* entsprechend gesetzt und neue Daten in der *Connection* vorhanden sind. Der Consumer wechselt aus dem *Consuming* Zustand, wenn entweder die Flagge *C-CON.ProducerReady* auf False gesetzt ist oder in einem Zyklus keine neuen Daten vorhanden sind. Der Consumer kehrt zurück in den Zustand *Consuming*, wenn die Flagge *C-CON.ProducerReady* zurück auf True gesetzt ist und neue Daten vorhanden sind. Jedes Mal, wenn der Consumer in einem Zyklus im Zustand *Warning* ist, erhöht der Consumer den Zähler inkrementell um eins. Wenn der Zähler ein konfiguriertes Limit überschreitet, wechselt der Consumer in den Zustand *Error*. Der Consumer kann ausschließlich über ein Zurücksetzen des Fehlers aus dem *Error*-Zustand in den *Prepare*-Zustand wechseln. Der Consumer kann den Zustand *Stopped* über die Transition *Start-Stopped* aus den Zuständen *Waiting*, *Consuming* und *Warning* erreichen. Der Consumer kann nur über die Transition *Leave-Stopped* den Zustand *Stopped* verlassen.

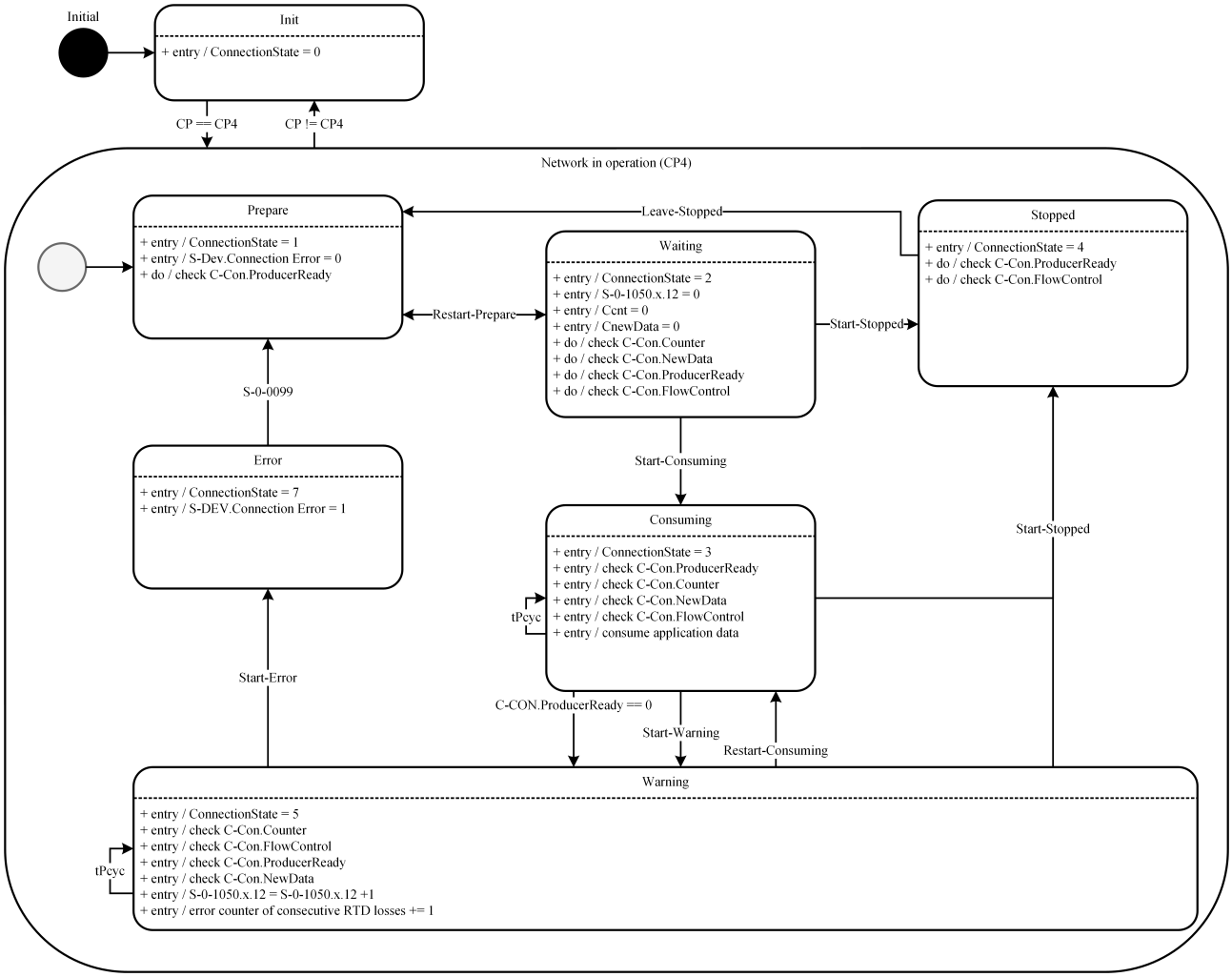


Abbildung 6.11: Zustandsmaschine des Consumers.

6.1.2.3 Vergleich der Zustandsmaschinen von PROFINET und Sercos III

Es werden nun Unterschiede und Gemeinsamkeiten der Zustandsmaschinen von Sercos III und PROFINET diskutiert. Bei Sercos III übernehmen die Zustandsmaschinen der Producer und Consumer den Datenaustausch und dessen Überwachung. Grundvoraussetzung ist die Hochlaufphase 4, die eine funktionierende zyklische Kommunikation gewährleistet. Die Überwachung des Datenaustauschs läuft hauptsächlich auf Seiten des Consumers ab, der über Warnings und Errors Unregelmäßigkeiten anzeigt. Der Producer ist sich dieser Warnings und Errors nicht direkt bewusst und reagiert auch nicht auf diese. Die Information der Warnings und Errors wird im Status-Device zurückübertragen. Bei PROFINET hingegen sind mehrere Zustandsmaschinen an dem Datenaustausch und der Überwachung beteiligt. Die Zustandsmaschinen sind nach Funktionen unterteilt. So überwacht z. B. die Zustandsmaschine CTLIO die Zustandsmaschinen PPM und CPM. Die Zustandsmaschine CMCTL von PROFINET überprüft beim

Hochlauf, ob die Applikation betriebsbereit ist. Ein vergleichbarer Zustand existiert bei Sercos III nicht. Die Zustandsmaschinen der Producer und Consumer sind bei beiden Protokollen für das Lesen und Schreiben der APDU-Daten sowie das Setzen und Evaluieren der Connection Control (C-CON) bzw. des APDU_Status zuständig.

6.1.3 Application Protocol Data Unit

In diesem Kapitel werden die Strukturen der Frames für den zyklischen Datenverkehr beschrieben und die Relevanz der einzelnen Inhalte diskutiert.

6.1.3.1 PROFINET

Wie in Abbildung 6.12 gezeigt, ist die APDU von PROFINET in einem Ethernet-Frame mit dem Ethertype 0x8892 eingebettet. Neben der Preamble (Pre), dem SFD, der Destination Address (DA) und der Source Address (SA) ist außerdem noch eine VLAN-Priorität im Frame enthalten. Die APDU besteht aus einem Header (HDR), Data und einem APDU_Status. Der APDU_Status besteht aus vier Bytes und gliedert sich in ein *Cycle Counter* mit zwei Bytes, einem *Data Status* und einem *Transfer Status* mit jeweils einem Byte. Der Empfänger überwacht den *Cyclic Counter*, indem er den Zähler auswertet und anhand von Sprüngen im Zähler erkennt, ob es zu Unregelmäßigkeiten im Datenaustausch gekommen ist. Die genaue Enkodierung des *Data Status* ist in Tabelle 6.1 aufgelistet.

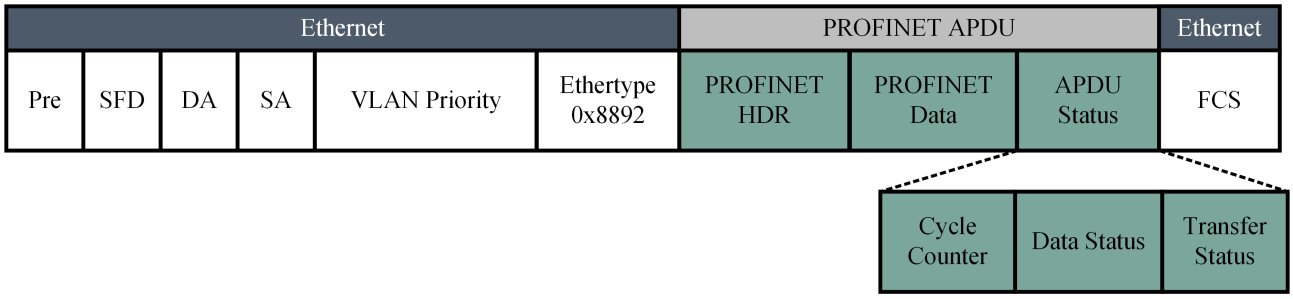


Abbildung 6.12: Struktur eines PROFINET-Frames.

Aus dem *Data Status* kann der Consumer mehrere Informationen extrahieren und entsprechend darauf reagieren. Er kann z. B. anhand des *Data Status* erkennen, ob ein Problem vorliegt oder ob die Operation normal verläuft. Des Weiteren kann er erkennen, ob der Provider im *Stop*- oder *Run*-Zustand ist und ob das empfangene *DataItem* gültig ist. Der *Transfer Status* beinhaltet Informationen über die Übertragung des Frames. So sind z. B. Informationen über die FCS

oder der Überprüfung der Länge des Frames enthalten. In der Tabelle 6.1 ist gekennzeichnet, ob die einzelnen Felder der Enkodierung relevant für die Überwachung sind.

Tabelle 6.1: Data Status Bytes.

Bit Nr.	**Wert**	**Name**	**Relevant**
7		DataStatus.Ignore	✓
	0x00	Evaluiere den DataStatus	
	0x01	Ignoriere den DataStatus	
6		DataStatus.Reserved	
5		DataStatus.StationProblemIndicator	✓
	0x00	Problem erkannt	
	0x01	Normale Operation	
4		DataStatus.ProviderState	✓
	0x00	Stop	
	0x01	Run	
3		DataStatus.Reserved	
2		DataStatus.DataValid	✓
	0x00	DataItem ungültig	
	0x01	DataItem gültig	
1		DataStatus.Redundancy	
0		DataStatus.State	

6.1.3.2 Sercos 3

In diesem Kapitel wird die APDU von Sercos III in der Hochlaufphase 4 betrachtet. Die APDU ist in einem Ethernet-Frame mit dem Ethertype 0x88CD eingebettet. Bei Sercos III wird zwischen zwei Arten von Telegrammen, MDT und AT, unterschieden. Das MDT überträgt Daten vom Master zu den Slaves. Das AT überträgt Daten vom Slave zum Master und zu anderen Slaves. Unabhängig ob MDT oder AT – ein Sercos III-Telegramm besteht grundsätzlich aus einem Header und einer Payload. Je nachdem, ob es sich bei dem Frame um ein MDT oder AT handelt, unterscheidet sich die Struktur des Headers und der Payload. Ein Beispiel dieses Unterschieds ist in Abbildung 6.13 dargestellt. Im Fall eines MDTs ist eine Variable namens *Device Control* und im Fall eines ATs eine Variable namens *Device Status* enthalten. Neben den Variablen *Device Control* und *Device Status* können außerdem noch eine oder mehrere

Connections in dem Frame enthalten sein. Producer und Consumer nutzen *Connections* für den Datenaustausch.

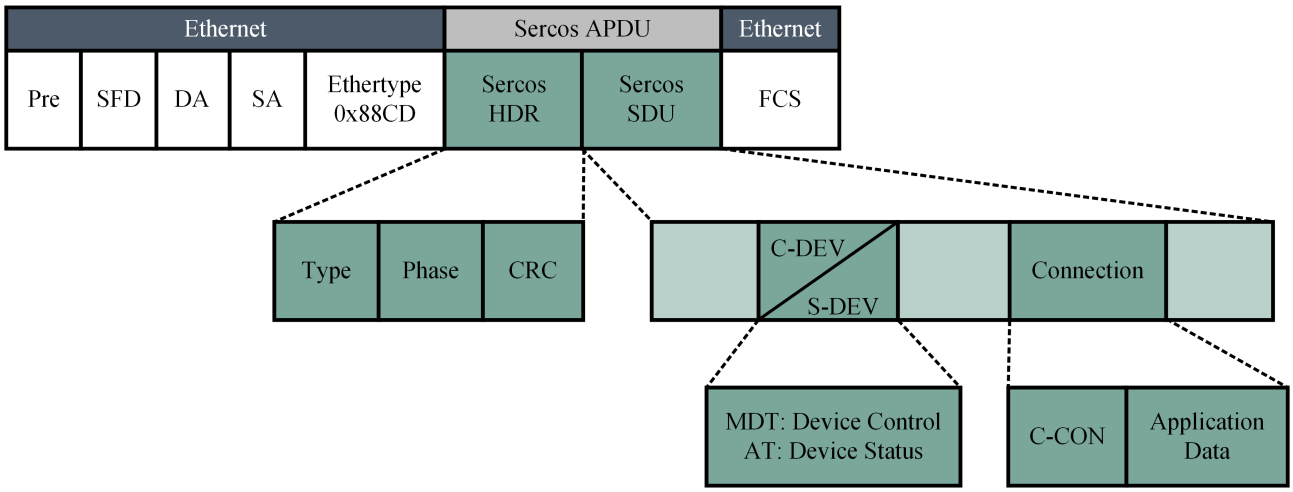


Abbildung 6.13: Struktur eines Sercos III-Frames.

Über die zwei Bytes der Variablen *Device Status* kann ein Slave einige Informationen über seinen Zustand abbilden. Besonders interessant ist das Bit 15 *Communication Warning Interface*, das Bit 9 *Error Connection* und das Bit 8 *Slave Valid*. Die Bits zeigen an, ob es bei der Kommunikation zu Warnungen oder Fehlern gekommen ist und ob der Slave gültig ist. In der Tabelle 6.2 ist gekennzeichnet, ob die einzelnen Felder der Enkodierung relevant für die Überwachung sind.

Tabelle 6.2: Status Device (S-DEV).

Bit Nr.	**Wert**	**Name und Beschreibung**	**Relevant**
15		Communication Warning Interface	✓
	0x00	Keine Warning ist aufgetreten	
	0x01	Kommunikationswarnung ist aufgetreten	
14		Topologie HS	
13-10		Topologie- und Port-Status	
9		Error Connection	✓
	0x00	Fehlerfreie Verbindung	
	0x01	Fehler in der Verbindung ist aufgetreten	
8		Slave Valid	✓
	0x00	Ungültig	
	0x01	Gültig	
7		Error of Device (C1D)	
6		Warning of Device (C2D)	
5		Procedure Command Change Bit	
4		Sub-Device Level	
3		Loopback	
2-0		Reserved	

Der Producer nimmt über die *C-CON* Einfluss auf die Zustandsmaschine des Consumers. Die C-CON besteht aus zwei Bytes mit mehreren Flaggen. Vier der Flaggen werden nun näher beschrieben. Die C-CON beinhaltet in den Bits 12 bis 15 einen Zähler, der in jedem Zyklus inkrementell um eins vom Producer erhöht wird. Über diesen Zähler kann auf Seiten des Consumers überprüft werden, ob es zu Sprüngen im Datenaustausch kommt. Der Producer teilt dem Consumer über das Bit 4 *Flow-Control* mit, dass er das Produzieren von Prozessdaten beendet. Dadurch ist der Consumer informiert, dass er nicht mit einem Fehler auf das Fehlen der Prozessdaten reagieren muss. Bit 1 *New Data* sagt aus, ob neue Prozessdaten in der *Connection* enthalten sind. Bit 0 zeigt an, ob der Producer betriebsbereit ist.

Tabelle 6.3: Connection Control (C-CON).

Bit Nr.	**Wert**	**Name und Beschreibung**	**Relevant**
15-12		Counter	✓
11-8		Reserved	
7		Real-time bit 2	
6		Real-time bit 1	
5		Reserved	
4		Flow-Control	✓
	0x00	Run	
	0x01	Stop	
3		Reserved	
2		Data Field Delay	
1		New Data	✓
	toggle		
0		ProducerReady	✓
	0x00	Nicht Betriebsbereit	
	0x01	Betriebsbereit	

6.1.3.3 Vergleich der Strukturierung der Application Protocol Data Unit von PROFINET und Sercos III

Der Grundaufbau beider APDUs ist sehr ähnlich. Sie bestehen beide aus einem Protokoll-Header, einem Steuerwort (C-DEV) oder Statuswort (S-DEV oder APDU_Status) und den eigentlichen Daten. Bei Sercos III ist im Fall eines MDTs ein Steuerwort und im Fall eines ATs

ein Statuswort in der APDU enthalten. Da es sich bei Sercos III um ein Summenrahmentelegramm handelt, werden die Daten in der APDU noch weiter strukturiert. Sercos III definiert sogenannte *Connections*, die es ermöglichen, Prozessdaten an mehrere Entitäten über ein Telegramm zu verteilen. Diese *Connections* verfügen über ihr eigenes Steuerwort (C-CON). In Tabelle 6.4 sind nun die relevanten Steuer- und Statuswörter von PROFINET und Sercos III gegenübergestellt und abstrahiert.

Tabelle 6.4: Gegenüberstellung der S-DEV/C-CON und dem APDU_Status.

Sercos III	**PROFINET**	**Abstraktion**
S-DEV.Warning	DataState.ProblemIndicator	Es liegt eine Warnung in der Kommunikation vor.
S-DEV.Error	DataState.ProblemIndicator	Es liegt ein Fehler in der Kommunikation vor.
S-DEV.Slave	DataState.Ignore	Es soll das Status-Wort ignoriert werden.
C-CON.Counter	APDU_Status.Cycle Counter	Zyklischer Zähler, der inkrementell um 1 pro Zyklus erhöht wird.
-	DataState.ProducerState	Der Zustand des Producers.
C-CON.ProducerReady	DataState.DataValid	Die Gültigkeit der Daten.
C-CON.FlowControl	-	Ein Steuerwort zur Beeinflussung des Consumers.
C-CON.NewData	-	Ein Indikator, ob neue Daten vorhanden sind.

6.1.4 Datenaustausch

Um den Datenaustausch zwischen zwei Entitäten zu beschreiben, werden ARs zwischen diesen definiert. Die ARs können aus mehreren Communication Relationship (CR)s bestehen. Es gibt verschiedene CR-Typen, die unterschiedliche Eigenschaften besitzen. Um die CRs ausreichend beschreiben zu können, werden sieben Eigenschaften verwendet:

Der Modelltyp beschreibt das verwendete Übertragungsmodell. Es kann zwischen einem Client/Server- oder Producer/Consumer-Modell ausgewählt werden.

Die Kardinalität beschreibt die Anzahl der Sender- und Empfänger-Entitäten, die miteinander in Relation stehen. Möglichkeiten sind 1-zu-1, 1-zu-N und N-zu-N.

Die Übertragungsrichtung beschreibt, ob es sich um eine unidirektionale oder bidirektionale Übertragung zwischen den Entitäten handelt.

Der Übertragungstyp beschreibt, ob die Übertragung zyklisch oder azyklisch stattfindet.

Der Speichertyp sagt aus, ob eine Queue oder ein Buffer bei der Übertragung verwendet wird. Im Fall einer Queue werden alle Datensätze übertragen und im Fall eines Buffers nur der aktuelle.

Der Verbindungstyp beschreibt, ob es sich beim Datenaustausch um einen verbindungsorientierten oder verbindungslosen Datenaustausch handelt. Der große Unterschied zwischen den beiden Typen ist, dass der verbindungsorientierte Datenaustausch aus den drei Phasen Verbindungsaufbau, Datenaustausch und Verbindungsabbau besteht und der Datenaustausch überwacht werden kann. Der verbindungslose Datenaustausch hingegen besteht nur aus der Phase Datenaustausch und wird nicht überwacht.

Der Bestätigungstyp sagt aus, ob die Übertragung von Daten bestätigt wird.

6.1.4.1 PROFINET

Ein PROFINET FAL Application Process (AP) nutzt Application Relationship Application Service Element (AR ASE)s, um Prozessdaten mit einem anderen FAL AP auszutauschen. Ein AR ASE besteht aus einem oder mehreren Application Relationship End Point (AREP)s. Die Aufgabe der AREPs ist es, die APDUs über CRs zu übertragen. Eine AR kann aus einer oder mehreren CRs bestehen. Ein Communication Relationship End Point (CREP) kann entweder ein Client, Server, Client-Server (Peer), Producer oder Consumer sein. CRs sind als unidirektionale oder bidirektionale Übertragungsrichtungen zwischen CREPs modelliert. Die CRs von PROFINET unterstützen die zwei Kardinalitäten 1-zu-1 und 1-zu-N. PROFINET verwendet das Producer/Consumer- und das Client/Server-Modell. Die APDU wird entweder in einem Buffer oder in einer Queue zur Übertragung gespeichert. Beim Speichern der APDU in einem Buffer wird immer die alte APDU überschrieben. Die Queue hat eine endliche Zahl von APDU-Plätzen, die nach dem First-In-First-Out-Prinzip übertragen werden. Der Zeitpunkt zur Übertragung der APDUs hängt davon ab, ob die Übertragung zyklisch oder azyklisch ausgeführt wird. Je nach verwendeter CR, wird die Übertragung der APDU bestätigt.

