VDI

REIHE 08
MESS-,
STEUERUNGS-
UND REGELUNGS-
TECHNIK

Fortschritt-Berichte VDI

Mahyar Azarmipour, M. Sc.,
Aachen

NR. 1275

Virtualisierung prozessnaher Steuerungen in der Prozessautomatisierung

BAND
1|1

VOLUME
1|1

AACHENER PROZESSLEITTECHNIK

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Virtualisierung prozessnaher Steuerungen in der Prozessautomatisierung

Von der Fakultät für Georessourcen und Materialtechnik der Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

vorgelegt von

Mahyar Azarmipour, M. Sc.

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ.-Prof. Dr.-Ing. Stefan Kowalewski
Univ.-Prof. Dr.-Ing. Tobias Kleinert

Tag der mündlichen Prüfung: 25.01.2022

VDI

REIHE 08
MESS-,
STEUERUNGS-
UND REGELUNGS-
TECHNIK

Fortschritt-Berichte VDI

Mahyar Azarmipour, M. Sc.,
Aachen

NR. 1275

Virtualisierung prozessnaher Steuerungen in der Prozessautomatisierung

BAND
1|1

VOLUME
1|1

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

VDI verlag

Azarmipour, Mahyar

Virtualisierung prozessnaher Steuerungen in der Prozessautomatisierung

Fortschritt-Berichte VDI, Reihe 08, Nr. 1275. Düsseldorf: VDI Verlag 2022.

126 Seiten, 62 Bilder, 4 Tabellen.

ISBN 978-3-18-527508-1, E-ISBN 978-3-18-627508-0, ISSN 0178-9546

48,00 EUR/VDI-Mitgliederpreis: 43,20 EUR

Für die Dokumentation: Virtualisierung – Speicherprogrammierbare Steuerung – Informationsdiode – Industrie 4.0 – Dynamisches Deployment – Automatisierungstechnik – IT/OT-Konvergenz

Keywords: Virtualization – Programmable logic controller – Information diode – Industry 4.0 – Dynamic deployment – Automation – IT/OT-Convergence

Die vorliegende Arbeit wendet sich an Ingenieur*innen und Wissenschaftler*innen aus der Prozessautomatisierung. Ziel dieser Arbeit ist ein Architekturentwurf für die Steuerungsgeräte der prozessnahen Komponenten, um diese mit einer höheren Vernetzung und Wandelbarkeit auszustatten. Die Architektur setzt Hypervisor-Virtualisierung ein, um eine Trennung der Anwendungen mit unterschiedlichen Anforderungen auf der gleichen Hardware zu ermöglichen. Die Anwendungen werden in vorkonfigurierten Partitionen gekapselt und betrieben. Um die Modularisierung der Anwendungen zu erhöhen, werden Container als zusätzliche Virtualisierungskomponenten eingesetzt. Für die Verwaltung der gesamten Komponentenhierarchie ist ein Verwaltungssystem vorgesehen, das die erforderlichen Dienste zur Komponentenverwaltung zur Verfügung stellt.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter www.dnb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek (German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at www.dnb.de.

D82 (Diss. RWTH Aachen University, 2022)

ISBN 978-3-18-527508-1, E-ISBN 978-3-18-627508-0, ISSN 0178-9546

Vorwort

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Prozessleittechnik der RWTH Aachen University entstanden. An dieser Stelle möchte ich mich herzlich für die Unterstützung und die ermöglichten Chancen bedanken.

An erster Stelle gilt mein großer Dank Herrn Prof. Dr.-Ing. Ulrich Epple für die Unterstützung meines Promotionsvorhabens. Die angenehme und konstruktive Arbeitsatmosphäre sowie der ausgezeichnete fachliche Austausch mit ihm haben zum erfolgreichen Abschluss meiner Arbeit beigetragen.

Bei Herrn Prof. Dr.-Ing. Stefan Kowalewski, Inhaber des Lehrstuhls Informatik 11 – Embedded Software an der RWTH Aachen University, möchte ich mich für die Übernahme der Rolle des Zweitgutachters bedanken.