PROFINET unterstützt insgesamt fünf unterschiedliche CRs:

1. **Buffer Buffer Unconfirmed Unidirectional (BBUU)**
   Z. B. zyklische Übertragung von Eingangs- und Ausgangsdaten zwischen einem Producer und einem Consumer
2. **Buffer Buffer Unconfirmed Bidirectional (BBUB)**
   Z. B. zyklische bidirektionale Übertragung von Eingangs- und Ausgangsdaten zwischen einem Client und einem Server
3. **Buffer Buffer Unconfirmed Unidirectional: One to Many (BBUU-OM)**
   Z. B. zyklische Übertragung von Eingangs- und Ausgangsdaten zwischen einem Producer und mehreren Consumern
4. **Queue Queue Confirmed Bidirectional: Connection-Oriented (QQCB-CO)**
   Z. B. Zugriff auf Diagnose oder Identifikationsinformationen
5. **Queue Queue Confirmed Bidirectional: Connectionless (QQCB-CL)**
   Z. B. Übertragung von Alarmen

Eine Übersicht über die Eigenschaften der fünf CRs ist in Tabelle 6.5 gegeben.

Tabelle 6.5: Eigenschaften der fünf ARs.

Eigenschaften	**BBUU**	**BBUU-OM**	**BBUB**	**QQCB-CO**	**QQCB-CL**
Modell	Producer/ Consumer	Producer/ Consumer	Client/ Server	Client/ Server	Client/ Server
Kardinalität	1-zu-1	1-zu-N	1-zu-1	1-zu-1	1-zu-1
Übertragungs-richtung	Unidirektional	Unidirektional	Bidirektio-nal	Bidirektio-nal	Bidirektio-nal
Übertragungs-typ	Zyklisch	Zyklisch	Zyklisch	Azyklisch	Zyklisch
Speicher	Buffer	Buffer	Buffer	Queue	Queue
Verbindungstyp	Gekoppelt	Gekoppelt	Gekoppelt	Gekoppelt	Entkoppelt
Bestätigungs-typ	Unbestätigt	Unbestätigt	Unbestätigt	Bestätigt	Bestätigt

In Abbildung 6.14 ist die CR BBUU-OM zwischen zwei FAL AP dargestellt. Die Producer schreiben die Prozessdaten und setzen den APDU_Status in der APDU. Der Consumer liest die Prozessdaten und evaluiert den APDU_Status aus der APDU. Die FAL APs tauschen über

den bidirektionalen Datenaustausch Status-Informationen im APDU_Status aus und reagieren entsprechend darauf.

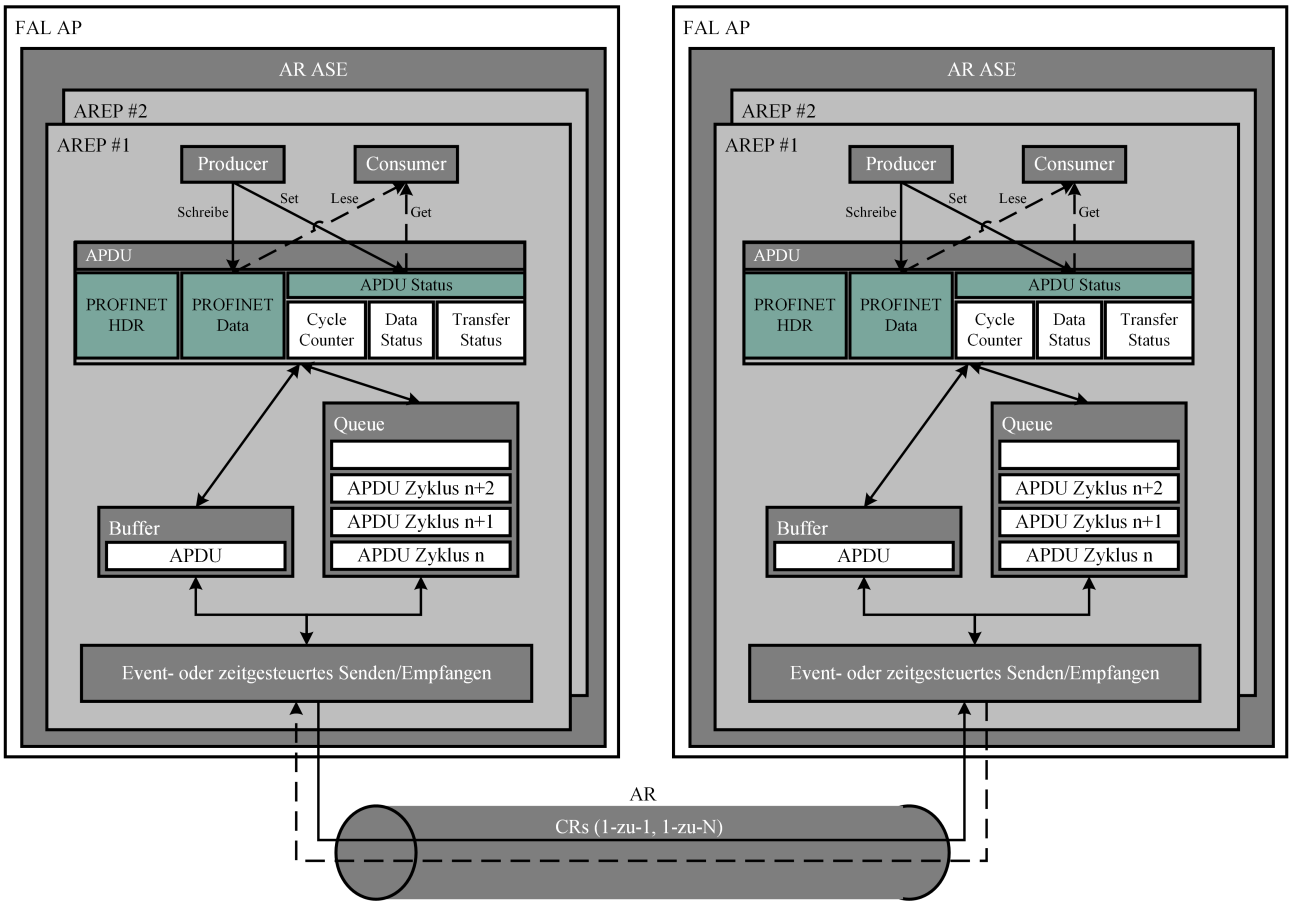


Abbildung 6.14: Datenfluss bei PROFINET.

6.1.4.2 Sercos III

Eine Sercos III FAL AP nutzt Sercos III AR ASEs, um Prozessdaten mit einem anderen FAL AP auszutauschen. Ein AR ASE besteht aus einem oder mehreren AREPs. Die Aufgabe der AREPs ist es, die APDUs über CRs zu übertragen. Es gibt bei Sercos III drei verschiedene ARs:

1. **Point-to-Point (1-zu-1) Azyklisch Bestätigt Client/Server AREP**
   Service Channel (SVC)-Kommunikation.

2. **Point-to-Point (1-zu-1) Zyklisch Unbestätigt Producer/Consumer AREP**
   RTC-MS-Kommunikation.

3. **Point-to-Multipoint (1-zu-N) Zyklisch Unbestätigt Producer/Consumer** RTC-CC-Kommunikation.

Die SVC-Kommunikation basiert auf dem Client/Server-Modell mit einer 1-zu-1-Kardinalität. Es wird eine bidirektionale Kommunikation verwendet, die azyklisch stattfindet. Bei der Übertragung werden die APDUs in einer Queue zwischengespeichert, bis sie an der Reihe sind, übertragen zu werden. Die RTC-MS-Kommunikation verwendet das Producer/Consumer-Modell mit einer 1-zu-1-Kardinalität. Die Producer übertragen dabei die APDU zyklisch und unidirektional. Als Speicher der APDU wird ein Buffer verwendet, der im Fall einer neuen APDU überschrieben wird. Die RTC-CC-Kommunikation unterscheidet sich nicht von der RTC-MS-Kommunikation bis auf die Kardinalität, bei der es sich um eine 1-zu-N-Kardinalität handelt. Eine Zusammenfassung über die drei ARs ist in Tabelle 6.6 gegeben.

Tabelle 6.6: Eigenschaften der drei ARs.

Eigenschaften	**SVC**	**RTC-MS**	**RTC-CC**
Modell	Client/Server	Producer/Consumer	Producer/Consumer
Kardinalität	1-zu-1	1-zu-1	1-zu-N
Übertragungsrichtung	Bidirektional	Unidirektional	Unidirektional
Übertragungstyp	Azyklisch	Zyklisch	Zyklisch
Speicher	Queue	Buffer	Buffer
Verbindungstyp	Gekoppelt	Entkoppelt	Entkoppelt
Bestätigungstyp	Bestätigt	Unbestätigt	Unbestätigt

In Abbildung 6.15 ist der Datenfluss zwischen zwei FAL APs dargestellt. Der Producer ist für das Setzen der *C-CON*-Flagge und das Schreiben der *Application Data* verantwortlich. Auf der Empfängerseite ist der Consumer für das Evaluieren der *C-CON*-Flagge sowie für das Lesen der *Application Data* verantwortlich. Bei der *C-CON*-Flagge handelt es sich um zwei Bytes, mit denen der Producer Einfluss auf den Consumer nehmen kann. Ein interessanter Punkt des Datenaustauschs bei Sercos III ist, dass der Austausch prinzipiell von einem Producer unidirektional zu einem oder mehreren Consumern abläuft. Das heißt, Störungen in der Connection werden aufgrund des unidirektionalen Datenflusses dem Producer nicht mitgeteilt. Sercos III bietet jedoch in einem AT die Möglichkeit, die Status-Informationen über die Status-Device (S-DEV)-Variable zu übertragen. Die S-DEV-Variable enthält Informationen darüber, ob es in einer *Connection* zu Warnungen oder Fehlern gekommen ist.

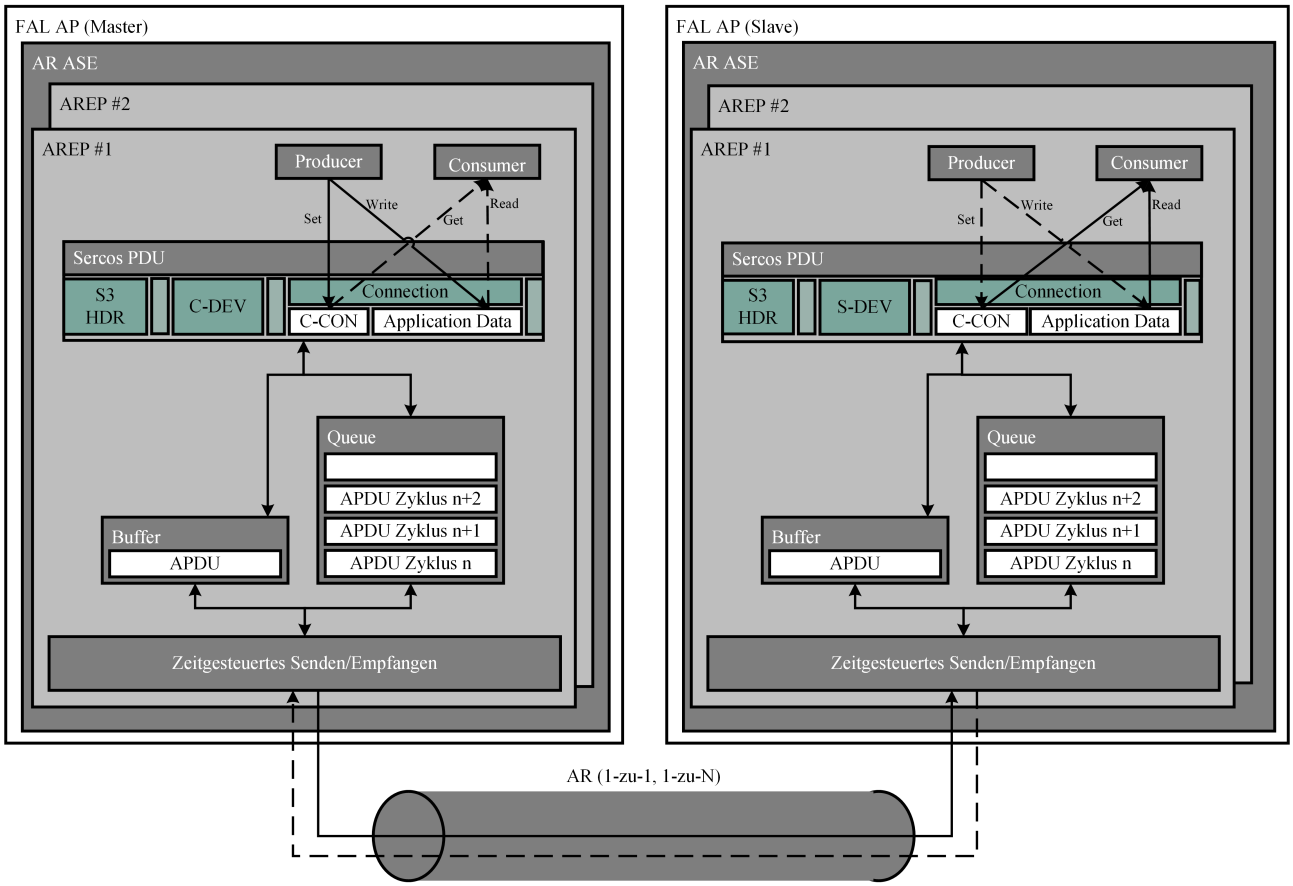


Abbildung 6.15: Datenfluss bei Sercos III.

6.1.4.3 Vergleich des Datenaustauschs von PROFINET und Sercos III

Sercos III und PROFINET unterscheiden sich bei der Strukturierung des Datenaustauschs insofern, dass bei Sercos III der Austausch rein über die zwei ARs – SVC und Real-Time Channel (RTC) – beschrieben wird. Bei PROFINET hingegen werden ARs definiert, die wiederum aus fünf CRs bestehen. Zusammengefasst gibt es den azyklischen Austausch von Geräteinformationen und Konfigurationsparametern sowie den zyklischen Datenaustausch von Prozessdaten. Bei Sercos III wird der azyklische Austausch über SVC ARs und der zyklische Datenaustausch über RTC ARs abgebildet. PROFINET bildet den azyklischen Austausch über die Implicit ARs sowie den zyklischen Austausch über die IO ARs ab. Ein weiterer Unterschied ist, dass bei PROFINET alle CRs, bis auf die QQCB-CL, auf einer verbindungsorientierten Kommunikation basieren. Die verbindungsorientierte Kommunikation besteht aus drei Phasen – Verbindungsaufbau, Datenaustausch und Verbindungsabbau. Da Sercos III über die Hochlaufphasen in Phase 3 und Phase 4 eine stabile zyklische Kommunikation voraussetzt, ist ein weiterer Verbindungsaufbau nicht notwendig. Beide Protokolle haben aber auch Gemeinsamkeiten. Bei beiden

Protokollen werden Producer und Consumer eingesetzt. Die Producer und Consumer schreiben und lesen die Prozessdaten und setzen und evaluieren die Status-Informationen. Es ist auch bei beiden Protokollen möglich, Status-Informationen wie Warnungen oder Fehler zwischen den FAL AP auszutauschen.

6.1.5 Zusammenfassung

In dem Kapitel wurden die Klasse 3 Busprotokolle PROFINET und Sercos III miteinander verglichen. Der Fokus wurde auf die Hochlaufphasen, die Zustandsmaschinen, die APDU und den Datenaustausch gerichtet. Der Vergleich hat gezeigt, dass sich die Hochlaufphasen der Busprotokolle insofern ähneln, dass sie erst eine stabile Kommunikation aufbauen, bevor der zyklische Datenaustausch überwacht wird.

Die Anzahl der Zustandsmaschinen unterscheidet sich deutlich. PROFINET teilt viele der Funktionen in einzelne Zustandsmaschinen auf, wohingegen Sercos III den Datenaustausch hauptsächlich über eine Zustandsmaschine je Producer und Consumer abbildet. Die Struktur der APDU von PROFINET und Sercos III ähnelt sich insofern, dass beide Busprotokolle Prozessdaten als auch Status-Informationen über die APDU versenden.

Bei Beiden Busprotokollen wird der zyklische Datenaustausch dadurch überwacht, dass in jedem Zyklus eine APDU erwartet wird. Über diesen Mechanismus werden Rückschlüsse über die Qualität der Verbindung geschlossen.

6.2 Ableitung eines Konzepts zur Überwachung des Datenaustauschs

In diesem Kapitel wird ein Konzept zur Kommunikationsüberwachung für OPC UA PubSub abgeleitet. Dazu wird die Top-down-Methode Objektorientierte Analyse und Design (OOAD) angewendet. In Abbildung 6.16 ist dargestellt, wie die Top-down-Methode bei der Ableitung der Kommunikationsüberwachung zum Einsatz kommt. Dazu werden sechs Ebenen definiert. Mit jeder Ebene sinkt der Abstraktionsgrad, wohingegen der Detaillierungsgrad steigt. Die Struktur des Kapitels orientiert sich an drei W-Fragen:

Was wird überwacht? Wie im Stand der Technik gezeigt, gibt es im OSI-Referenzmodell mehrere Möglichkeiten, eine Kommunikation zu überwachen. Beispielsweise kann eine Kommunikation auf Signal- oder Bit-Ebene überwacht werden. Da verschiedene Überwachungen möglich sind, grenzt die Aussage, eine OPC UA PubSub-Kommunikation zu überwachen, die Möglichkeiten der Überwachung noch nicht vollständig ein. Um die Frage zu beantworten, wird anhand der gewonnenen Erkenntnisse aus der Analyse und der Abstraktion der Busprotokolle das Ziel der Überwachung definiert. Das Ergebnis ist eine genaue Definition der Überwachung (Ebene 6) als auch ein erstes Grundkonzept (Ebene 5), das die Basis für die weiteren Top-down-Schritte darstellt.

Welche Entitäten sind beteiligt? An einer OPC UA PubSub basierten Kommunikation sind mehrere Entitäten mit unterschiedlichen Aufgaben beteiligt. Es wird untersucht, welche Entität sich für die zusätzliche Aufgabe der Überwachung am besten eignet. Diese Entscheidung wird zum einen aus den Erkenntnissen der Analyse und der Abstraktion der Busprotokolle aus Kapitel 6.1, als auch aus einer Analyse der OPC UA PubSub-Entitäten mit ihren Aufgaben und Informationsmodellen aus Kapitel 6.2 getroffen. Dazu werden die Entitäten des zuvor erarbeiteten Grundkonzepts weiter analysiert und im Detail ausgearbeitet (Ebene 3 und 4).

Wie funktioniert die Überwachung? Nachdem die beteiligten Entitäten identifiziert und bestimmt sind, wird das Grundkonzept um weitere Details erweitert (Ebene 1 bis 3). Dazu werden zum einen ergänzende Zustandsmaschinen zur Überwachung des Datenaustauschs definiert. Zum andern wird das Message Mapping UADP um die nötigen Inhalte für die Überwachung ergänzt. Zum Schluss werden neue Informationsmodelle erstellt, die zur Modellierung der Entitäten verwendet werden.