Ich bedanke mich bei Herrn Prof. Dr.-Ing. Tobias Kleinert für seine Unterstützung und die Fachdiskussionen, welche ebenfalls zum Gelingen der Arbeit beigetragen haben.

Ich danke meinen Kollegen und den studentischen Hilfskräften für die gute Zusammenarbeit und die interessanten Gespräche. Besonders hervorheben möchte ich (in alphabetischer Reihenfolge) Haitham Elfaham, Julian Grothoff, Daniel Jakob, Lars Nothdurft und Christian von Trotha. Bei Frau Milescu bedanke ich mich für die gute Zusammenarbeit und organisatorische Hilfe.

Ein weiterer Dank gilt Afrooz Nazari für die Unterstützung und Motivation in den vergangenen Jahren.

Besonderer Dank gilt auch meinen Eltern und meiner Schwester für die Unterstützung während meiner gesamten Promotionszeit.

Warstein, im März 2022 Mahyar Azarmipour

Wissenschaft: Es ist nicht ihr Ziel, der unendlichen Weisheit eine Tür zu öffnen, sondern eine Grenze zu setzen dem unendlichen Irrtum.

Bertolt Brecht

Inhaltsverzeichnis

List of Symbols

ASIL Automotive Safety Integrity Level

AUTOSAR AUTomotive Open System ARchitecture

CFC Continuous Function Charts

CPC Core Process Control

CPU Central Processing Unit

CPS Cyber Physical Systems

SFC Sequential Function Chart

DT Digital Twin

ESE Einzelsteuereinheiten

ECU Electronic Control Units

ERP Enterprise Resource Planning

EAL Evaluation Assurance Levels

FS File System

FB Funktionsbaustein

FIFO First In First Out

GSE Gruppensteuereinheiten

HMI Human Machine Interface

I40 Industrie 4.0

IIC Industrial Internet Consortium

IoT Internet of Things

IP Internet Protocol

IT Informationstechnik

IACS Industrial Automation and Control System

IPC Industrie-PC

KAS komponentenbasierte Architektur für Automatisierungssysteme

LWIP Light Weight Internet Protocol

M+O Monitoring and Optimization

MMU Memory Management Unit

MES Manufacturing Execution System

NOA NAMUR Open Architecture

OT Operational Technology

OPC UA Open Platform Communications Unified Architecture

OO Object Oriented

OS Betriebssystem

POSIX Portable Operating System Interface

PNK Prozessnahe Komponente

QoS Quality of Service

RTOS Real Time Operating System

ROM Read Only Memory

SSC Sequential State Charts

SFC Sequential Function Charts

SAP Service Access Points

SIL Safety Integrity Level

SPS Speicherprogrammierbare Steuerung

SOA Service Oriented Architecture

SCADA Supervisory control and data acquisition

TCP Transmission Control Protocol

UDP User Datagram Protocol

VM Virtuelle Maschine

VMM Virtual Machine Monitor

VoR Verification of Request

Kurzfassung

Industrie 4.0 ist ein neues Paradigma, das eine zentrale Rolle in der Entwicklung der zukünftigen Automatisierungssysteme spielt. Die neue Generation der industriellen Automatisierungstechnik zielt auf die Erhöhung der Wandelbarkeit der Automatisierungssysteme ab. Dabei ist die Vernetzung und die Kooperation mit der IT-Welt eine wichtige Voraussetzung, um die angeforderte Wandelbarkeit zu erreichen. Daher müssen neue Architekturen und Lösungen eingesetzt werden, um eine Kooperation zwischen den Automatisierungssystemen und der IT zu realisieren. Ziel dieser Arbeit ist ein Architekturentwurf für die Steuerungsgeräte der prozessnahen Komponenten, um diese mit einer höheren Vernetzung und Wandelbarkeit auszustatten. Die Hauptanforderungen, welche von der Architektur erfüllt werden, sind:

- Der parallele Betrieb von Anwendungen unterschiedlicher Kritikalität
- Das dynamische Deployment von neuen Anwendungen zur Laufzeit
- Die Realisierung eines sicheren Gateways für die Kommunikation zwischen Systemen der Automatisierungsebene und übergeordneten IT-Systemen
- Die offene Kommunikation mit der IT-Welt
- Die Realisierung eines lokalen Software- und Zugriffsverwaltungssystems