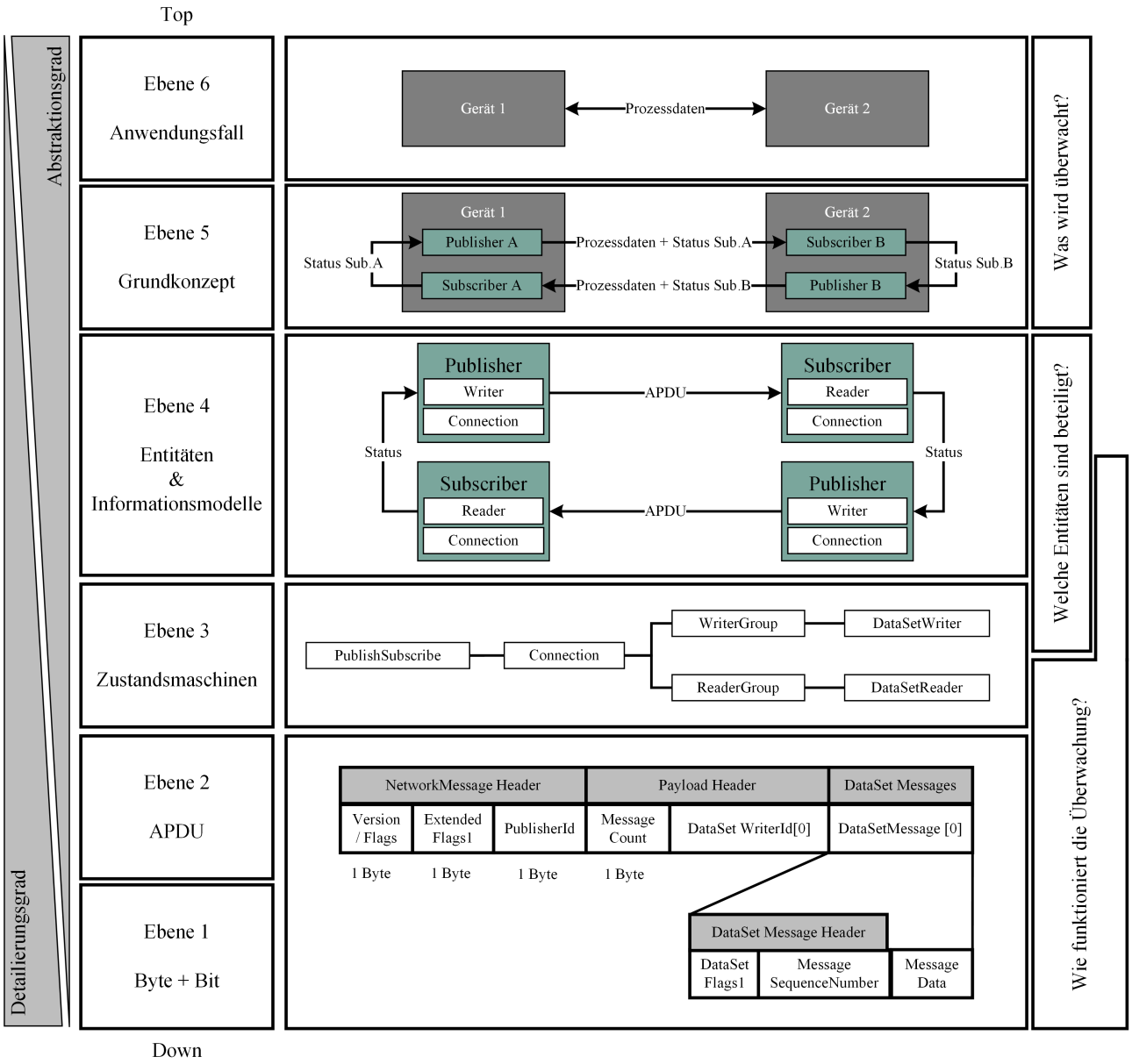


Abbildung 6.16: Ableitung der Überwachung mit der Top-down-Methode.

6.2.1 Ableitung des Konzepts zur Überwachung des Datenaustauschs

Um den unterschiedlichen Anforderungen der Anwendungsfälle gerecht zu werden, bieten PROFINET und Sercos III verschiedene ARs und CRs an. Diese können entweder durch einen azyklischen oder zyklischen Übertragungstyp näher charakterisiert werden. Bei OPC UA entspricht die azyklische Kommunikation dem Client/Server- und die zyklische Kommunikation dem Publish/Subscribe-Modell. Als Ausgangspunkt des Datenaustauschs setzen PROFINET

und Sercos III auf unterschiedliche Ansätze. PROFINET nutzt einen verbindungsorientierten Ansatz, bei dem zunächst ein Verbindungsaufbau durchgeführt wird, bevor dann der eigentliche zyklische Datenaustausch startet. Sercos III basiert auf einer verbindungslosen Kommunikation basierend auf der Sercos III Hochlaufphase. In der Hochlaufphase 3 wird überprüft, ob die zyklische Kommunikation stabil ist und wechselt erst danach in die Hochlaufphase 4, in der der eigentliche zyklische Datenaustausch über Producer und Consumer abläuft. Bisher bietet das Publisher/Subscriber-Modell von OPC UA PubSub keinen vergleichbaren Mechanismus an, um die Stabilität der zyklischen Kommunikation zu überwachen. Innerhalb der zyklischen Kommunikation werden die APDU-Daten sowie der APDU-Status-Informationen zwischen FAL APs ausgetauscht. Das heißt, den FAL APs liegen Status-Informationen über die Kommunikation vor, die sie überwachen und worauf sie, z. B. im Fehlerfall, reagieren.

In Abbildung 6.17 ist das Grundkonzept der Kommunikationsüberwachung für einen der Anwendungsfälle aus Kapitel 2.1 beschrieben. Bei diesem Anwendungsfall werden zyklisch Prozessdaten zwischen zwei Geräten, einem Closed-Loop-Steuerungssystems entsprechend, bidirektional ausgetauscht. Dazu erhält der Publisher A den Status des Subscribers A und sendet die Prozessdaten und den erhaltenen Status an das Gerät 2. Damit der Subscriber auf den Status reagieren kann, empfängt er die Nachricht und evaluiert den Status. Anhand des Status können Rückschlüsse über die Qualität der Kommunikation gezogen werden. Gleichzeitig übergibt der Subscriber B seinen Status an den Publisher B, der die Prozessdaten und den Status an das Gerät 1 sendet. Daraufhin ist in allen Geräten die Qualität der Kommunikation bekannt, um im Fall eines Kommunikationsausfalls mit einer geeigneten Fehlerreaktion darauf reagieren zu können.

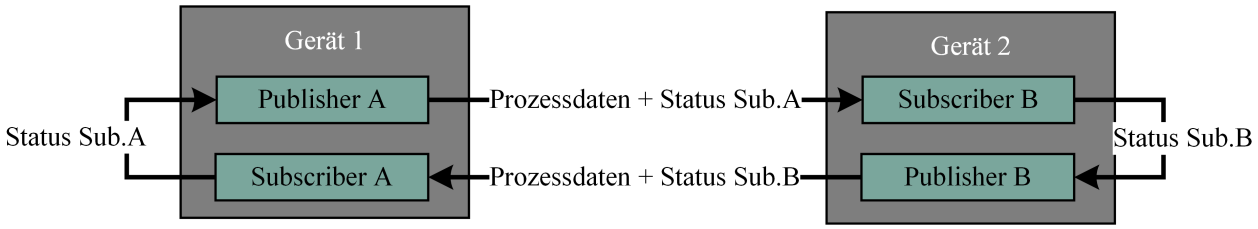


Abbildung 6.17: Anwendungsfall einer bidirektionalen zyklischen Kommunikation.

6.2.2 Analyse der OPC UA PubSub-Entitäten und Informationsmodelle

Üblicherweise besteht eine OPC UA PubSub-Applikation aus mehreren Entitäten, die jeweils ihre eigenen ObjectTypes im OPC UA-Informationsmodell besitzen. Die Entitäten bieten

Funktionen an, wie z. B. im Fall des Publishers und Subscribers das Senden und Empfangen von Ethernet-Frames. Eine solche logische Trennung der Funktionalitäten und Verteilung auf unterschiedliche Entitäten ist bereits von den Busprotokollen PROFINET und Sercos III bekannt. Bei PROFINET und Sercos III wird bereits die Erstellung und Übertragung von Frames voneinander entkoppelt. Ziel ist es, eine Entität zu identifizieren, die die Aufgabe der Überwachung des Datenaustauschs übernehmen kann. Dazu werden zunächst die OPC UA-Informationsmodelle erklärt und anschließend analysiert. Die Instanz *PublishSubscribe* dient als ein Einstiegspunkt im OPC UA-Server. Über die *HasComponent*-Referenz können 0..n *PubSubConnection*-Instanzen vom Typ *PubSubConnectionType* referenziert werden.

Der PubSubConnectionType ist vereinfacht in Abbildung 6.18 dargestellt. Der *PubSubConnectionType* beschreibt die Verbindung zwischen Publisher und Subscriber und beinhaltet alle nötigen Adress- und Transporteinstellungen. Über die Adresseinstellungen können das Netzwerkinterface und z. B. die IP-Adresse eingestellt werden.

Während der Laufzeit sendet der konfigurierte Publisher seine NetworkMessage an die eingestellte IP-Adresse. Außerdem beinhaltet der *PubSubConnectionType* noch eine weitere Property namens *PublisherId* zur Identifikation des Publishers. Hierbei handelt es sich um eine eindeutige ID in der Middleware, um den Publisher identifizieren zu können.

Über den *PubSubStatusType* wird der aktuelle Zustand der *PubSubConnection* abgebildet. Der *PubSubDiagnosticsConnectionType* beschreibt Diagnose-Informationen über die *PubSubConnection*.

Aus der Perspektive der Übertragung ist es möglich, die Einstellungen für eine überwachte Übertragung hier zu ergänzen. Die Überwachung könnte den UADP-*NetworkMessageHeader* überwachen und entsprechend auf Unregelmäßigkeiten reagieren. Auf diese Weise kann z. B. die *SequenceNumber* oder die *PublisherId* der empfangenen NetworkMessage überwacht werden. Die *PublisherId* ist jedoch nicht die einzige Id zur Identifikation der Daten. Da es sich bei einer NetworkMessage um einen Summenrahmen handeln kann, wird vielmehr die Kombination aus *PublisherId*, *WriterGroupId* und *WriterId* zur Identifikation der Daten genutzt.

Bei konsequenter Durchführung dieses Ansatzes müssen der *WriterGroupType* und *ReaderGroupType* sowie der *WriterType* und *ReaderType* angepasst werden. Dadurch würde die Anzahl der nötigen Anpassungen am OPC UA-Informationsmodell jedoch stark steigen. Deshalb wird nun die nächsttiefere Ebene im Informationsmodell betrachtet.

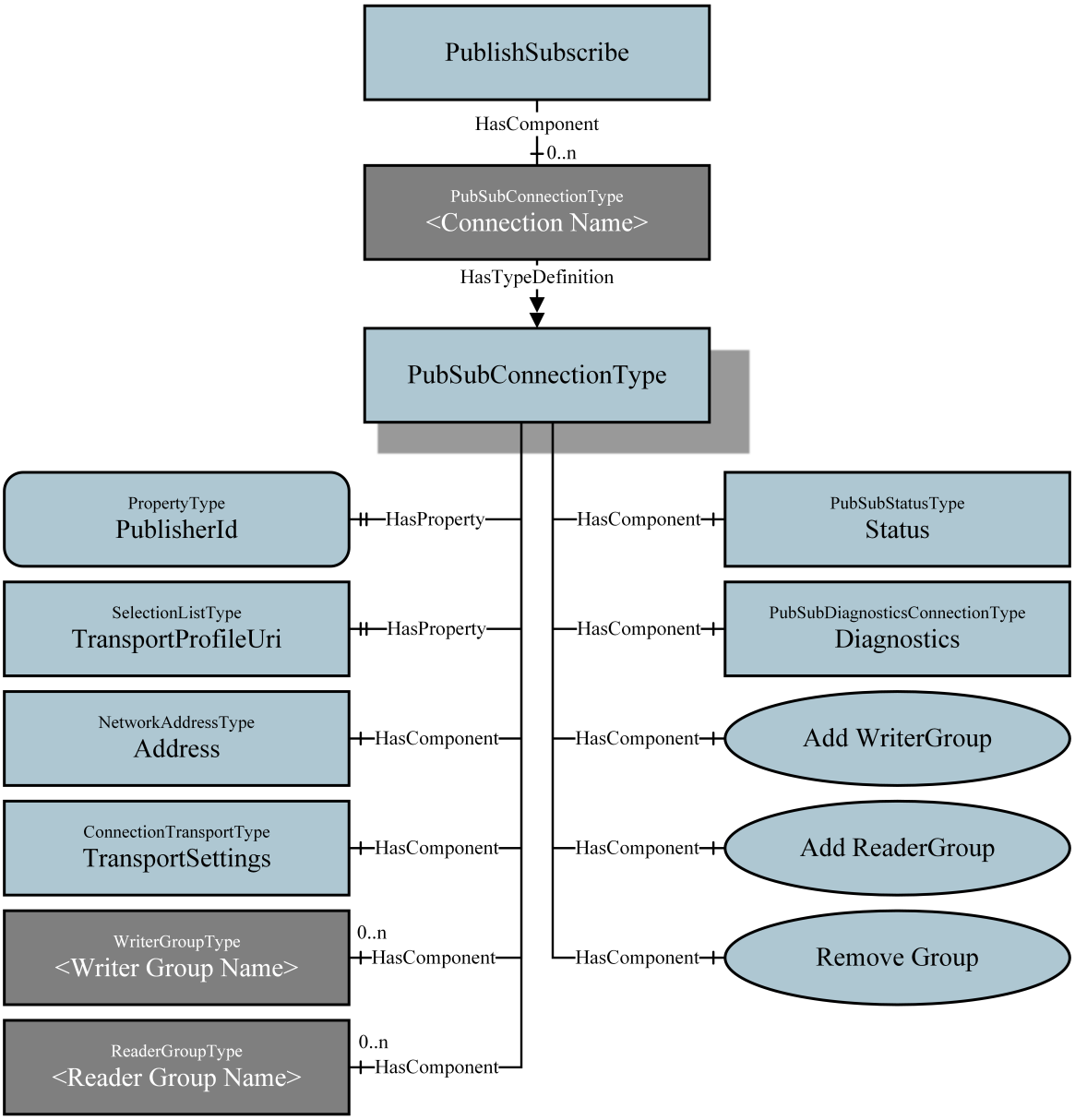


Abbildung 6.18: PubSubConnectionType.

Der WriterGroupType ist ein ObjectType, der dazu da ist, die DataSetWriter mit gleichen Einstellungen zu gruppieren. Der WriterGroupType ist in Abbildung 6.19 dargestellt. In den Einstellungen wird festgelegt, welches Transportprotokoll und Message Mapping für die NetworkMessage verwendet wird. In den *TransportSettings* kann zwischen dem brokerlosen Modell oder dem brokerbasierten Modell ausgewählt werden. Die *MessageSettings* beschreiben das Message Mapping der NetworkMessage. Es kann zwischen dem Message Mapping UADP und JSON ausgewählt werden. Außerdem können gruppenspezifische Einstellungen bei den *TransportSettings* oder *MessageSettings* vorgenommen werden. Das *PublishingInterval* ist vom

Typ *Duration* und gibt das Intervall an, in dem der Publisher seine NetworkMessages versendet. Wenn die DataSetWriter, die über die *WriterGroup* gruppiert werden, keine Daten in die NetworkMessage schreiben, kann trotzdem eine KeepAlive-Nachricht versendet werden. Der Mechanismus der KeepAlive-Nachrichten ist ein erster Ansatz für eine Überwachung des Datenaustauschs, allerdings nur auf der Seite des Subscribers. Der *Status* vom *PubSubStatusType* spiegelt den Status der Zustandsmaschine wider. Die Komponente *Diagnostics* vom *PubSubDiagnosticsConnectionType* bietet spezifische Diagnose-Informationen über die *WriterGroup* an.

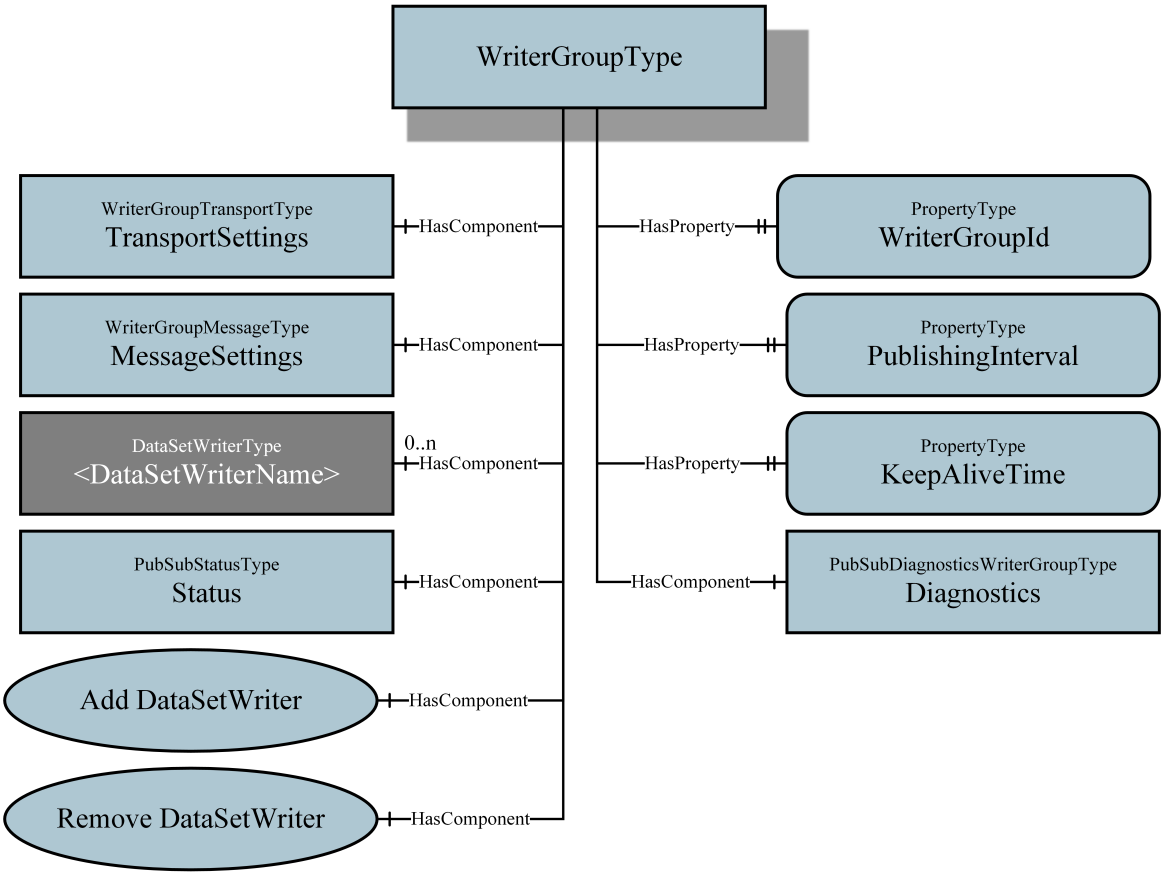


Abbildung 6.19: WriterGroupType.

Der ReaderGroupType ist ein ObjectType, der dazu da ist, die DataSetReader mit gleichen Einstellungen zu gruppieren. Der *ReaderGroupType* ist in Abbildung 6.20 dargestellt. Wie auch schon bei der *WriterGroup* können das Transportprotokoll und das Message Mapping konfiguriert werden. Die Komponente *Status* liefert Informationen über die Zustandsmaschine der *ReaderGroup*. Die Komponente *Diagnostic* bietet spezifische Diagnose-Informationen der *ReaderGroup*.

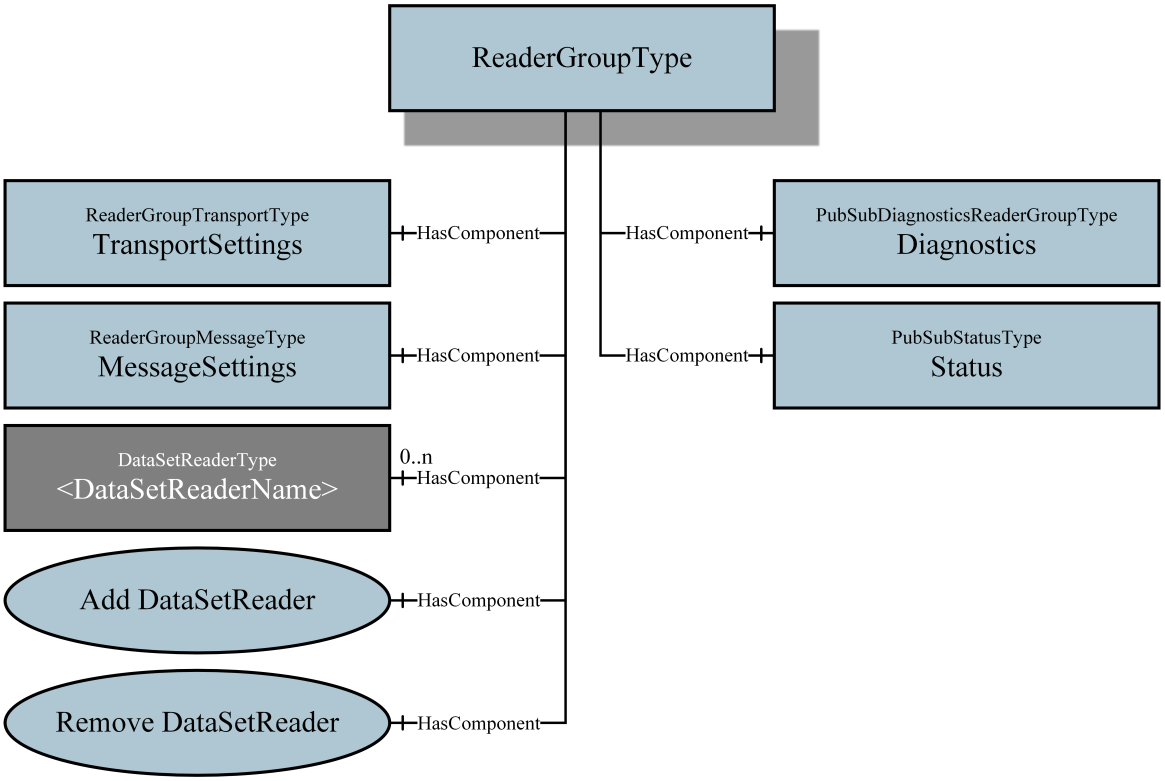


Abbildung 6.20: ReaderGroupType.

Die Hauptaufgaben der *ReaderGroup* und *WriterGroup* sind die Gruppierung der DataSetReader und DataSetWriter. Folglich sind die *ReaderGroup* und *WriterGroup* nicht als Einhängepunkt für die Konfigurationsparameter eines überwachten Datenaustauschs geeignet.

Der DataSetWriterType enthält die Einstellungen für einen DataSetWriter. Ein DataSetWriter ist die Entität, die die DataSetMessages enkodiert und in die NetworkMessage schreibt. Der *DataSetWriterType* ist in Abbildung 6.21 dargestellt. Es ist möglich, wie schon bei den vorherigen Typen, weitere Einstellungen am Transportprotokoll und Message Mapping vorzunehmen. Außerdem bieten die Komponenten *Status*- und *Diagnostics*-Informationen über den Zustand des DataSetWriters. Die *DataSetWriterId* ist die eindeutige ID eines DataSetWriters innerhalb einer NetworkMessage.

Der DataSetWriter eignet sich sehr gut als ein Einhängepunkt für eine Überwachung des Datenaustauschs, da er die Entität ist, die die DataSetMessages enkodiert und in die NetworkMessage schreibt. Dadurch hat der DataSetWriter direkten Zugriff auf die DataSetMessages und kann UADP so erweitern, dass eine Überwachung auf DataSetMessage-Ebene gewährleistet ist. Ein weiterer Punkt, der für diesen Einhängepunkt spricht, ist, dass dadurch eine NetworkMessage aus überwachten und nicht-überwachten DataSetMessages bestehen kann.

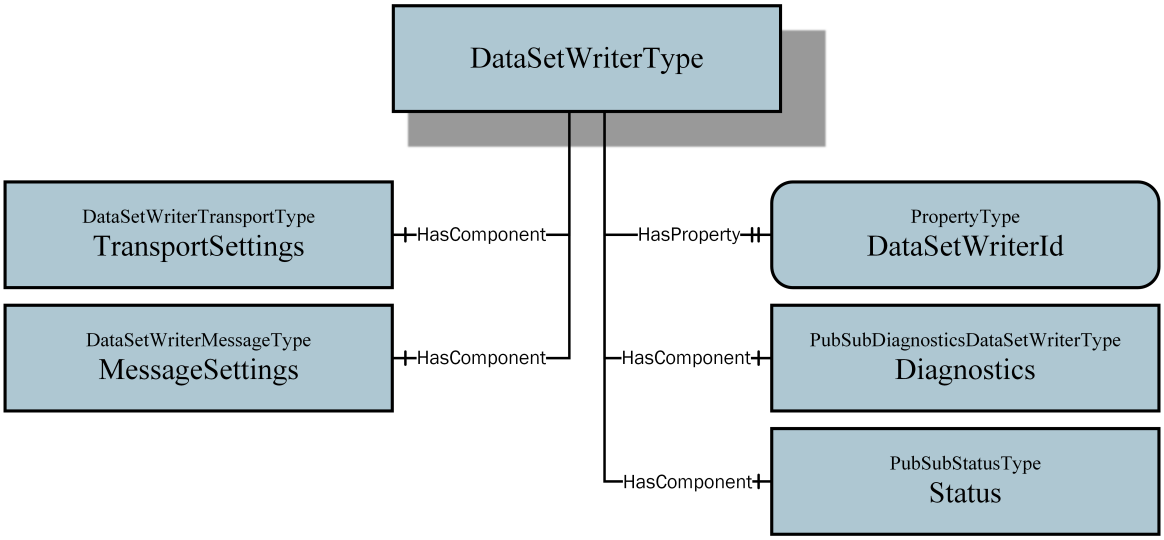


Abbildung 6.21: DataSetWriterType.

Der DataSetReaderType enthält die Einstellungen für einen DataSetReader. Der *DataSetReaderType* ist in Abbildung 6.22 dargestellt. Die Aufgabe des DataSetReaders ist es, die DataSetMessages entsprechend der vorgenommenen Einstellungen zu dekodieren. Die Einstellungen sind dazu in der Property *DataSetMetaData* gespeichert. Der *SubscribedDataSetType* stellt die Referenz zwischen den empfangenen DataSets und den DataSets eines OPC UA-Servers dar.

Der DataSetReader kann über die *PublisherId*, *WriterGroupId* und *WriterId* überprüfen, ob die empfangenen DataSetMessages für ihn relevant sind. Nicht relevante DataSetMessages werden ignoriert. Die *DataSetFieldContentMask* beschreibt, wie die *DataSetFields* enkodiert sind. Die Property *MessageReceiveTimeout* ist die maximal akzeptable Dauer zwischen zwei empfangenen DataSetMessages. Beim Überschreiten dieser Dauer wechselt der DataSetReader in den Fehlerzustand *Error*. Für diesen Mechanismus ist es nicht relevant, ob die DataSetMessage eine KeepAlive-Nachricht ist oder wirklich ein enkodiertes DataSet beinhaltet.

Der DataSetReader ist, abstrakt gesehen, das Gegenstück zum DataSetWriter. Die bisherige Standardisierung von OPC UA PubSub hat auch schon hier mit dem *MessageReceiveTimeout* eine gewisse Überwachung des Empfangs von DataSetMessages standardisiert. Dieser Mechanismus ist aber auf die Empfängerseite beschränkt. Die Folge ist, dass dem Publisher der Zustand des Subscribers nicht bekannt ist.

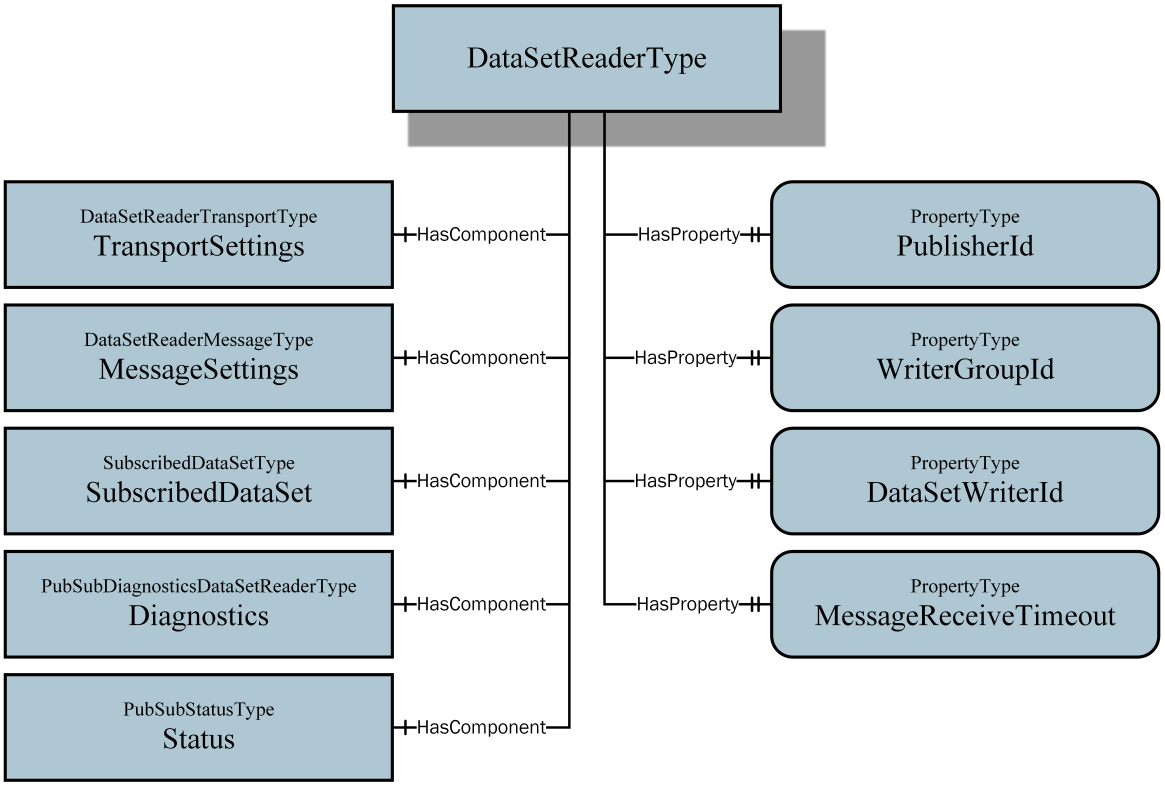


Abbildung 6.22: DataSetReaderType.

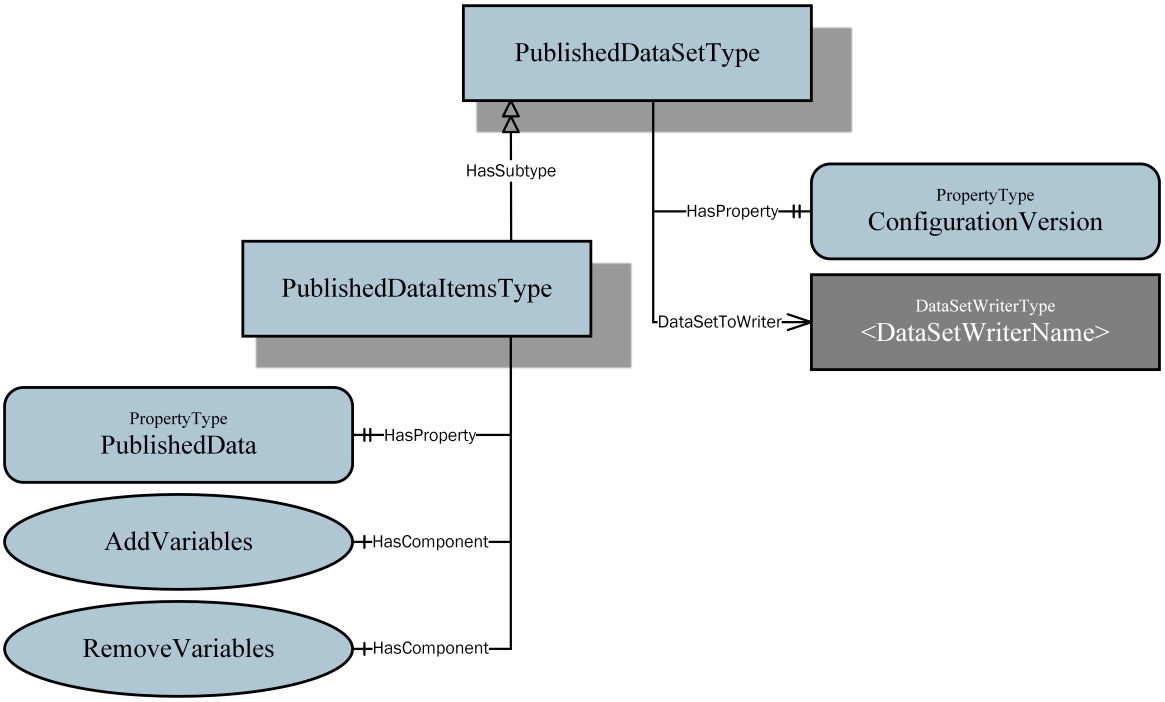


Abbildung 6.23: PublishedDataSetType.

Der PublishedDataSetType hat die Aufgabe, die Variablen in einer DataSet zu gruppieren und über die Referenz DataSetToWriter eine logische Verbindung zu einem DataSetWriter herzustellen. Der *PublishedDataSetType* ist in Abbildung 6.23 dargestellt. Über die Methoden *AddVariables* und *RemoveVariables* können Variablen zum *PublishedDataSet* hinzugefügt und entfernt werden.

6.2.2.1 Zusammenfassung

Basierend auf der dargelegten Analyse der Entitäten und deren Informationsmodelle kann nun die Frage – *Welche Entitäten sind beteiligt?* – beantwortet werden. Dazu wird die Entscheidung getroffen, die DataSetWriter und DataSetReader für die Datenaustausch-Überwachung auszuwählen. Die Hauptgründe hierfür sind:

1. Die DataSetWriter und DataSetReader haben über die Enkodierung und Dekodierung von DataSetMessages direkten Zugriff auf die DataSetMessages.
2. Es ist die Überwachung einzelner DataSetMessages möglich, was wiederum die Flexibilität erhöht.
3. Es ist bereits ein Mechanismus zur Überwachung einer DataSetMessage auf der Empfängerseite vorhanden. (*MessageReceiveTimeout*)

6.2.3 Interaktion der beteiligten Entitäten

Im vorherigen Kapitel 6.2.2 wird hergeleitet, dass der Datenaustausch über die Entitäten DataSetWriter und DataSetReader überwacht werden soll. Durch die Festlegung der Entitäten, die die Überwachung übernehmen, wird in einem nächsten Schritt überprüft, ob dadurch die geforderte Flexibilität erfüllt wird. Die Anforderung nach Flexibilität verlangt, dass der überwachte Datenaustausch parallel zum Standard-Datenaustausch im UADP-Protokoll betrieben werden kann. Dadurch können Geräte, die keinen überwachten Datenaustausch unterstützen, trotzdem parallel eingesetzt werden. Des Weiteren wird eine 1-zu-1- und 1-zu-N-Kardinalität, mit jeweils einem uni- als auch bidirektionalen Datenaustausch, gefordert. Die Flexibilität von UADP soll durch die Überwachung nicht eingeschränkt werden. Um die Anforderungen nach Flexibilität zu erfüllen, wird nachfolgend gezeigt, wie die Entitäten interagieren. In Abbildung 6.24 ist der überwachte Datenfluss zwischen zwei Geräten mit einer 1-zu-1-Kardinalität dargestellt. Eine Connection unterstützt wahlweise eine Uni-/Multi-/Broadcast-Adressierung. Die DataSetWriter erstellen eine DataSetMessage, die aus den Daten, sowie aus dem Status des eigenen DataSetReaders besteht. Der DataSetReader des anderen Geräts ist dann in der Lage, beim

Empfangen der DataSetMessage, den übertragenen Status des DataSetReaders auszuwerten. Dadurch ist dem *DataSetWriter A.W1* bekannt, ob seine DataSetMessages vom *DataSetReader B.R1* empfangen werden. Im Fall einer Kommunikationsstörung können der DataSetWriter und der DataSetReader entsprechend darauf reagieren. Die Übertragung von Daten in der DataSetMessage ist optional. Dadurch, dass es sich um eine entkoppelte Kommunikation handelt, können weitere Geräte, z. B. Gerät C, die DataSetMessage von *DataSetWriter A.W1* empfangen und auslesen. Es ist außerdem möglich, innerhalb einer NetworkMessage, überwachte DataSetMessages und nicht-überwachte DataSetMessages (A.W2 zu C.R1) zu kombinieren.

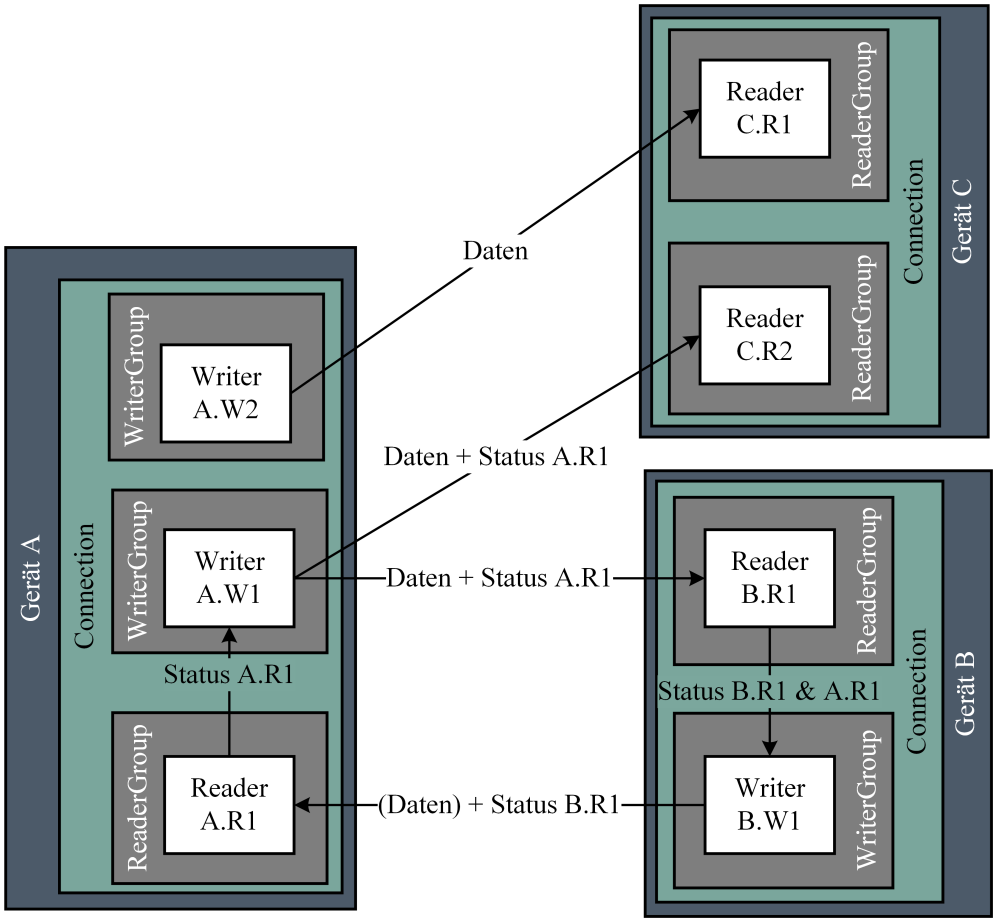


Abbildung 6.24: Kombination aus überwachten und nicht-überwachten Verbindungen.

In Abbildung 6.25 ist der Datenfluss mit einer 1-zu-N-Kardinalität mit N=2 dargestellt. Das Gerät A überträgt Daten an die Geräte B und C. Im Fall einer Kommunikationsstörung muss die Applikation entsprechend reagieren können. Dazu schreibt der *DataSetWriter A.W1* die Daten und den Status der *DataSetReader A.R1* und *A.R2* in die DataSetMessage. Die Geräte B und C empfangen die DataSetMessage und werten sie aus. Der jeweilige DataSetWriter der Geräte B und C schreibt den Status des eigenen DataSetReaders in die DataSetMessage und überträgt

die NetworkMessage an das Gerät A. Durch die Übertragung aller Status-Informationen können die Entitäten Kommunikationsstörungen identifizieren und entsprechend darauf reagieren.

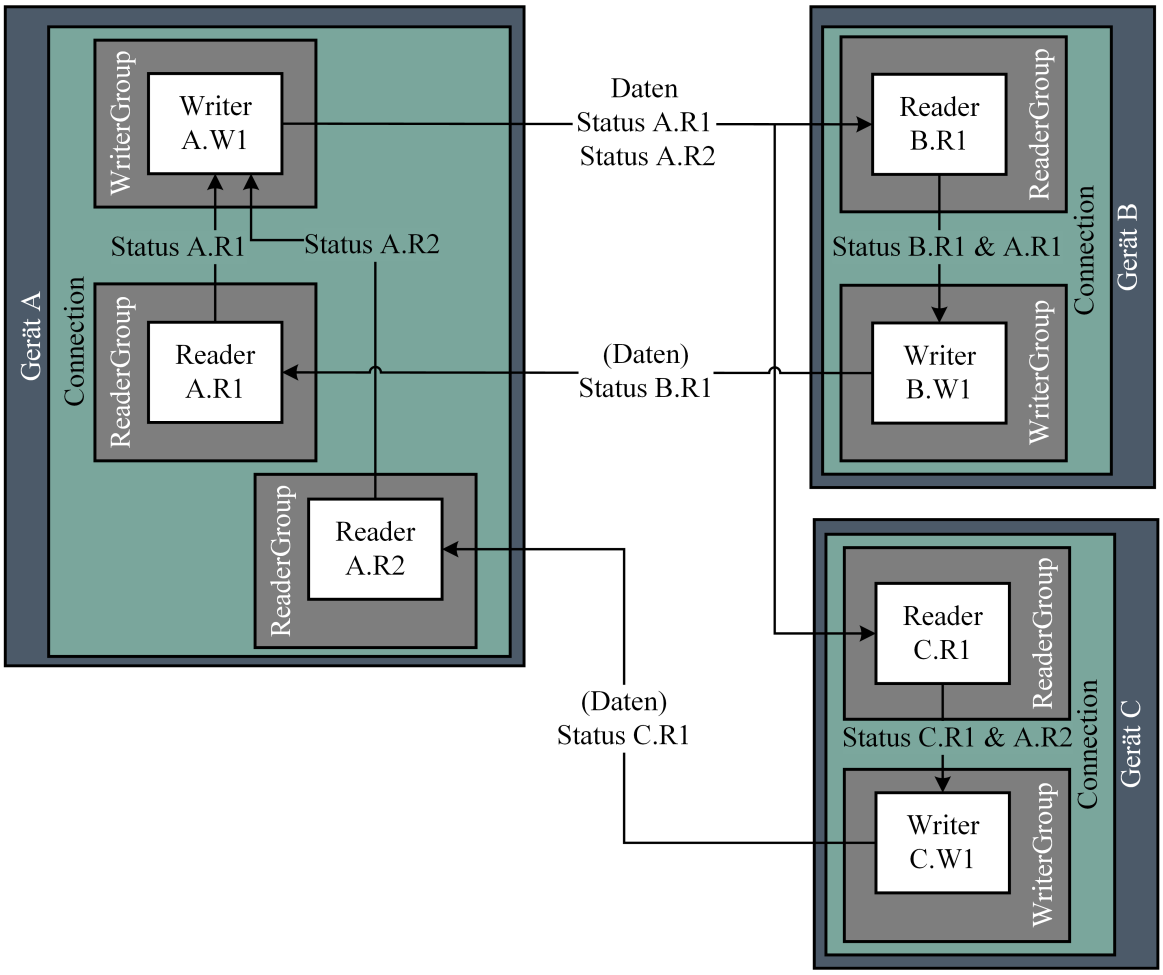
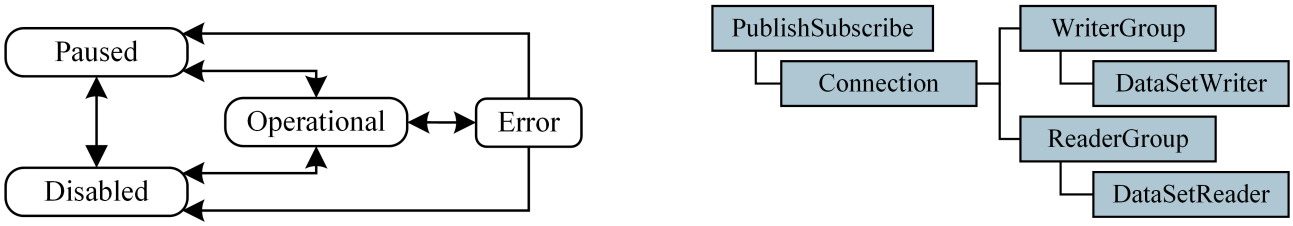


Abbildung 6.25: Drei überwachte 1-zu-N-Verbindungen.