Die vorgeschlagene Architektur besteht aus einem Mehrebenen-Komponentenmodell und wird als komponentenbasierte Architektur für Automatisierungssysteme (KAS) bezeichnet. Die unterste Ebene der KAS-Architektur ist die Ebene der Partitionen. Die KAS-Architektur setzt Hypervisor-Virtualisierung ein, um eine Trennung der Anwendungen mit unterschiedlichen Anforderungen auf der gleichen Hardware zu ermöglichen. Die Anwendungen werden in vorkonfigurierten Partitionen gekapselt und betrieben. Um die Modularisierung der Anwendungen zu erhöhen werden Container als zusätzliche Virtualisierungskomponenten eingesetzt. Containertechnologien ermöglichen die Kapselung und Verwaltung der Anwendungen in unterschiedlichen Containern innerhalb einer Partition. Dadurch können beispielsweise unterschiedliche Versionen der Anwendungen in einer Partition verwaltet werden. Die Container bilden die zweite Komponentenebene in der KAS-Architektur. Die letzte Komponentenebene stellt die Kapselung in die Funktionsbausteine dar. Für die Verwaltung der gesamten Komponentenhierarchie ist in der KAS-Architektur ein Verwaltungssystem vorgesehen, das die erforderlichen Dienste zur Komponentenverwaltung zur Verfügung stellt. Das Verwaltungssystem ist eine Systemfunktionalität der KAS-Architektur und in einer eigenen Partition gekapselt. Eine weitere Systemfunktion der KAS-Architektur ist das Interface. Dieses wird ebenfalls in einer eigenen Partition gekapselt. Diese Interface-Partition ist die einzige Partition, die mit externen Komponenten außerhalb der Kernautomatisierung kommunizieren darf. In der Arbeit werden für die Validierung der KAS-Architektur beispielhaft Anwendungspartitionen für die Prozessführung

und die Simulation entwickelt. Mit Hilfe dieser Anwendungen können realistische Szenarien der Automatisierungsebene prototypisch implementiert und getestet werden. Die Ergebnisse zeigen, dass die KAS-Architektur eine leistungsfähige und übersichtlich verwaltbare Systemumgebung darstellt, um für neue Anforderungen eine hohe Flexibilität zu bieten, sowie der durchgängigen Interoperabilität der Automatisierungsebene zu genügen, ohne die Integrität der Kernautomation zu gefährden.

Abstract

Industry 4.0 is a new paradigm that plays a central role in the development of future automation systems. The new generation of industrial automation aims to increase the agility of the automation system. In this context, cooperation with the IT world is an important prerequisite to achieve the requested agility. Therefore, new architectures and solutions have to be developed to realize a cooperation between the automation systems and the IT. The goal of this work is an architecture design for the control devices in order to provide them with a higher level of connectivity and agility. The main features, which are fulfilled by the architecture, are:

- The parallel operation of applications of different criticality
- The dynamic deployment of new applications at runtime
- The realization of a secure gateway for communication between automation level systems and higher level IT systems
- The open communication with the IT world
- The realization of a local software and access management system

The proposed architecture consists of a multi-level component model and is referred to as a component-based architecture for automation systems (KAS). The lowest level of the KAS architecture is the partition level. The KAS architecture employs hypervisor virtualization to enable separation of applications with different requirements on the same hardware. Applications are encapsulated in preconfigured partitions. Containers are used as additional virtualization components to increase modularization of applications. Container technologies enable the encapsulation and management of applications in different containers within a partition. This means, for example, that different versions of the applications can be managed in one partition. The containers are the second component level in the KAS architecture. The last component level represents the encapsulation in the function blocks. For the management of the entire component hierarchy in the KAS architecture a management system is developed. The management system is a system functionality of the KAS architecture and is encapsulated in its own partition. Another system function of the KAS architecture is the interface. This is also encapsulated in its own partition. The interface partition is the only partition that is allowed to communicate with outside of the core automation domain. In this work, application partitions for process control and simulation are developed as examples for the validation of the KAS architecture. These applications can be used for a prototype implementation of automation level scenarios. The results show that the KAS architecture provides a powerful and clearly manageable system environment to meet the new requirements for agility as well as continuous interoperability of the automation level without compromising the integrity of the core automation.

1 Einleitung

Der zunehmende Bedarf nach agilen und dynamischen Produktionssystemen, im Kontext von neuen Paradigmen wie Industrie 4.0 (I40) oder Industrial Internet Consortium (IIC), ruft die Anforderung nach neuen Technologien und Informationsmodellen [56], [55] hervor. Internet of Things (IoT) [116], Cyber Physical Systems (CPS) [48], Cloud Computing [84] und Verwaltungsschale [28] sind einige Beispiele für diese neuen Technologien und Informationsmodelle. Sie werden entwickelt, um die Vernetzung, die Wandelbarkeit und den Optimierungsgrad von Produktionssystemen durch das Vorantreiben der Digitalisierung zu erhöhen. Die Digitalisierung der Produktion definiert neue Aufgaben und Funktionen für die Komponenten der Automatisierungspyramide. Diese Komponenten müssen wiederum hinsichtlich Hardware- und Software-Aspekten modifiziert werden, um den genannten Anforderungen gerecht zu werden. Prozessnahe Komponenten sind eine dieser Komponenten, die neue Funktionalitäten bereitstellen und neue Aufgaben und Anforderungen erfüllen müssen, um die Wandelbarkeit des Produktionssystems zu erhöhen. Die in dieser Arbeit vorgeschlagene Architektur zielt darauf ab, die Steuerungsgeräte der prozessnahen Komponenten mit neuen Funktionen auszustatten, um sie an die Anforderungen durch die zunehmende Prozessdigitalisierung anzupassen. Diese Funktionen müssen eine verbesserte Zusammenarbeit zwischen der industriellen Automatisierung und der Informationstechnik (IT) (z. B. Optimierungsfunktionen und Machine-Learning-Applikationen) ermöglichen. Die klassischen Aufgaben eines Automatisierungssystems (Prozessführung) müssen jedoch neben den neuen Funktionen weiterhin erfüllt werden. Darüber hinaus darf das Zusammenspiel von Industrieautomation und IT die Integrität des industriellen Automatisierungssystems in keiner Weise beeinträchtigen.

1.1 Motivation

I40 strebt eine Förderung der Digitalisierung der Automatisierungssysteme an [102]. Dies erfordert neue Technologien, die entweder in der Automatisierungstechnik entwickelt oder aus der IT-Domäne in die Automatisierungsdomäne integriert werden müssen. Die Konvergenz von IT und Operational Technology (OT) verfolgt in diesem Zusammenhang ein ähnliches Ziel [73]. Einige Beispiele für IT-Technologien, die zur Erhöhung des Optimierungsgrades und der Wandelbarkeit der industriellen Automatisierung eingesetzt werden, sind IoT, Cloud Computing, CPS. [103]. Diese Technologien spielen eine entscheidende Rolle für die Vernetzung, den Informationsaustausch, die Datenspeicherung usw. Eine der Hauptanforderungen im Zusammenhang mit I40 ist die Agilität (Wandelbarkeit). Agilität beschreibt die Fähigkeit eines Systems, sich auf ungeplante Veränderungen anzupassen [88]. Diese Veränderungen können beispielsweise hinsichtlich der Produktionsmenge, der Topologie und der Produkteigenschaften auftreten [105].