6.2.4 Erweiterung der Zustandsmaschinen

Wie im Kapitel 3 Stand der Technik gezeigt, bietet OPC UA bereits die Möglichkeit, Zustandsmaschinen in einem Informationsmodell zu modellieren. Des Weiteren definiert OPC UA PubSub eine Zustandsmaschine, dargestellt in Abbildung 6.26(a), für die ObjectTypes *PublishSubscribe*, *PubSubConnectionType*, *WriterGroupType*, *ReaderGroupType*, *DataSetWriterType* und *DataSetReaderType*. Die Zustandsmaschine besteht aus den vier Zuständen *Disabled*, *Paused*, *Operational* und *Error*. Die Zustandsmaschinen sind, wie in Abbildung 6.26(b) dargestellt, über eine Parent/Child-Referenz miteinander verkettet.



(a) *Zustandsmaschine der Entitäten.* **(b)** *Verkettung der Zustandsmaschinen.*

Abbildung 6.26: OPC UA PubSub-Zustandsmaschinen.

Zur Überwachung des Datenaustauschs wird die Zustandsmaschine der DataSetWriter und DataSetReader erweitert. Dazu wird eine Unter-Zustandsmaschine für den Zustand *Operational* des DataSetWriters und DataSetReaders definiert. Die Analyse und Abstraktion der Busprotokolle zeigt, dass es Gemeinsamkeiten und Unterschiede bei den Zustandsmaschinen gibt. Nachfolgend sind die Gemeinsamkeiten und Unterschiede sowie ihre Relevanz für OPC UA PubSub erklärt und diskutiert.

Init: Dies ist ein Zustand, der die Zustandsmaschine initialisiert und darauf wartet, dass die Parent-Zustandsmaschine einen bestimmten Zustand erreicht. Bei Sercos III wird die Initialisierung verlassen, sobald die Hochlaufphase 4 erreicht ist. Um bei PROFINET die Initialisierung zu verlassen, muss die Verbindung erfolgreich aufgebaut und die Parametrierung abgeschlossen sein.

Prepare: Dieser Zustand wird nach erfolgreicher Transition aus dem Initialisierungszustand erreicht. Es werden die benötigten lokalen Variablen mit Standardwerten überschrieben und die Zustandsmaschine für den Betriebszustand vorbereitet.

Waiting: Dies ist ein Zustand des Empfängers, der darauf wartet, zum ersten Mal Daten zu erhalten. In diesem Zustand wird so lange gewartet, bis entweder Daten empfangen werden, zum Zustand *Stop* gewechselt wird oder der DataSetReader den *Operational* Zustand verlässt.

Receiving: Mit Receiving wird ein Zustand einer zyklischen Übertragung beschrieben, bei der in jedem Zyklus neue Daten erwartet werden. Bei PROFINET wird dieser Zustand nach einem erfolgreichen CONNECT und bei Sercos III über die Hochlaufphase CP3 erreicht. Ein solcher Zustand ist per se noch nicht in der Zustandsmaschine von OPC UA PubSub enthalten und wird über die Unter-Zustandsmaschinen erreicht.

Application Ready: Ist ein Zustand, der aussagt, dass die Applikation betriebsbereit ist. Ein derartiger Zustand ist bei PROFINET bereits integriert, ist aber noch nicht Bestandteil der Zustandsmaschine von OPC UA PubSub.

Consuming: Ist ein Zustand, in dem die empfangenen Daten verarbeitet werden. *Consuming* ist der Zielzustand der Zustandsmaschine des Empfängers. Solch ein Zustand ist bei beiden Busprotokollen PROFINET und Sercos III vorzufinden.

Producing: Ist ein Zustand, in dem die Daten produziert werden. Dabei handelt es sich um den Zielzustand der Zustandsmaschine des Senders. Solch ein Zustand ist bei beiden Busprotokollen PROFINET und Sercos III vorzufinden.

Warning: Wird ein Zustand bezeichnet, in dem eine Warnung vorliegt. Dabei handelt es sich um einen wichtigen Zustand, um Unregelmäßigkeiten im Betrieb zu signalisieren. Bei Sercos III ist dieser Zustand direkt in der Consumer-Zustandsmaschine integriert.

Error: In diesem Zustand werden Fehler signalisiert. Dieser Zustand wird in der Regel nur durch das Zurücksetzen des Fehlers durch den Bediener wieder verlassen.

Stop: Ist ein Zustand zum manuellen Stoppen der Zustandsmaschinen. Der Zustand ist von allen Zuständen aus erreichbar – mit Ausnahme des Fehlerzustands.

6.2.4.1 Unter-Zustandsmaschine eines DataSetWriters

In Abbildung 6.27 ist die Unter-Zustandsmaschine des DataSetWriters dargestellt und wird nachfolgend erklärt. Solange der DataSetWriter nicht im Zustand *Operational* ist, bleibt die Unter-Zustandsmaschine im Zustand *Init*. Jeder Zustand besitzt eine eigene OP_State ID, über die der Zustand identifiziert wird. Wechselt der DataSetWriter in den Zustand *Operational*, wird der *Init*-Zustand (0) verlassen und in den *Prepare*-Zustand (1) gewechselt. Außerdem wird die bereits vorhandene *DataSetMessage.Valid*-Flagge auf *False* gesetzt. Die Flagge sagt aus, ob die DataSetMessage gültig ist.

Über ein externes Event oder eine Methode wird die Transition *Start-Creating* ausgelöst. Es erfolgt ein Zustandswechsel von Zustand *Prepare* (1) auf Zustand *Creating* (2). Im Zustand *Creating* (2) werden die Zustände der eigenen DataSetReader, die an der Überwachung beteiligt sind, in die DataSetMessage eingefügt. Außerdem wird die *DataSetMessage.SequenceNumber* inkrementell um eins erhöht.

Der DataSetWriter verharrt so lange im Zustand *Creating* (2), bis alle DataSetReader im *OP_State* 4 sind und die Applikation betriebsbereit ist. Im nächsten Zustand *Producing* (3) werden weiterhin die Zustände der DataSetReader in die DataSetMessage eingefügt und die Prozessdaten in die DataSetFields kopiert (insert process data). Außerdem wird die *DataSetMessage.Valid*-Flagge auf *True* gesetzt. Wenn ein DataSetReader in den Zustand 6 wechselt, ist die Bedingung der Transition erfüllt und der DataSetWriter wechselt in den nächsten Zustand

Producing with Warning (4). Der DataSetWriter verharrt im Zustand *Producing with Warning* (4), solange einer der DataSetReader im Zustand *Warning* ist. Wenn alle DataSetReader wieder im Zustand *Consuming* sind, wechselt der DataSetWriter zurück in den Zustand *Producing* (3).

Wenn einer der DataSetReader in den Zustand *Error* wechselt, wechselt der DataSetWriter in den Zustand *Error* (5). Erst nach dem Zurücksetzen des Fehlers über die Transition *Reset-Error* verlässt der DataSetWriter den Fehlerzustand und wechselt in den Zustand Prepare (1).

Über den weiteren Zustand *Stopped* (6) kann die Unter-Zustandsmaschine gestoppt werden. Es ist über die Transition *Start-Stopped* möglich, aus den Zuständen *Creating* (2), *Producing* (3) und *Producing with Warning* (4) in den Zustand *Stopped* zu wechseln. Die Transition *Start-Stopped* ist üblicherweise ein Methodenaufruf.

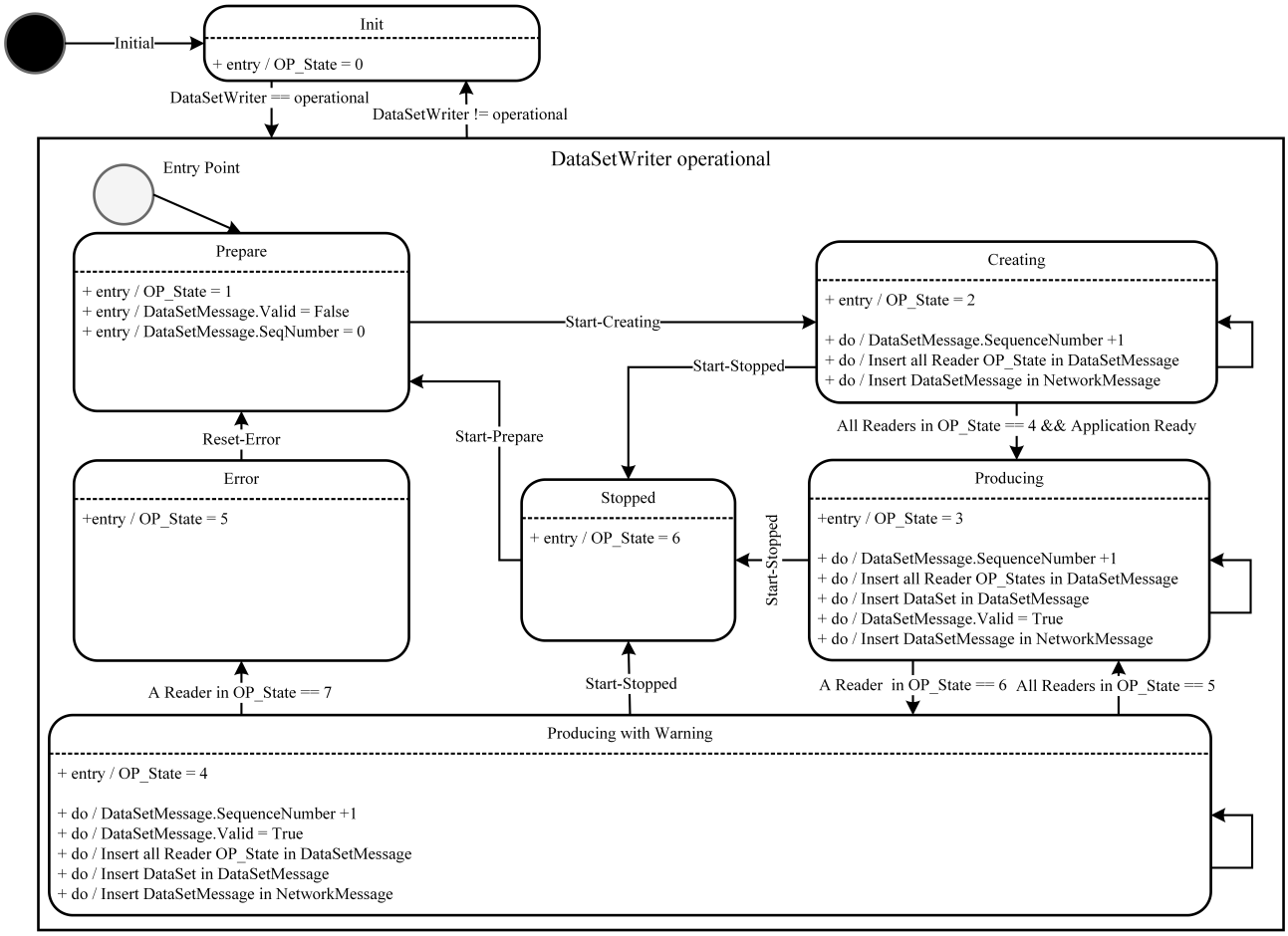


Abbildung 6.27: UML-Diagramm der Unter-Zustandsmaschine eines DataSetWriters.

6.2.4.2 Unter-Zustandsmaschine eines DataSetReaders

Die Unter-Zustandsmaschine ist eine Erweiterung der Standard-Zustandsmaschine des DataSetReaders und ist in Abbildung 6.28 dargestellt. Die Unter-Zustandsmaschine startet im Zustand *Init* (0). Sobald der DataSetReader in den Zustand *Operational* wechselt, wechselt die Unter-Zustandsmaschine von Zustand *Init* (0) auf Zustand *Prepare* (1). In diesem Zustand wartet die Unter-Zustandsmaschine auf die Transition *Start-Waiting*. Die Bedingung für die Transition ist z. B. ein externes Event oder ein Methodenaufruf.

Der nächste Zustand *Waiting to receive DataSetMessage* (2) überprüft zyklisch, ob eine DataSetMessage mit der richtigen *WriterId* empfangen wurde. Sobald der DataSetReader eine DataSetMessage mit der richtigen *WriterId* empfängt, wechselt die Unter-Zustandsmaschine in den Zustand *Receiving* (3). In diesem Zustand überprüft der DataSetReader zyklisch den Inhalt der *DataSetMessage.Valid*-Flagge und die *DataSetMessage.SequenceNumber*.

Sobald die Applikation betriebsbereit ist, verlässt der DataSetReader den Zustand *Receiving* (3) und wechselt in den Zustand *Receiving and Application ready to consume* (4). Im Zustand *Receiving and Application ready to consume* (4) evaluiert der DataSetReader die *DataSetMessage.Valid-Flagge*. Wenn die *DataSetMessage.Valid-Flagge* auf True gesetzt ist, wechselt der DataSetReader in den nächsten Zustand *Consuming* (5). Beim Eintritt in den Zustand *Consuming* (5) setzt der DataSetReader die Variable *Error counter of consecutive DataSetMessage losses* zurück. Die Variable *Error counter of consecutive DataSetMessage losses* ist ein Zähler der die hintereinander verlorenen Nachrichten zählt. Außerdem konsumiert der DataSetReader die Prozessdaten. Die Zustände *Receiving* (3), *Receiving and Application ready to consume* (4) und *Consuming* (5) besitzen eine Transition namens *Start-Warning*. Die Bedingung für diese Transition ist erfüllt, wenn der DataSetReader in einem Zyklus keine DataSetMessage mit richtiger *WriterId* empfängt. Wenn dies eintritt, wechselt die Unter-Zustandsmaschine des DataSetReaders in den Zustand *Warning* (6). Der DataSetReader kann aus dem Zustand *Warning* (6) zurück in die Zustände *Receiving* (3), *Receiving and Application ready to consume* (4) und *Consuming* (5) wechseln, wenn dafür die entsprechenden Bedingungen erfüllt sind.

Im Fall, dass der DataSetReader hintereinander in mehreren Zyklen keine DataSetMessages empfängt, wechselt die Unter-Zustandsmaschine in den Zustand *Error* (7). Über ein externes Event oder eine Methode kann der Fehler zurückgesetzt und in den Zustand *Prepare* (1) überführt werden. Über den weiteren Zustand *Stopped* (8) kann die Unter-Zustandsmaschine gestoppt werden. Es ist über die Transition *Start-Stopped* möglich, aus den Zuständen *Waiting* (2), *Receiving* (3), *Receiving and Application ready to consume* (4) und *Warning* (6) in den

Zustand *Stopped* (8) zu wechseln. Die Transition *Start-Stopped* ist üblicherweise ein Methodenaufruf.

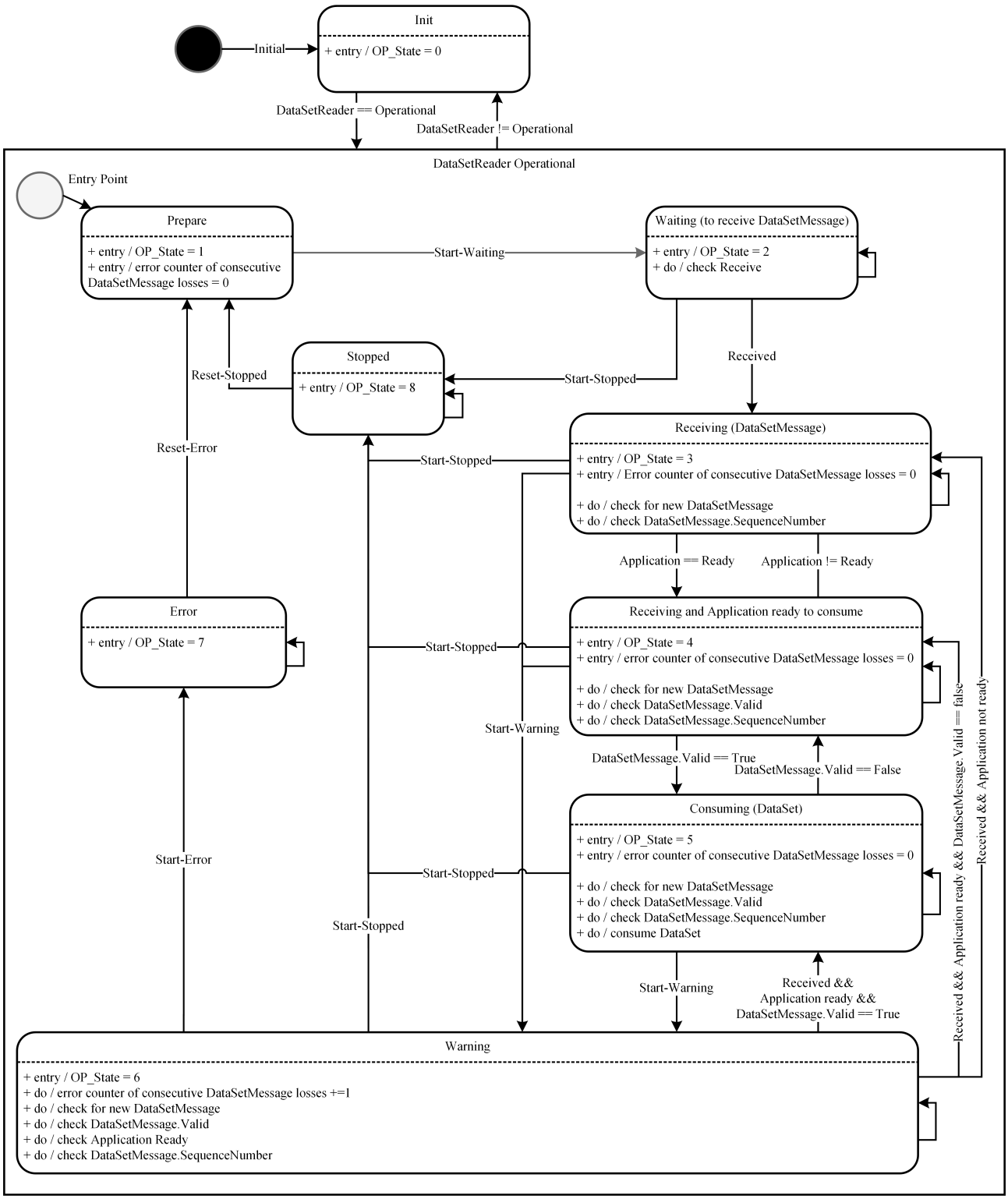


Abbildung 6.28: UML-Diagramm der Unter-Zustandsmaschine eines DataSetReaders.

6.2.5 UADP-Protokollerweiterung

Damit nicht mehrere inkompatible Lösungen entstehen, die auf unterschiedlicher Weise die Status-Informationen austauschen, ist es notwendig, das UADP-Protokoll entsprechend zu erweitern. Dazu wird zunächst eine geeignete Stelle im UADP identifiziert. Da jede DataSetMessage einzeln überwacht wird, ist der logische Schluss, dass die Status-Informationen direkt in die DataSetMessage eingefügt werden. Bisher besteht eine DataSetMessage aus einem DataSetMessage Header und den Daten. Im DataSetMessage Header werden über die *DataSetFlags1*-Flagge und die *DataSetFlags2*-Flagge Optionen – beispielsweise Zeitstempel oder Sequenznummern – aktiviert oder deaktiviert. Es sind bereits alle Bits der *DataSetFlags1*-Flagge vergeben, sodass das erste noch freie Bit das Bit 6 der *DataSetFlags2*-Flagge ist. Dieses Bit wird für die Aktivierung und Deaktivierung der Überwachung genutzt. Ob die DataSetMessage die zusätzlich notwendigen Felder für den Status-Austausch beinhaltet, ist über dieses Bit identifizierbar. Zur Position der Status-Informationen innerhalb der UADP-Protokollerweiterung kommen drei Möglichkeiten in Frage. Die Status-Informationen können, wie schon die *SequenceNumber*, eine Erweiterung des DataSetMessage Headers darstellen. Ein Gegenargument ist, dass die Status-Information bereits eigene Header-Informationen besitzt, was zu einer Verkapselung von Headern führt. Ein weiteres Gegenargument für diese Möglichkeit ist der große Umfang der Status-Informationen. Aus diesen Gründen wird diese Möglichkeit nicht ausgewählt.