Die Erfüllung einer agilen Produktion erfordert Anpassungen und Neukonfigurationen in verschiedenen Domänen und Ebenen der Automatisierungspyramide. Diese Modifikationen

betreffen sowohl Software-, als auch Hardware-Aspekte. Ein Beispiel für Modifikationen in der Infrastruktur sind die aktuellen Entwicklungen im Bereich der Sensoren und Messgeräte, die als „Sensoren 4.0“ oder „intelligente Sensoren“ bezeichnet werden [90]. Zur Realisierung dieser neuen Feldfunktionen ist eine direkte Verknüpfung der Sensoren mit IT-Instandhaltungsfunktionen, der Geräteverwaltung und den Datenanalysefunktionen erforderlich. Im Zentrum der Entwicklung steht jedoch die Anpassung der Steuerungsgeräte. Sie sind für die operative Prozessführung verantwortlich und verknüpfen die Feldebene mit dem Rest der Welt. Die Modifikation von Steuerungsgeräten ist das Thema verschiedener Forschungen in der industriellen Automatisierung. Diese Modifikationen zielen darauf ab, Steuerungsgeräte zu entwerfen, die interaktiver mit IT-Technologien arbeiten können und neue Funktionalitäten wie z. B. die Integration von Optimierungs- und Managementfunktionen, Kommunikation mit externen Datenanalysesystemen, dynamisches Deployment und Selfx-Technologien anbieten [62].

Deployment ermöglicht die Installation neuer Softwarekomponenten von einer dezentralen Plattform aus. Auf diese Weise kann das Produktionssystem durch die Nutzung der flexiblen Automatisierungsplattformen dynamischer agieren. Einer der Hauptaspekte, der bei der hohen Vernetzung und der Kommunikation mit der IT-Welt berücksichtigt werden muss, ist die Integrität der Kernautomation. Die derzeitigen Automatisierungssysteme bieten ein hohes Maß an Verfügbarkeit und Integrität, sind aber begrenzt miteinander und im Wesentlichen nur untereinander vernetzt. Im Gegensatz zu den heutigen Automatisierungssystemen müssen die zukünftigen Automatisierungssysteme in einer mit der IT-Welt hoch vernetzten Umgebung agieren. Dies ermöglicht eine Vielzahl neuer Funktionen und Geschäftsprozesse. Die hohe Vernetzung darf die Verfügbarkeit oder Integrität des Automatisierungssystems nicht gefährden. Ziel der Entwicklungen sind neue Architekturen, welche die Vorteile heutiger und zukünftiger Automatisierungssysteme kombinieren.

1.2 Zielsetzung

In dieser Arbeit werden die neuen Anforderungen an Steuerungssysteme im Zuge der Digitalisierung erörtert und eine Architektur zur Erfüllung dieser Anforderungen vorgestellt. Die wichtigsten Anforderungen und Aufgaben, die von zukünftigen Steuerungssystemen erfüllt werden müssen, sind:

- die offene Kommunikation mit der IT-Welt
- die Einbindung von Cloud-Mechanismen
- das dynamische Deployment
- das Ermöglichen eines sicheren und leistungsfähigen Datenkanals aus der Feld- und Automatisierungsebene in die übergeordneten IT-Systeme
- Integration der operativen Funktionen aus der MES-Ebene
- der parallele Betrieb von Applikationen unterschiedlicher Kritikalität sowie neue Applikationen, die im Vorfeld nicht vorgesehen waren, auf derselben Hardware
- die Realisierung eines lokalen Software- und Zugriffsverwaltungssystems

- die Realisierung einer Software-Abstraktion zur einfacheren Skalierbarkeit und zur Verringerung der Abhängigkeit von spezieller Software und Hardware

Abb. 1.1 zeigt eine generische Architektur für zukünftige Steuerungssysteme. Um die genannten Anforderungen zu erfüllen wird ein neues Systemkonzept vorgestellt, das durch seine Architektur die beschriebenen Anforderungen grundsätzlich unterstützt. Die vorgeschlagene Architektur in dieser Arbeit bietet eine Trennung der Komponenten auf derselben Hardware mittels Virtualisierung. Die Komponenten werden gemäß ihrer Anforderungen und Quality of Service (QoS) voneinander getrennt [8]. Sie setzt sowohl Container- als auch Hypervisor-Virtualisierungsmethoden ein, um einerseits eine strikte Trennung der Komponenten zu ermöglichen und andererseits ein dynamisches Deployment und Versionsmanagement zu realisieren. Die Komponenten können nur durch festgelegte Kommunikationsschnittstellen miteinander kommunizieren. Die Kommunikationsverbindungen zwischen unterschiedlichen Komponenten sind rückwirkungsfrei und gefährden die Anforderungen der kritischen Anwendungen in keiner Weise. Ein weiteres Merkmal der Architektur ist der Entwurf eines Verwaltungssystems, das lokale Komponentenverwaltungsdienste anbietet. Die Architektur bietet ein dediziertes Interface zur offenen Umgebung. Durch dieses Interface können Daten und Informationen mit externen Komponenten ausgetauscht oder neue Funktionalitäten heruntergeladen werden. Die Partitionen werden gemäß der Anwendungen, welche sie kapseln, konfiguriert und verfügen über entsprechende Fähigkeiten und Zugriffsrechte.

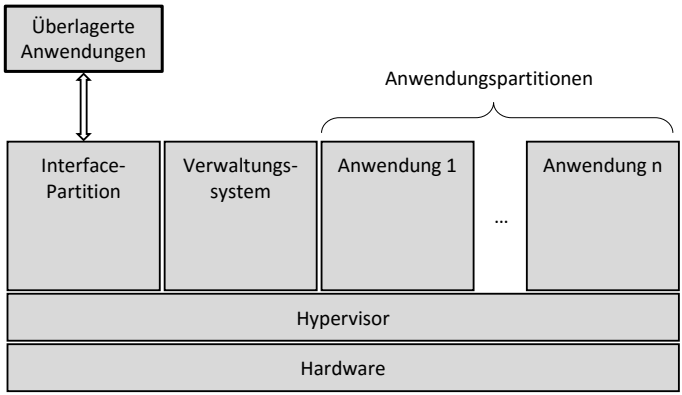

Abbildung 1.1: Eine Architektur für Steuergeräte

1.3 Struktur dieser Arbeit

Diese Arbeit ist wie folgt strukturiert:

- In Kapitel 2 wird der Stand der Technik präsentiert. In diesem Kapitel werden relevante Arbeiten und wichtige Konzepte diskutiert, die für die Zusammenstellung der vorgeschlagenen Architektur notwendig sind.

- In Kapitel 3 werden die Anforderungen an die Architektur erläutert. Der Fokus der Erläuterung liegt auf den Anforderungen, welche durch die neue Generation der Automatisierungssysteme hervorgerufen werden.
- In Kapiteln 4 und 5 wird eine Architektur vorgestellt, um die genannten Anforderungen zu erfüllen. Für die Zusammenstellung der Architektur wird Virtualisierung als eine Grundlage eingesetzt, um den Betrieb verschiedener Anwendungen auf der gleichen Hardware zu ermöglichen.
- In Kapitel 6 wird das Anwendungsszenario und eingesetzte Hardware-Ressourcen für die Implementierung erörtert.
- In Kapitel 7 werden die Implementierung der Architektur und die Implementierungsergebnisse präsentiert.
- In Kapitel 8 wird die Arbeit zusammengefasst und in einem Ausblick bewertet.

2 Grundlage und Stand der Technik

In diesem Kapitel werden einige grundlegende Konzepte der IT und der Automatisierungstechnik erläutert. Diese Konzepte werden für die Realisierung der in dieser Arbeit vorgeschlagenen Architektur eingesetzt. Dieses Kapitel gibt auch einen Überblick über den Stand der Technik dieser Konzepte.