Die zweite und dritte Möglichkeit platziert die Status-Informationen vor und hinter den Prozessdaten. Die Auswirkungen dieser Entscheidung auf die Leistung sind so gering, dass sie vernachlässigt werden. Da bereits die *SequenceNumber* vor den Prozessdaten angeordnet ist, macht es die DataSetMessage übersichtlicher, wenn die Status-Informationen sich diesem Trend anschließen und vor den Prozessdaten integriert werden. In Abbildung 6.29 ist zum einen die Position und zum andern der Inhalt der UADP-Protokollerweiterung dargestellt.

Bit 0 der *DataSetFlags1*-Flagge ist das, bereits in Kapitel 6.2.4 erwähnte Bit *DataSetMessage.Valid*. Das Bit sagt aus, ob die DataSetMessage gültig ist. Wenn Bit 6 der *DataSetFlags2*-Flagge auf True gesetzt ist, folgt nach der *SequenceNumber*, ebenfalls bekannt aus Kapitel 6.2.4, der neue *DataSetMessage Status*. Ein *DataSetMessage Status* besteht aus einem *ReaderCount* und einem oder mehreren *Reader Status*-Informationen. Ein *Reader Status* besteht aus einem *Reader Header*, der aktivierten IDs und der Status-Information. Die IDs sind nötig, um die Status-Information zu identifizieren. Der *Reader Status* kann folgendermaßen interpretiert werden: *Im Reader Status ist der OP_State des Readers, der NetworkMessages mit der PublisherId X, GroupId Y und WriterId Z empfängt.*

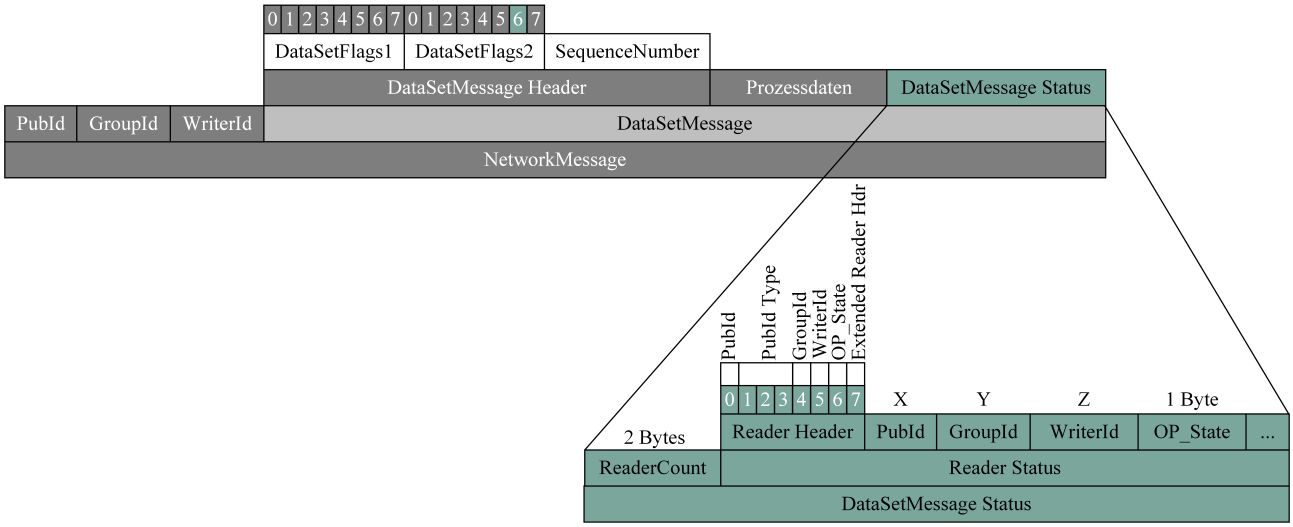


Abbildung 6.29: Protokollerweiterung zur Überwachung des Datenaustauschs.

6.2.6 Virtual Reader

In Abbildung 6.30 wird eine neue Entität beschrieben. Über diese Entität können die Status-Informationen eines DataSetReaders in einem OPC UA-Server dargestellt werden. In der Abbildung 6.30 tauschen die Geräte A und B Daten bidirektional miteinander aus. Dazu sendet der Publisher A eine NetworkMessage an das Gerät B. Die NetworkMessage enthält eine DataSetMessage, in die der DataSetWriter 1 die Daten für das Gerät B und die Status-Information über seinen DataSetReader in die DataSetMessage einfügt. Gerät B empfängt die NetworkMessage, die dann vom DataSetReader dekodiert wird. Die dekodierten Daten werden in dem *SubscribedDataSet 1* gespeichert und die empfangenen Status-Informationen werden im *Virtual Reader* abgebildet. Das bedeutet, dass der *Virtual Reader* in Gerät B ein virtuelles Abbild des DataSetReaders von Gerät A ist.

Ein *Virtual Reader* besitzt einen eigenen ObjectType und kann entsprechend in einem OPC UA-Server abgebildet werden. Der Rückkanal von Gerät B zu Gerät A folgt der gleichen Logik und wird demnach nicht weiter erklärt.

Die Referenz *HasVirtualDataSetReader* ist eine neue Referenz zwischen DataSetWriter und *Virtual Reader*, durch die der DataSetWriter weiß, auf welche *Virtual Reader* er bei der Überwachung des Datenaustauschs achten muss.

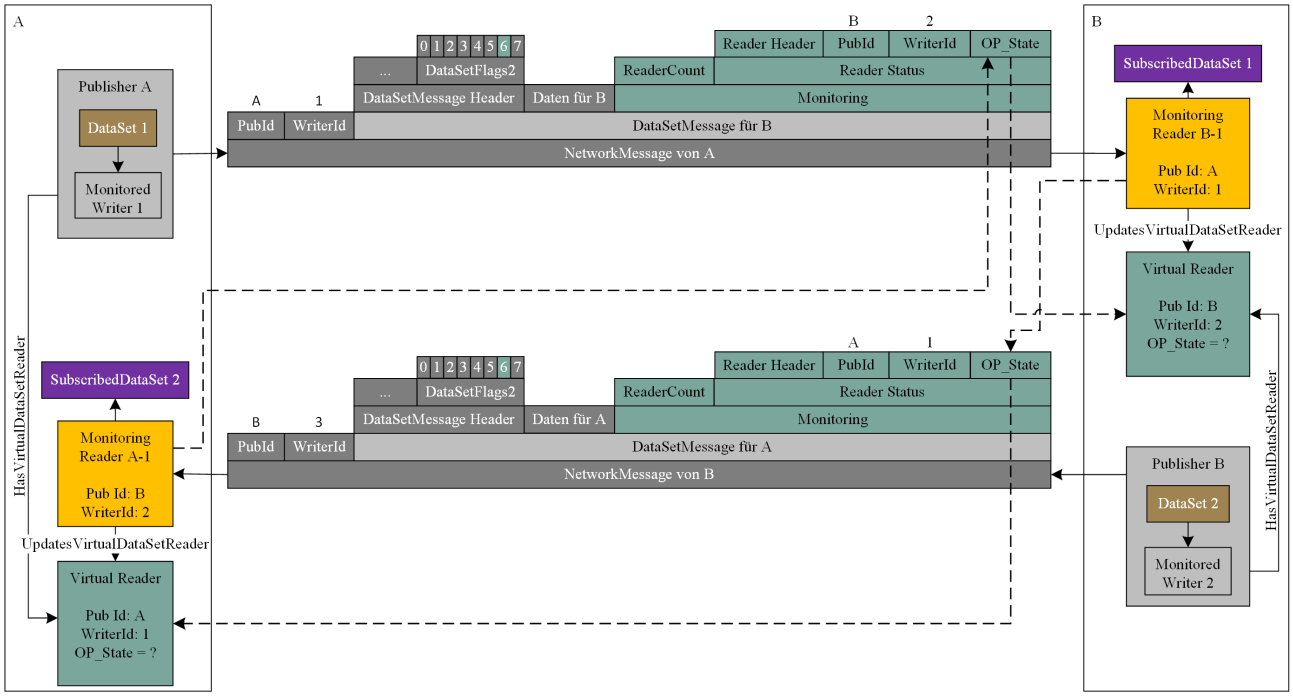


Abbildung 6.30: Funktionsweise eines Virtual Readers.

6.2.7 Informationsmodelle

Um den überwachten Datenaustausch in einem OPC UA-Server konfigurieren und anzeigen zu können, muss ein Informationsmodell erstellt werden. In Abbildung 6.31 ist eine Übersicht über die erstellten ObjectTypes sowie deren Referenzen zueinander dargestellt. Insgesamt werden fünf neue ObjectTypes und zwei neue Referenzen definiert.

Der *MonitoredPublishSubscribeType* leitet sich vom *PublishSubscribeType* ab und bietet die Möglichkeit, zusätzliche Komponenten wie den *MonitoredPubSubConnectionType* neben dem Standard-*PubSubConnectionType* zur Nutzung bereitzustellen. Der *MonitoredPubSubConnectionType* leitet sich vom *PubSubConnectionType* ab und besitzt, neben den geerbten Komponenten, die weiteren Komponenten *MonitoringReaderGroupType* und *MonitoredWriterGroupType*. Beide leiten sich von den Standard-Typen *ReaderGroupType* und *WriterGroupType* ab. Der Unterschied zu den Standard-Typen ist, dass sie die abgeleiteten Typen *MonitoringDataSetReaderType* und *MonitoredDataSetWriterType* als Komponenten besitzen. Die ObjectTypes *MonitoringDataSetReaderType* und *MonitoredDataSetWriterType* verwenden die neuen Referenzen *UpdatesVirtualDataSetReader* und *HasVirtualDataSetReader*, um mit dem neuen *VirtualDataSetReaderType* eine logische Verbindung aufzubauen. Die Referenz *UpdatesVirtualDataSetReader* wird vom *MonitoringDataSetReader* verwendet, um die empfangenen Status-Informationen in den *VirtualDataSetReader* zu schreiben. Die Referenz *HasVirtualDataSetReader* wird vom

MonitoredDataSetWriter genutzt, um auf den Status des referenzierten *VirtualDataSetReader* reagieren zu können.

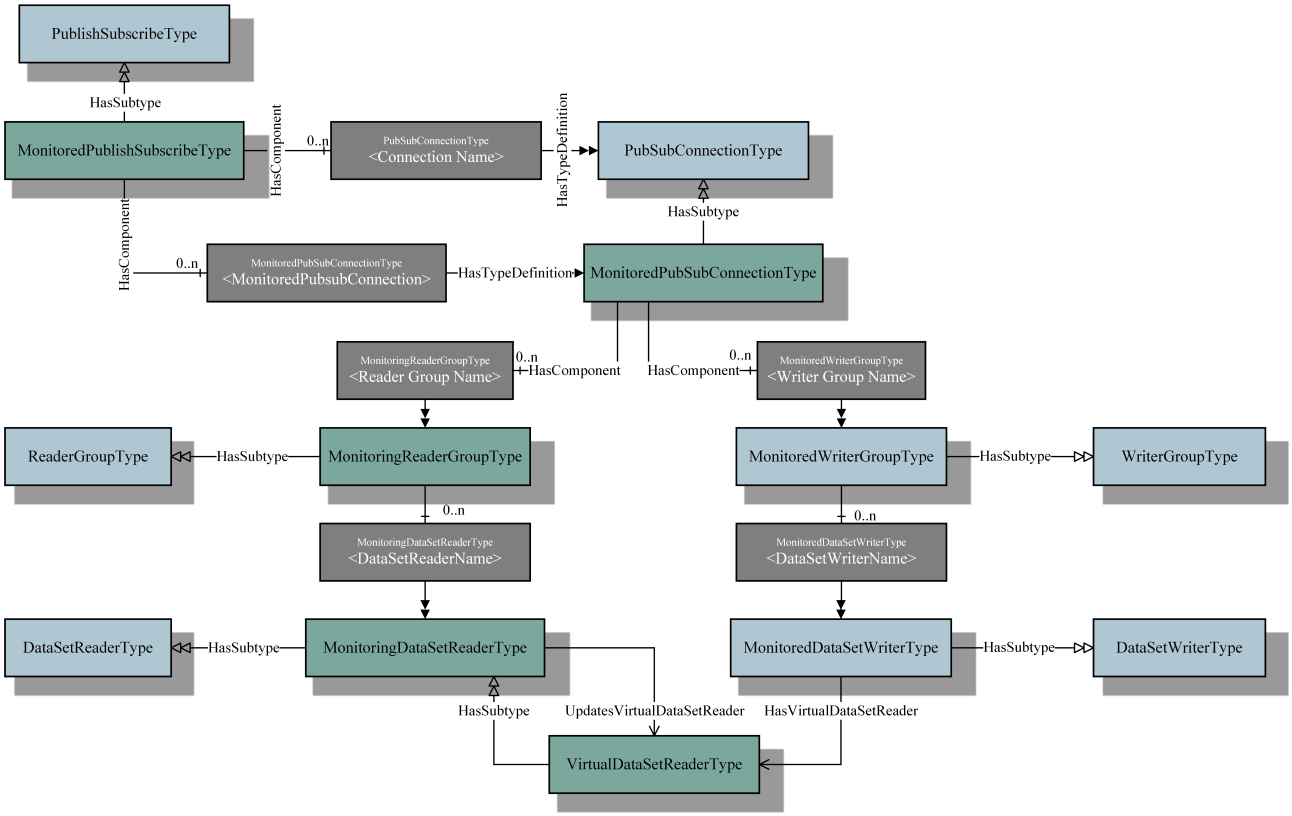


Abbildung 6.31: Übersicht über die Informationsmodelle.

6.2.8 Zusammenfassung

Die aus Unterkapitel 6.1 gewonnenen Erkenntnisse bilden die Basis für das Kapitel 6.2. In Kapitel 6 wurde ein überwachter Datenaustausch einer OPC UA PubSub basierten Kommunikation entwickelt. Dazu wurden die bisherigen Zustandsmaschinen um weitere Unter-Zustandsmaschinen erweitert. Die UADP-Struktur wurde um Status-Informationen ergänzt. Diese Status-Informationen können über einen bidirektionalen Datenaustausch über zwei Paare, bestehend aus Publisher und Subscriber, ausgetauscht werden. Die empfangenen Status-Informationen können über eine neu definierte Entität – ein Virtual Reader – im OPC UA-Server angezeigt werden. Darüber hinaus wurde das bisherige OPC UA PubSub-Informationsmodell erweitert, um zukünftig über die OPC UA-Client/Server-Architektur konfigurierbar zu sein.

7 Validierung

In diesem Kapitel 7 wird die Lösung validiert. Zunächst wird eine 1-zu-N-Kardinalität konkretisiert. Danach wird eine Simulation über Simulink durchgeführt. Abschließend wird ein Testaufbau implementiert, bei dem eine typische C2C-Anwendung realisiert wird. Anhand der Validierung wird überprüft, ob die Anforderungen aus Kapitel 2 erfüllt werden.

7.1 Konkretisierung einer 1-zu-N-Kardinalität

In Abbildung 7.1 sind insgesamt vier Geräte – A, B, C und D – vorhanden. Es wird eine 1-zu-N-Kardinalität mit N=3 abgebildet. Die Geräte A, B und C bestehen aus MonitoredWriter, MonitoringReader und Virtual Reader. Gerät D besteht ausschließlich aus einem DataSetReader. Die Geräte A, B und C verwenden einen überwachten Datenaustausch. Das Gerät D empfängt die Nachrichten von A ohne eine Überwachung des Datenaustauschs.

Dadurch wird zum einen veranschaulicht, wie 1-zu-N-Kardinalitäten realisiert werden können und zum anderen, wie überwachter und nicht-überwachter Datenaustausch parallel ausgeführt werden kann.

Des Weiteren wird in der Abbildung 7.1 nachvollziehbar, wie alle Entitäten logisch miteinander verknüpft sind und entsprechend konfiguriert werden müssen. Außerdem wird ersichtlich, wie die Status-Informationen über das erweiterte UADP zwischen den Entitäten verteilt werden.

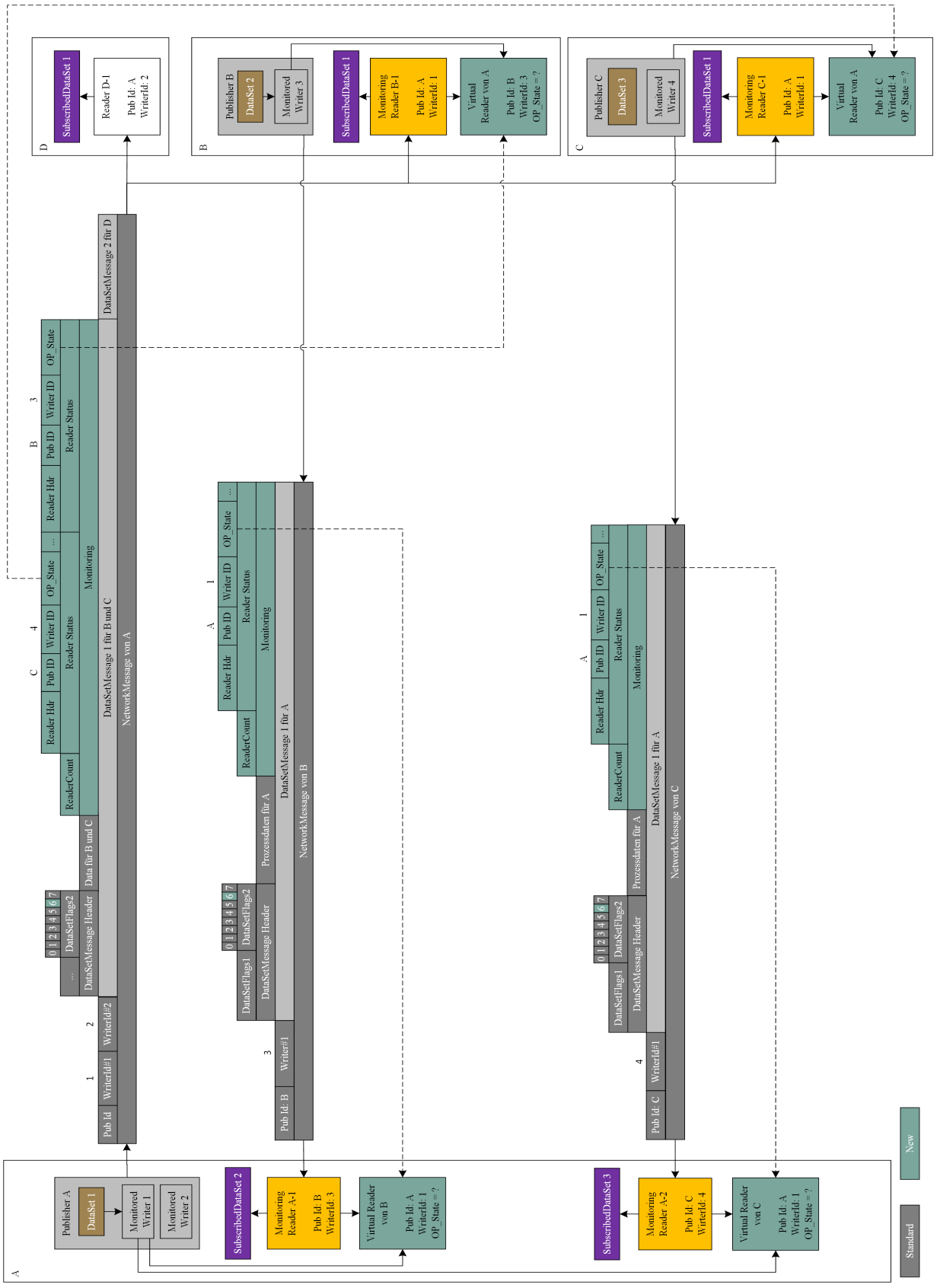


Abbildung 7.1: Protokollerweiterung zur Überwachung des Datenaustauschs.

7.2 Simulation über Simulink

Um das Konzept zu validieren, wird die Simulationssoftware Simulink verwendet. Simulink bietet die Möglichkeit, über sogenannte Stateflow-Modelle Entscheidungslogiken zu modellieren und zu simulieren. Entscheidungslogiken sind z. B. Zustandsmaschinen und Flussdiagramme. Die entworfenen Zustandsmaschinen der DataSetWriter und DataSetReader zur Überwachung des Datenaustauschs aus Kapitel 6.2.4 werden durch ein Stateflow-Modell modelliert und validiert. In Abbildung 7.2 ist das Sequenzdiagramm für den normalen Betrieb ohne Paketausfälle dargestellt.

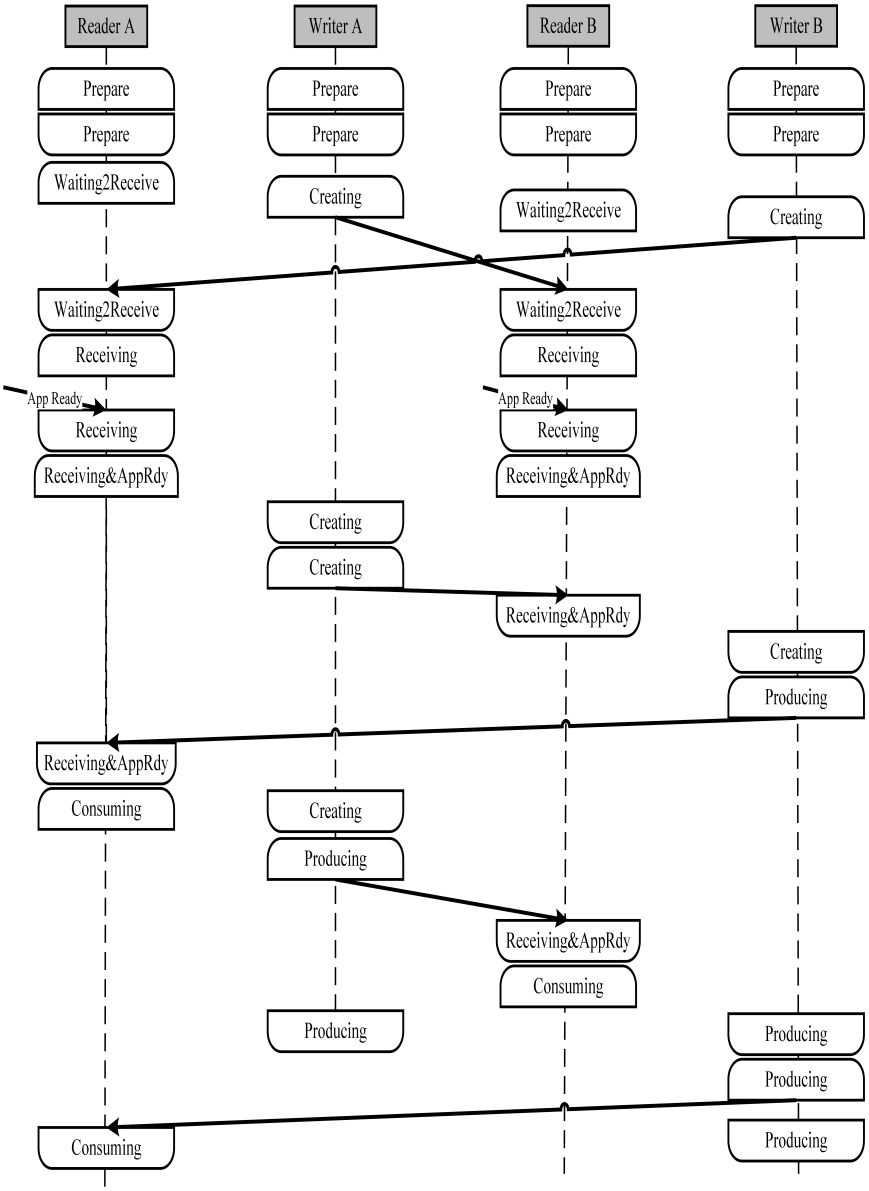


Abbildung 7.2: Sequenzdiagramm aus der Simulink-Simulation.

7.3 Testaufbau

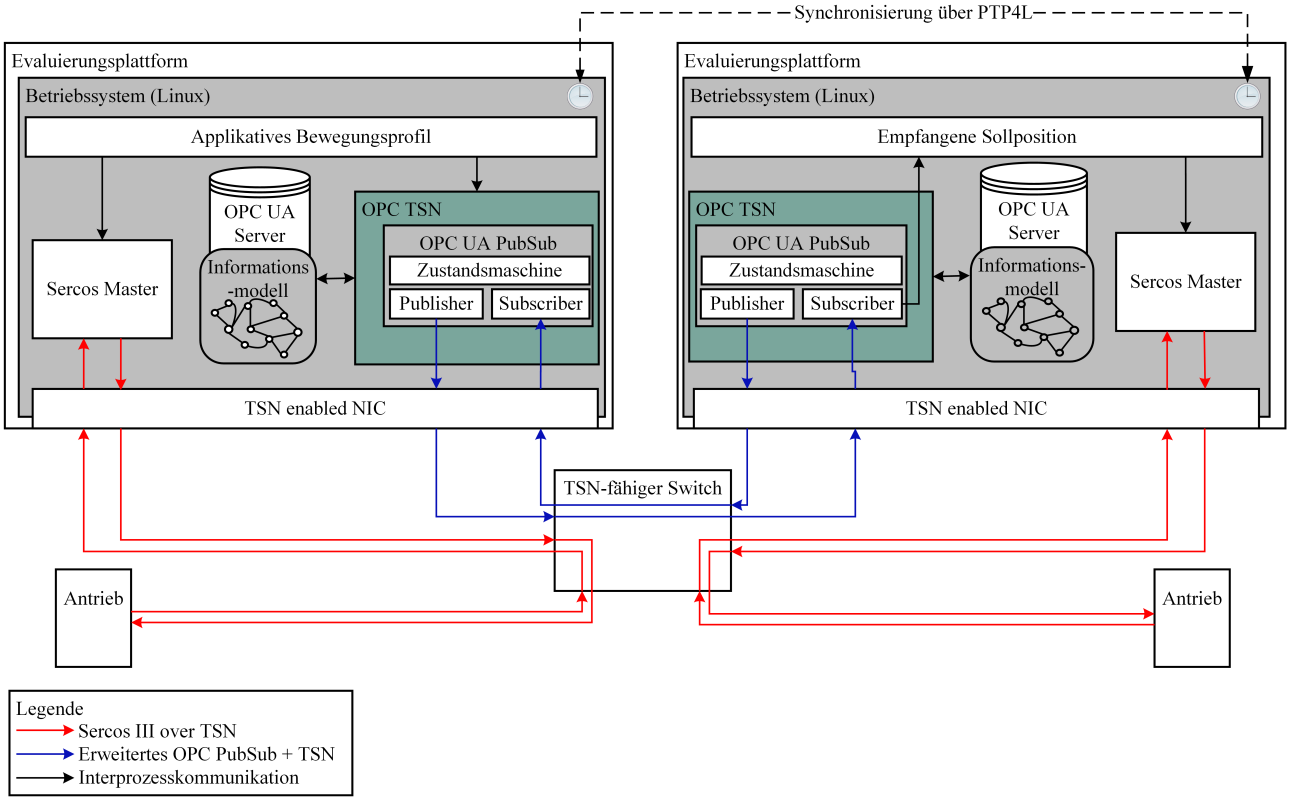


Abbildung 7.3: Architektur des Testaufbaus.

Um die Realisierbarkeit zu testen, wird ein Testaufbau entwickelt. Der Testaufbau basiert auf der Evaluierungsplattform aus Kapitel 4. Als Referenzapplikation wird eine Achskopplung über zwei Steuerungen ausgewählt. Die erste Steuerung sendet die Sollposition des nächsten Zyklus zur zweiten Steuerung. Die Sollposition wird dann von beiden Steuerungen im nächsten Zyklus aktiv geschaltet. In Abbildung 7.3 ist die Architektur des Testaufbaus dargestellt. Auf beiden Steuerungen ist ein Sercos III Master, ein OPC UA-Server, ein Publisher und ein Subscriber integriert. Alle Komponenten können die TSN-Funktionalitäten der Evaluierungsplattform nutzen. Sämtliche Kommunikation der Steuerungen – Sercos III over TSN, OPC UA PubSub und gPTP – wird über einen TSN-fähigen Switch geleitet. Die OPC UA PubSub-Komponenten nutzen die in dieser Arbeit erarbeiteten Erweiterungen des UADP-Protokolls, der Zustandsmaschine und des Informationsmodells. Die Kommunikation zwischen den Steuerungen wird zyklisch mit einer Zykluszeit von 1 ms ausgeführt. Das Gleiche gilt auch für die Kommunikation zwischen den Steuerungen und den Antrieben. Unregelmäßigkeiten beim Datenaustausch zwischen Publisher und Subscriber werden von den erweiterten Zustandsmaschinen erkannt.

Die Zustände der Zustandsmaschinen sind über die Virtual Reader in den integrierten OPC UA-Servern abgebildet.

7.4 Ergebnisse

Im Testaufbau wird gezeigt, dass der Austausch von Status-Informationen über die UADP-Erweiterung performant genug ist, um die Anforderungen von C2C-Anwendungsfällen zu erfüllen. Darüber hinaus wurde bewiesen, dass die entworfenen Zustandsmaschinen für die Überwachung des Datenaustauschs eingesetzt werden können. Ferner wurde bewiesen, dass die Status-Informationen über Virtual Readers in einem OPC UA-Server abgebildet werden können.

Zu Beginn dieser Arbeit wurden folgende Anforderungen an die Überwachung des Datenaustauschs gestellt:

Kompatibilität: Die Erweiterungen am OPC UA PubSub-Protokoll müssen kompatibel zum bisherigen OPC UA PubSub-Protokoll sein und gering gehalten werden. Im Testaufbau hat sich gezeigt, dass diese Anforderung erfüllt ist. Alle Erweiterungen am UADP, an den Zustandsmaschinen und den Informationsmodellen sind kompatibel zu Standard-OPC UA PubSub.

Flexibilität: Der parallele Einsatz eines überwachten Datenaustauschs zum standardmäßigen Datenaustausch im OPC UA PubSub-Protokoll ist eine Grundanforderung. Durch die Überwachung einzelner DataSets die parallel zu nicht-überwachten DataSets verwendet werden können, ist diese Anforderung erfüllt.

Universalität: Die Universalität zeichnet sich durch die Unabhängigkeit zu den Transport Protokollen aus. Die Überwachung funktioniert mit UDP und Ethernet Layer 2 und erfüllt damit die Anforderung.

Echtzeitfähigkeit: Der überwachte Datenaustausch darf die Leistungsfähigkeit der Kommunikation zwischen Publisher und Subscriber nicht nennenswert beeinflussen. Der Datenaustausch funktioniert mit nur wenigen zusätzlichen Bytes. Wie in Kapitel 5 gezeigt, ist die Echtzeitfähigkeit nur geringfügig durch die Enkodierung und Dekodierung beeinflusst, sofern eine Compiler-Optimierung eingesetzt wird. Die Anforderung gilt somit als erfüllt.

Konfigurierbarkeit: Der überwachte Datenaustausch muss auch über die Client/Server-Architektur konfigurierbar sein. Dadurch dass sich die Informationsmodelle des überwachten Datenaustauschs von den Standard-Informationsmodellen von OPC UA PubSub ableiten, ist diese Anforderung automatisch erfüllt.

Ressourcenbedarf: Der überwachte Datenaustausch darf keinen übermäßigen Ressourcenbedarf auf den Geräten erzeugen. Die Überwachung basiert nach wie vor auf einer entkoppelten Kommunikation und weist dadurch den fast gleichen Ressourcenbedarf wie Standard-OPC UA PubSub auf. Die Anforderung gilt demnach als erfüllt.

7.5 Zusammenfassung

In diesem Kapitel wurde in einem ersten Schritt über eine Konkretisierung einer 1-zu-N-Kardinalität gezeigt, wie prinzipiell der überwachte Datenaustausch zwischen mehreren Entitäten funktioniert. Außerdem wird verdeutlicht, wie überwachter Datenaustausch parallel zu nicht-überwachten Datenaustausch betrieben werden kann. In einem zweiten Schritt wurden die Zustandsmaschinen über die Simulationssoftware Simulink getestet. In einem letzten Schritt wurde ein C2C-Anwendungsfall mit einer Achskopplung implementiert. Über diese drei Schritte konnte gezeigt werden, dass die im Kapitel 2 definierten Anforderungen vom vorgestellten Lösungsvorschlag erfüllt werden.

8 Zusammenfassung und Ausblick

Industrie 4.0 ist die vierte industrielle Revolution. Sie verlangt eine durchgehende Vernetzung aller Geräte und die digitale Kommunikation über alle Ebenen der Automatisierungspyramide hinweg. Bisher besteht die Feldebene jedoch aus einer Vielzahl von Bussystemen, die nur bedingt miteinander kompatibel sind. Basierend auf OPC UA PubSub und TSN wird nun ein herstellerübergreifender und interoperabler Feldbus entstehen, der dieses Problem löst.

Bisher bietet OPC UA PubSub jedoch nur eine Überwachung des Datenaustauschs auf Seiten des Subscribers. Auf Grund der entkoppelten Kommunikation ist dem Publisher nicht bekannt, ob seine versendeten Nachrichten bei den Subscribern ankommen. Dadurch ist es nur bedingt möglich, mit einer angemessenen Fehlerreaktion auf Kommunikationsprobleme zu reagieren. Folglich sind applikative Lösungen nötig, um den Datenaustausch zu überwachen. Es besteht hier die Gefahr, dass inkompatible Lösungen zur Überwachung des Datenaustauschs entstehen.

Deshalb wurde in der vorliegenden Arbeit eine Überwachung des Datenaustauschs erarbeitet. Durch diese Lösung lässt sich eine zyklische Kommunikation, wie sie unter anderem bei Anwendungsfällen aus der Feldebene eingesetzt wird, auf Publisher- und Subscriber-Seite überwachen. Es wurde folgendermaßen vorgegangen.

Auf Grund fehlender Literatur über die Leistung von OPC UA PubSub in Kombination mit TSN wurde zunächst eine Leistungsmetrik von OPC UA PubSub und TSN erstellt. Um die Leistungsmetrik zu erstellen, wurden über eine entwickelte Evaluierungsplattform und eine experimentelle Methode Daten erhoben und evaluiert. Die daraus gewonnenen Erkenntnisse stellen die Rahmenbedingungen für die Entwicklung der Überwachung des Datenaustauschs dar.

Bei der Konzeption der Überwachung wurde in einem ersten Schritt die Überwachung der Feldbusprotokolle PROFINET und Sercos III verglichen und abstrahiert. In einem zweiten Schritt wurde, ausgehend von dieser Abstraktion, eine Überwachung des Datenaustauschs über OPC UA PubSub abgeleitet. Dazu wurden entsprechende Zustandsmaschinen, Informationsmodelle und Erweiterungen am UADP-Protokoll erarbeitet. Des Weiteren wurde eine Möglichkeit

zur Darstellung sämtlicher Zustände im OPC UA-Server über sogenannte Virtual Reader definiert.

Die entwickelten Lösungen wurden über das Simulationswerkzeug Simulink validiert und im Rahmen einer implementierten Referenzapplikation untersucht. Es konnte gezeigt werden, dass die Anforderungen einer typischen C2C-Anwendung – trotz Überwachung der Kommunikation – weiterhin erfüllt werden.

Die erarbeitete Überwachung kann auf weitere zyklische Anwendungsfälle übertragen werden. Darüber hinaus ist es vorstellbar, diese Überwachung in den OPC UA PubSub-Standard aufzunehmen, um eine standardisierte Lösung zur Überwachung des Datenaustauschs sicherzustellen.

Literatur

Arthur et al. 2007
Arthur, Siro; Emde, Carsten; McGuire, Nicholas, 2007.
Assessment of the Real-Time Preemption Patches (RT-Preempt) and their Impact on the General Purpose Performance of the System.
In: *Proceedings of the 9th OSADL Real-Time Linux Workshop.*
Verfügbar unter: `http://www.osadl.org/Assessment-of-the-Realtime-Preemption-Pa.osadl_rtlws2007_rtpreempt.0.html`.
Zugriff am: 06. 04. 2020

Bauernhansl et al. 2014
Bauernhansl, Thomas; Hompel, Michael ten; Vogel-Heuser, Birgit, 2014.
Industrie 4.0 in Produktion, Automatisierung und Logistik : Anwendung - Technologien - Migration.
Wiesbaden: Springer Vieweg.
ISBN 978-3-658-04681-1

Betz et al. 2009
Betz, Wolfgang; Cereia, Marco; Bertolotti, Ivan Cibrario, 2009.
Experimental Evaluation of the Linux RT Patch for Real-Time Applications.
In: *2009 IEEE Conference on Emerging Technologies and Factory Automation (ETFA).*
New York: Curran Associates Inc.
S. 1–4.
ISBN 978-1-4244-2727-7.
DOI: `10.1109/ETFA.2009.5347056`

Brosky et al. 2003
Brosky, Steve; Rotolo, Steve, 2003.
Shielded Processors: Guaranteeing Sub-Millisecond Response in Standard Linux.
In: *Proceedings International Parallel and Distributed Processing Symposium.*
New York: Curran Associates Inc.

S. 9.
ISBN 0-7695-1926-1.
DOI: 10.1109/IPDPS.2003.1213237

Brown et al. 2010 Brown, Jeremy H.; Martin, Brad, 2010.
How fast is fast enough? Choosing between Xenomai and Linux for Real-Time Applications.
In: *Proceedings of the 12th Real-Time Linux Workshop.*
Verfügbar unter: https://www.semanticscholar.org/paper/How-fast-is-fast-enough-Choosing-between-Xenomai-Brown/9eb51dbe38fb23034e80b8664d8281996d2a5ef6.
Zugriff am: 06. 04. 2020

Bruckner et al. 2019 Bruckner, Dietmar; Stanica, Marius-Petru; Blair, Richard; Schriegel, Sebastian; Kehrer, Stephan; Seewald, Maik; Sauter, Thilo, 2019.
An Introduction to OPC UA TSN for Industrial Communication Systems.
Proceedings of the IEEE, **107** (6), S. 1121–1131.
DOI: 10.1109/JPROC.2018.2888703

Buttazzo et al. 2006 Buttazzo, Giorgio; Lipari, Giuseppe; Abeni, Luca, 2006.
Soft Real-Time Systems : Predictability Vs. Efficiency.
New York: Springer-Verlag US.
ISBN 978-0-387-28147-6

Candido et al. 2010 Candido, Goncalo; Jammes, Francois; de Oliveira, Jose Barata; Colombo, Armando W., 2010.
SOA at device level in the industrial domain: Assessment of OPC UA and DPWS specifications.
In: *2010 8th IEEE International Conference on Industrial Informatics (INDIN).*
New York: Curran Associates Inc.
S. 598–603.
ISBN 978-1-4244-7298-7.
DOI: 10.1109/INDIN.2010.5549676

Cerqueira et al. 2013 Cerqueira, Felipe; Brandenburg, Björn B., 2013.
A Comparison of Scheduling Latency in Linux, PREEMPT-RT, and LITMUS RT.
In: *Proceedings of International Workshop on Operating Operating Systems Platforms for Embedded Real-Time Applications (OSPERT).*

S. 19–29.
Verfügbar unter: https://people.mpi-sws.org/~bbb/papers/pdf/ospert13.pdf.
Zugriff am: 06. 04. 2020

Chalas 2015 Chalas, Konstaninos, 2015.
Evaluation of Real-Time Operating Systems for FGC Controls.
Verfügbar unter: https://cds.cern.ch/record/2053700/files/Evaluation_of_Real_time_operating_systems_for_FGC_controls.pdf
Zugriff am: 18. 03. 2020

Dantam et al. 2015 Dantam, Neil T.; Lofaro, Daniel M.; Hereid, Ayonga; Oh, Paul Y.; Ames, Aaron D.; Stilman, Mike, 2015.
The Ach Library: A New Framework for Real-Time Communication.
IEEE Robotics & Automation Magazine, **22** (1), S. 76–85.
DOI: 10.1109/MRA.2014.2356937

Norm DIN 44300-1 DIN 44300-1:1988.
Teil 1: Begriffe, Allgemeine Begriffe

Norm DIN 44300-2 DIN 44300-2:1988.
Teil 2: Begriffe, Informationsdarstellung

Norm DIN 44300-3 DIN 44300-3:1988.
Teil 3: Begriffe, Datenstrukturen

Norm DIN 44300-4 DIN 44300-4:1988.
Teil 4: Begriffe, Programmierung

Norm DIN 44300-5 DIN 44300-5:1988.
Teil 5: Begriffe, Aufbau digitaler Rechensysteme

Norm DIN 44300-6 DIN 44300-6:1988.
Teil 6: Begriffe, Speicherung

Norm DIN 44300-7 DIN 44300-7:1988.
Teil 7: Begriffe, Zeiten

Norm DIN 44300-8 DIN 44300-8:1988.
Teil 8: Begriffe, Verarbeitungsfunktionen

Norm DIN 44300-9 DIN 44300-9:1988.
Teil 9: Begriffe, Verarbeitungsabläufe

Norm DIN EN 61158-1 DIN EN 61158-1:2014.