2.1 Virtualisierung

Die Virtualisierung hat grundsätzlich das Ziel, die Ressourcen einer physischen Computer-Hardware zwischen verschiedenen Betriebssystemen (OSs) und Anwendungen aufzuteilen [19], [14], [22], [58]. Die OSs und Anwendungen sind in verschiedene Virtuelle Maschinen (VMs) gekapselt. In der IT wird die Virtualisierung insbesondere in großen Datenzentren und Rechenclustern eingesetzt. Sie ist eine wichtige Grundlage für die Cloud-Technologie [71], [78]. In feldnahen Automatisierungskomponenten und in den eingebetteten Systemen spielte die Virtualisierung in der Vergangenheit keine Rolle. Diese Systeme waren gezielt auf eine Anwendung ausgerichtet und optimiert. Mit den zunehmenden Anforderungen an Flexibilität und Modularität findet die Virtualisierung jedoch auch zunehmend Einzug in diesem Bereich. Darüber hinaus wird sie u.a. in der Avionik, der Fahrzeugtechnik und der industriellen Automatisierungstechnik eingesetzt [3]. Der Einsatz von Virtualisierung ermöglicht eine optimierte Hardware-Nutzung und verhindert unterausgelastete Hardware und CPUs. Dies spielt eine wichtige Rolle für die Skalierbarkeit, für die Reduzierung des Wartungsaufwands und der Kosten usw. [6]. Neben der traditionellen Virtualisierungsmethode, bei der eine Softwareabstraktionsschicht verwendet wird, um VMs auf einer Hardware zu verwalten, können auch Containertechnologien als Virtualisierungsmethode in Betracht gezogen werden. Containertechnologien werden verwendet, um Anwendungen in verschiedene Container zu kapseln. Dies erhöht die Modularität und bietet eine Grundlage für das dynamische Deployment [64]. Allerdings bieten Containertechnologien eine Virtualisierung nur auf der Anwendungsebene an. Im Gegensatz zur traditionellen Virtualisierung, die sich mit der Erstellung von VMs befasst, die ihre eigenen OSs enthalten, kapseln Container nur die Anwendungen und Bibliotheken, die für die Ausführung benötigt werden [31].

2.1.1 Virtualisierungstypen

In der IT werden verschiedene Typen der Virtualisierung eingesetzt. Alle diese Typen zielen darauf ab, die Hardwareressourcen der realen Hardware zu abstrahieren, umfassen aber unterschiedliche Methoden und Spezifikationen, um dieses Ziel zu erreichen. Einige dieser Virtualisierungstypen sind unten aufgeführt [107], [3]:

- Vollvirtualisierung: Bei der Vollvirtualisierung werden die Gast-Betriebssysteme nicht für die Ausführung auf dem Hypervisor modifiziert. Dies führt zu dem Nachteil,

dass die privilegierten Operationen weiterhin unverändert an die Hardware gesendet werden. Um die privilegierten Operationen abzuwickeln, bietet der Hypervisor eine CPU-Emulation, welche Zeit und Ressourcen benötigt.

- Paravirtualisierung: Die Paravirtualisierung wird zur Überwindung des oben genannten Problems eingesetzt. Bei der Paravirtualisierungsmethode sind die Gastbetriebssysteme über die Hardwarevirtualisierung informiert und werden so modifiziert, dass sie nur noch auf Operationen zurückgreifen, die der Hypervisor standardmäßig zur Verfügung stellt. Privilegierte Operationen werden ausgeführt, indem einzelne Anfragen an den Hypervisor gesendet werden. Diese Anfragen oder Aufrufe werden als Hypercalls bezeichnet. Ein Nachteil dieser Methode ist der erforderliche Overhead für die Modifizierung der Gast-Betriebssysteme.

2.1.2 Virtualisierung mit Hypervisoren

Der Hypervisor oder auch Virtual Machine Monitor (VMM) fungiert als eine Middleware zwischen Hardware und Anwendungen (die in VMs laufen) und ermöglicht die Kommunikation zwischen diesen. Hypervisoren werden in zwei Gruppen unterteilt, nämlich Typ-1 und Typ-2. Der Hypervisor Typ-1 kann direkt auf einer Hardware installiert werden, ohne dass ein OS erforderlich ist (siehe Abb. 2.1). Der Hypervisor Typ-2 kann hingegen nur auf einem Host OS installiert werden [29]. Für letzteren können einige VMs generiert werden, während andere Anwendungen parallel auf dem Host OS laufen. Das Konzept der Virtualisierung mit Hilfe eines Hypervisors wurde zuerst von IBM eingeführt. Der Hypervisor von IBM erstellt unabhängige Umgebungen auf der gleichen Hardware (virtuelle Maschine), um Hardwareressourcen sowie Rechenressourcen, Speicher und Netzwerkkonnektivität zu virtualisieren [40]. Die Virtualisierung sollte eine zum ursprünglichen System äquivalente