Industrielle Kommunikationsnetze – Feldbusse – Teil 1: Überblick und Leitfaden zu den Normen der Reihe IEC 61158 und IEC 61784

Norm DIN EN 61158-2 DIN EN 61158-2:2014.
Industrielle Kommunikationsnetze – Feldbusse – Teil 2: Spezifikation und Dienstfestlegungen des Physical Layer (Bitübertragungsschicht)

Norm DIN EN 61158-3-16 DIN EN 61158-3-16:2008.
Industrielle Kommunikationsnetze – Feldbusse – Teil 3-16: Dienstfestlegungen des Data Link Layer (Sicherungsschicht) – Typ 16-Elemente

Norm DIN EN 61158-3-19 DIN EN 61158-3-19:2014.
Industrielle Kommunikationsnetze - Feldbusse - Teil 3-19: Dienstfestlegungen des Data Link Layer (Sicherungsschicht) - Typ 19-Elemente

Norm DIN EN 61158-4-10 DIN EN 61158-4-10:2014.
Industrielle Kommunikationsnetze - Feldbusse - Teil 4-10: Protokollspezifikation des Data Link Layer (Sicherungsschicht) - Typ 10-Elemente

Norm DIN EN 61158-4-19 DIN EN 61158-4-19:2014.
Industrielle Kommunikationsnetze – Feldbusse – Teil 4-19: Protokollspezifikation des Data Link Layer (Sicherungsschicht) – Typ 19-Elemente

Norm DIN EN 61158-5-10 DIN EN 61158-5-10:2014.
Industrielle Kommunikationsnetze - Feldbusse - Teil 5-10: Dienstfestlegungen des Application Layer (Anwendungsschicht) - Typ 10-Elemente

Norm DIN EN 61158-5-19 DIN EN 61158-5-19:2014.
Industrielle Kommunikationsnetze - Feldbusse - Teil 5-19: Dienstfestlegungen des Application Layer (Anwendungsschicht) - Typ 19-Elemente

Norm DIN EN 61158-6-10 DIN EN 61158-6-10:2014.
Industrielle Kommunikationsnetze - Feldbusse - Teil 6-10: Protokollspezifikation des Application Layer (Anwendungsschicht) - Typ 10-Elemente

Norm DIN EN 61158-6-19 DIN EN 61158-6-19:2014.

Industrielle Kommunikationsnetze - Feldbusse - Teil 6-19: Protokollspezifikation des Application Layer (Anwendungsschicht) - Typ 19-Elemente

Norm DIN EN 62264-1 DIN EN 62264-1:2013.
Integration von Unternehmensführungs- und Leitsystemen - Teil 1: Modelle und Terminologie

Dozio et al. 2003 Dozio, Lorenzo.; Mantegazza, Paolo., 2003.
Real Time distributed Control Systems using RTAI.
In: *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003.*
New York: Curran Associates Inc.
S. 11–18.
ISBN 0-7695-1928-8.
DOI: 10.1109/ISORC.2003.1199229

Drahos et al. 2018 Drahos, Peter; Kucera, Erik; Haffner, Oto; Klimo, Ivan, 2018.
Trends in industrial communication and OPC UA.
In: *2018 Cybernetics & Informatics (K&I).*
New York: Curran Associates Inc.
S. 1–5.
ISBN 978-1-5386-4419-5.
DOI: 10.1109/CYBERI.2018.8337560

Eckhardt et al. 2019a Eckhardt, Andreas; Müller, Sebastian, 2019.
Analysis of the Round Trip Time of OPC UA and TSN based Peer-to-Peer Communication.
In: *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA).*
New York: Curran Associates Inc.
S. 161–167.
ISBN 978-1-7281-0303-7.
DOI: 10.1109/ETFA.2019.8869060

Eckhardt et al. 2019b Eckhardt, Andreas; Müller, Sebastian, 2019.
OPC UA over TSN in der Automatisierung,
2019 (9), S. 643–649

Eckhardt et al. 2018 Eckhardt, Andreas; Müller, Sebastian; Leurs, Ludwig, 2018.
An Evaluation of the Applicability of OPC UA Publish Subscribe on Factory Automation Use Cases.

In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*.
New York: Curran Associates Inc.
S. 1071–1074.
ISBN 978-1-5386-7108-5.
DOI: 10.1109/ETFA.2018.8502445

Emde 2010 Emde, Carsten, 2010.
Long-Term Monitoring of Apparent Latency in PREEMPT RT Linux Real-Time Systems.
In: *Proceedings of the 12th Real-Time Linux Workshop*.
Verfügbar unter: https://www.semanticscholar.org/paper/Long-term-monitoring-of-apparent-latency-in-PREEMPT-Emde/2a4286ba7065cb058af87b0ce18485b9c82f0105.
Zugriff am: 06. 04. 2020

Fayyad-Kazan et al. 2014 Fayyad-Kazan, Hasan; Perneel, Luc; Timmerman, Martin, 2014.
Linux PREEMPT-RT v2.6.33 versus v3.6.6.
ACM SIGBED Review, **11** (1), S. 26–31.
DOI: 10.1145/2597457.2597460

Feuerer 2007 Feuerer, Peter, 2007.
Benchmark and Comparison of Real-Time Solutions based on Embedded Linux.
Hochschule Ulm, Diplomarbeit

Garimella 2018 Garimella, Phani Kumar, 2018.
IT-OT Integration Challenges in Utilities.
In: *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*.
New York: Curran Associates Inc.
S. 199–204.
ISBN 978-1-5386-6227-4.
DOI: 10.1109/CCCS.2018.8586807

Garre et al. 2014 Garre, Carlos; Mundo, Domenico; Gubitosa, Marco; Toso, Alessandro, 2014.
Real-Time and Real-Fast Performance of General-Purpose and Real-Time Operating Systems in Multithreaded Physical Simulation of Complex Mechanical Systems.
Mathematical Problems in Engineering, **2014** (945850), S. 1–14.
DOI: 10.1155/2014/945850

Gogolev et al. 2018 Gogolev, Alexander; Mendoza, Francisco; Braun, Rol, 2018. TSN-Enabled OPC UA in Field Devices. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA).* New York: Curran Associates Inc. S. 297–303. ISBN 978-1-5386-7108-5. DOI: 10.1109/ETFA.2018.8502597

HMS 2019 HMS, 2019. *Feldbus Markverteilung.* Verfügbar unter: https://de.statista.com/statistik/daten/studie/457627/umfrage/marktanteile-industrieller-netzwerke-weltweit/ Zugriff am: 18. 03. 2020

IANA 2019 2019 IANA, 2019. *IPv4 Multicast Address Space Registry.* Verfügbar unter: https://www.iana.org/assignments/multicast-addresses/multicast-addresses.xhtml Zugriff am: 18. 03. 2020

Norm IEC PAS 63088 IEC PAS 63088:2017. *Smart Manufacturing - Reference Architecture Model Industrie 4.0*

Norm IEEE 1934 IEEE 1934-2018. *IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing*

Norm IEEE 802.11 IEEE 802.11:2016. *IEEE Standard for Information Technology: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*

Norm IEEE 802.1AB IEEE 802.1AB:2016. *IEEE Standard for Local and Metropolitan Area Networks: Station and Media Access Control Connectivity Discovery*

Norm IEEE 802.1AS IEEE 802.1AS:2018. *IEEE Standard for Local and Metropolitan Area Networks: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*

Norm IEEE 802.1AX IEEE 802.1AX:2014.

IEEE Standard for Local and Metropolitan Area Networks: Link Aggregation

Norm IEEE 802.1BA IEEE 802.1BA:2011.
IEEE Standard for Local and Metropolitan Area Networks: Audio Video Bridging (AVB) Systems

Norm IEEE 802.1CB IEEE 802.1CB:2017.
IEEE Standard for Local and Metropolitan Area Networks: Frame Replication and Elimination for Reliability

Norm IEEE 802.1CM IEEE 802.1CM:2018.
IEEE Standard for Local and Metropolitan Area Networks: Time-Sensitive Networking for Fronthaul

Norm IEEE 802.1D IEEE 802.1D:2004.
IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges

Norm IEEE 802.1Q IEEE 802.1Q:2018.
IEEE Standard for Local and Metropolitan Area Networks: Bridges and Bridged Networks

Norm IEEE 802.1Qbu IEEE 802.1Qbu:2016.
IEEE Standard for Local and Metropolitan Area Networks: Bridges and Bridged Networks - Amendment 26: Frame Preemption

Norm IEEE 802.1Qbv IEEE 802.1Qbv:2015.
IEEE Standard for Local and Metropolitan Area Networks: Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic

Norm IEEE 802.3 IEEE 802.3:2018.
IEEE Standard for Ethernet

IEEE AVB Task Group 2020 IEEE AVB Task Group.
Audio Video Bridging Task Group.
Verfügbar unter: `http://www.ieee802.org/1/pages/avbridges.html`
Zugriff am: 18. 03. 2020

IEEE Ethertype 2020 IEEE Ethertype.
Ethertype Numbers.
Verfügbar unter: `http://standards-oui.ieee.org/ethertype/eth.txt`

Zugriff am: 18. 03. 2020

IEEE TSN Task Group 2020 IEEE TSN Task Group.
Time-Sensitive Networking Task Group.
Verfügbar unter: `http://www.ieee802.org/1/pages/tsn.html`
Zugriff am: 18. 03. 2020

IIC Traffic Types 2018 IIC Traffic Types, 2018.
Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types : An Industrial Internet Consortium Results White Paper.
Verfügbar unter: `https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Traffic_Whitepaper_20180418.pdf`
Zugriff am: 18. 03. 2020

Norm ISO 11898-1 ISO 11898-1:2015.
Road Vehicles - Controller Area Network (CAN) - Part 1: Data Link Layer and Physical Signalling

Jasperneite et al. 2007 Jasperneite, Juergen; Schumacher, Markus; Weber, Karl, 2007.
Limits of increasing the performance of Industrial Ethernet protocols.
In: *2007 IEEE Conference on Emerging Technologies and Factory Automation (ETFA).*
New York: Curran Associates Inc.
S. 17–24.
DOI: `10.1109/EFTA.2007.4416748`

Jiang et al. 2018 Jiang, Junhui; Li, Yuting; Hong, Seung Ho; Xu, Aidong; Wang, Kai, 2018.
A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++.
In: *2018 IEEE International Conference on Mechatronics and Automation (ICMA).*
New York: Curran Associates Inc.
S. 643–648.
ISBN 978-1-5386-6074-4.
DOI: `10.1109/icma.2018.8484302`

Koolwal 2009 Koolwal, Kushal, 2009.
Investigating Latency Effects of the Linux Real-Time Preemption Patches (PREEMPT RT) on AMD's GEODE LX Platform.

In: *Proceedings of the 11th Real-Time Linux Workshop*.
Verfügbar unter: https://www.semanticscholar.org/paper/Investigating-latency-effects-of-the-Linux-Patches-Koolwal/25e7d6093ab2cd4199b3278732f162b81f8a50c9.
Zugriff am: 06. 04. 2020

Kopetz 2011 Kopetz, Hermann, 2011.
Real-Time Systems : Design Principles for Distributed Embedded Applications.
2. Aufl.
Boston: Springer US.
ISBN 978-1-4419-8237-7.
DOI: 10.1007/978-1-4419-8237-7

Lechler 2011 Lechler, Armin, 2011.
Konzeption einer funktional einheitlichen Applikationsschnittstelle für Ethernet-basierte Bussysteme.
Heimsheim: Jost-Jetter.
ISW/IPA Forschung und Praxis 184.
Stuttgart, Univ., Diss., 2011.
ISBN 978-3-939890-78-2

Linux Manual 2020 Linux Manual.
SCHED.
Verfügbar unter: http://man7.org/linux/man-pages/man7/sched.7.html
Zugriff am: 18. 03. 2020

Litayem et al. 2011 Litayem, Nabil; Ben Saoud, Slim, 2011.
Impact of the Linux Real-time Enhancements on the System Performances for Multi-core Intel Architectures.
International Journal of Computer Applications, **17** (3), S. 17–23.
DOI: 10.5120/2202-2796

Lo Bello et al. 2019 Lo Bello, Lucia; Steiner, Wilfried, 2019.
A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems.
Proceedings of the IEEE, **107** (6), S. 1094–1120.
DOI: 10.1109/JPROC.2019.2905334

Marieska et al. 2011 Marieska, Mastura D.; Hariyanto, Paul G.; Fauzan, M. Firda; Kistijantoro, Achmad Imam; Manaf, Afwarman, 2011.

On Performance of Kernel based and Embedded Real-Time Operating System: Benchmarking and Analysis.
In: *International Conference on Advanced Computer Science and Information Systems (ICACSIS).*
New York: Curran Associates Inc.
S. 401–406.
ISBN 978-979-1421-11-9

Meudt et al. 2017 Meudt, Tobias; Pohl, Malte; Metternich, Joachim, 2017.
Die Automatisierungspyramide - Ein Literaturüberblick.
Verfügbar unter: `https://tuprints.ulb.tu-darmstadt.de/id/eprint/6298`
Zugriff am: 18. 03. 2020

Norm DCOM Microsoft Corporation Standard 2018.
Distributed Component Object Model (DCOM) Remote Protocol

Mossige et al. 2007 Mossige, Morten; Sampath, Pradyumna; Rao, Rachana G., 2007.
Evaluation of Linux rt-preempt for Embedded Industrial Devices for Automation and Power Technologies - A Case Study.
Verfügbar unter: `https://www.osadl.org/fileadmin/events/rtlws-2007/Sampath.pdf`
Zugriff am: 18. 03. 2020

Murikipudi et al. 2015 Murikipudi, Akhilesh; Prakash, V.; Vigneswaran, T., 2015.
Performance Analysis of Real Time Operating System with General Purpose Operating System for Mobile Robotic System.
Indian Journal of Science and Technology, **8** (19).
DOI: `10.17485/ijst/2015/v8i19/77017`

Nasrallah et al. 2019 Nasrallah, Ahmed; Thyagaturu, Akhilesh S.; Alharbi, Ziyad; Wang, Cuixiang; Shao, Xing; Reisslein, Martin; Elbakoury, Hesham, 2019.
Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS).
IEEE Access, **7**, S. 44165–44181.
DOI: `10.1109/ACCESS.2019.2908613`

Norm OASIS AMQP OASIS Standard AMQP 2012.
Advanced Message Queuing Protocol V1.0

Norm OASIS MQTT OASIS Standard MQTT 2019.
Message Queuing Telemetry Transport V5.0

Norm OPC UA Part 14 OPC Unified Architecture.
Part 14: PubSub V1.04

Norm OPC UA Part 1 OPC Unified Architecture.
Part 1: Overview and Concepts V1.04

Norm OPC UA Part 8 OPC Unified Architecture.
Part 8: Data Access V1.04

Norm OPC UA Part 2 OPC Unified Architecture.
Part 2: Security Model V1.04

Norm OPC UA Part 9 OPC Unified Architecture.
Part 9: Alarms and Conditions V1.04

Norm OPC UA Part 3 OPC Unified Architecture.
Part 3: Address Space Model V1.04

Norm OPC UA Part 10 OPC Unified Architecture.
Part 10: Programs V1.04

Norm OPC UA Part 4 OPC Unified Architecture.
Part 4: Services V1.04

Norm OPC UA Part 11 OPC Unified Architecture.
Part 11: Historical Access V1.04

Norm OPC UA Part 5 OPC Unified Architecture.
Part 5: Information Model V1.04

Norm OPC UA Part 12 OPC Unified Architecture.
Part 12: Discovery and Global Services V1.04

Norm OPC UA Part 6 OPC Unified Architecture.
Part 6: Mappings V1.04

Norm OPC UA Part 13 OPC Unified Architecture.
Part 13: Aggregates V1.04

Norm OPC UA Part 7 OPC Unified Architecture.
Part 7: Profiles V1.04

Norm OPC UA A.6 OPC Unified Architecture - Amendment 6.
UADP Header Layouts

Plattform Industrie 4.0 2018 Plattform Industrie 4.0, 2018.
Was ist Industrie 4.0?
Verfügbar unter: `https://www.plattform-i40.de/PI40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html`

Zugriff am: 18. 03. 2020

Pritschow 2006 Pritschow, Günter, 2006.
Einführung in die Steuerungstechnik : Mit 40 Tabellen.
München: Hanser.
ISBN 3-446-21422-4

Prytz et al. 2005 Prytz, Gunner; Johannessen, Svein, 2005.
Real-time Performance Measurements using UDP on Windows and Linux.
In: *2005 IEEE Conference on Emerging Technologies and Factory Automation (ETFA).*
New York: Curran Associates Inc.
S. 925–932.
ISBN 0-7803-9401-1.
DOI: 10.1109/ETFA.2005.1612771

Rauen et al. 2017 Rauen, Hartmut; Mosch, Christian, 2017.
Industrie 4.0 Kommunikation mit OPC UA : Leitfaden zur Einführung in den Mittelstand.
Verfügbar unter: http://industrie40.vdma.org/documents/4214230/16617345/1492669959563_2017_Leitfaden_OPC_UA_LR.pdf/f4ddb36f-72b5-43fc-953a-ca24d2f50840
Zugriff am: 18. 03. 2020

Reghenzani et al. 2019 Reghenzani, Federico; Massari, Giuseppe; Fornaciari, William, 2019.
The Real-Time Linux Kernel.
ACM Computing Surveys, **52** (1), S. 1–36.
DOI: 10.1145/3297714

Norm RFC 6455 RFC 6455:2011.
The WebSocket Protocol

Norm RFC 768 RFC 768:1980.
User Datagram Protocol

Norm RFC 793 RFC 793:1981.
Transmission Control Protocol

Norm RFC 8259 RFC 8259:2017.
The JavaScript Object Notation (JSON) Data Interchange Format

Norm RFC 8304 RFC 8304:2018.

Transport Features of the User Datagram Protocol (UDP) and Lightweight UDP (UDP-Lite)

Norm RFC 9446-1.3 RFC 9446-1.3:2018.
The Transport Layer Security (TLS) Protocol

Schwarz et al. 2013 Schwarz, Michael H.; Borcsok, Josef, 2013.
A Survey on OPC and OPC-UA: About the Standard, Developments and Investigations.
In: *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT).*
New York: Curran Associates Inc.
S. 1–6.
ISBN 978-1-4799-0431-0.
DOI: 10.1109/ICAT.2013.6684065

Siepmann et al. 2016 Siepmann, David; Graef, Norbert, 2016.
Industrie 4.0 – Grundlagen und Gesamtzusammenhang.
In: Roth, Armin (Hrsg.): *Einführung und Umsetzung von Industrie 4.0.*
Berlin: Springer-Verlag.
S. 17–82.
ISBN 978-3-662-48504-0.
DOI: 10.1007/978-3-662-48505-7_2

Vaduva et al. 2016 Vaduva, Alexandru; Gonzalez, Alex; Simmonds, Chris, 2016.
Embedded Linux for Developers.
Birmingham: Packt Publishing.
ISBN 1-787-12420-7

Wollschlaeger et al. 2018 Wollschlaeger, Martin; Debes, Thomas; Kalhoff, Johannes; Wickinger, Jens; Dietz, Holger; Feldmeier, Günther; Michels, Jan; Scholing, Heinz; Billmann, Meik, 2018.
Kommunikation im Industrie-4.0-Umfeld.
Verfügbar unter: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2018/April/Kommunikation_im_Industrie-4.0-Umfeld/Kommunikation_im_Industrie-4.0-Umfeld_Download-Neu.pdf
Zugriff am: 18. 03. 2020

Wollschlaeger et al. 2017 Wollschlaeger, Martin; Sauter, Thilo; Jasperneite, Juergen, 2017.

The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0.
IEEE Ind. Electron. Mag. (IEEE Industrial Electronics Magazine), **11** (1), S. 17–27.
DOI: 10.1109/MIE.2017.2649104

Yanyan et al. 2018 Yanyan, Zhang; Xiangjin, Ran, 2018.
Analysis of Linux Kernel's Real-Time Performance.
In: *2018 International Conference on Smart Grid and Electrical Automation (ICSGEA)*.
New York: Curran Associates Inc.
S. 191–194.
ISBN 978-1-5386-6953-2.
DOI: 10.1109/ICSGEA.2018.00055

Zhao et al. 2018 Zhao, Luxi; Pop, Paul; Craciunas, Silviu S., 2018.
Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus.
IEEE Access, **6**, S. 41803–41815.
DOI: 10.1109/ACCESS.2018.2858767

Universität Stuttgart

Beim OPC UA Publish Subscribe Standard liegen dem Publisher auf Grund der entkoppelten Kommunikation keine Informationen über den Zustand der Subscriber vor. Dadurch werden Kommunikationsstörungen auf Seiten des Publishers nicht apparent, was eine angemessene Fehlerreaktion in der Applikation unmöglich macht.

Im Rahmen dieser Arbeit wird ein Lösungsweg aufgezeigt, der mit Hilfe einer Überwachung der Kommunikation diesen Mangel behebt. Der vorgeschlagene Lösungsweg basiert dabei auf einer Erweiterung des OPC UA Publish Subscribe Standards. Dabei werden über einen bidirektionalen Kommunikationskanal relevante Zustandsinformationen des Subscribers an den Publisher übermittelt. Die vorgestellte Lösung erfüllt dabei die Anforderungen an eine echtzeitbasierte Querkommunikation zwischen industriellen Steuerungen. Methodisch wurde dabei eine Evaluierungsplattform eingesetzt, um zunächst eine Leistungsmetrik von OPC UA Publish Subscribe mit Time-Sensitive Networking zu erstellen. Parallel dazu wurden PROFINET IRT und Sercos 3 miteinander verglichen und abstrahiert. Basierend darauf wurde die Überwachung abgeleitet.

Fraunhofer VERLAG

ISBN 978-3-8396-1816-5

9 783839 618165