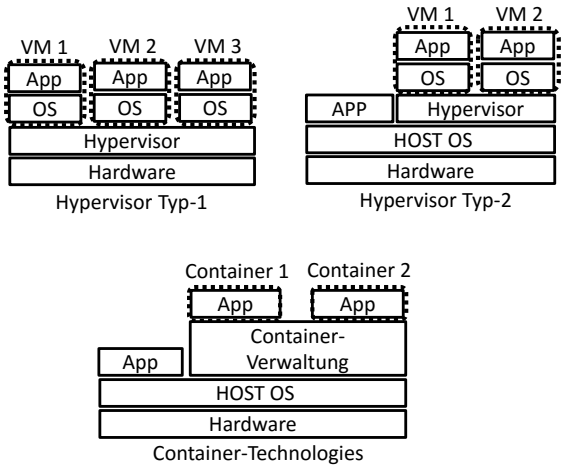

Abbildung 2.1: Virtualisierungsmethoden

Umgebung zur Verfügung stellen. Darüber hinaus muss sie eine zuverlässige Kontrolle über die virtualisierten Ressourcen bieten. Diese Bedingungen gewährleisten einerseits das gleiche Verhalten der Anwendungen in der virtualisierten Umgebung und andererseits die Sicherheit der VMs vor Bedrohungen und Konflikten, die durch die gemeinsame Nutzung einer Hardware verursacht werden. Die Aufgabe der Bereitstellung solcher VMs wird neben dem Hypervisor auch anderen Komponenten wie Mikrokernel zugewiesen. In den folgenden Abschnitten werden diese Komponenten miteinander verglichen.

2.1.3 Virtualisierung mit Mikrokernels

Mikrokernels werden entwickelt, um die Komplexität des Hypervisors zu reduzieren. Die Komplexitätsreduktion basiert auf der Modularisierung. Mikrokernel werden ebenfalls verwendet, um Hardware zu virtualisieren und mehrere VMs auf der gleichen Hardware zu betreiben. Sie unterscheiden sich jedoch in einigen ihrer Eigenschaften von den Hypervisoren. Die wichtigsten Unterschiede liegen in ihrer Abstraktionsebene. Die Hypervisoren verwalten Hardwareressourcen wie Speicher und CPU in Bezug auf die VMs und OSs. Sie befassen sich jedoch nicht mit den Prozessen innerhalb einer VM. Mikrokernels bieten im Gegensatz zu Hypervisoren eine Abstraktion auch für höhere Ebenen wie Tasks und Threads. Das High-Level-Management der Mikrokernels ist mit den folgenden Merkmalen verbunden [3], [51]:

- Threads, Aufgaben und Prozesse: Der Mikrokernel kennt alle Threads, Aufgaben und Prozesse eines OS. Der Hypervisor verwaltet nur die Gast-OSs.
- Speicher: Bei der Speicherverwaltung im Hypervisor werden die Speicherzuordnungen auf Gültigkeit überprüft (die Speicherverwaltung kennt den jeder VM zugewiesenen Anteil an Speicher). Ein Mikrokernel verwaltet den Speicher entsprechend den Aufgaben, die er unterstützt.
- Kommunikation: Hypervisoren bieten Low-Level-Kommunikationsmechanismen, wie z.B. Shared Memory für die Kommunikation der OSs. Solche Low-Level-Mechanismen werden jedoch vom Mikrokernel nicht bereitgestellt.
- Gerätetreiber: Die Hypervisoren enthalten die Kerntreiber, während sich periphere Treiber in den OS befinden. Im Gegensatz zu den Hypervisoren verwalten Mikrokernel die Gerätetreiber als Mikrokernprozesse.
- VM-Management: Die Ausführung von Verwaltungsdiensten wie Create, Read, Delete auf den OSs der Gäste erfordert die Implementierung dieser Dienste auf dem Mikrokernel. Dadurch wird dem Mikrokernel ein Speicher-Overhead hinzugefügt.

Sicherheitskonzepte

Wie bereits erwähnt, wird der Hypervisor im IT-Bereich verwendet, um die Trennung von OSs auf einer Hardware (beispielsweise VirtualBox und VmWare) zu ermöglichen. Diese Trennung bietet keine zertifizierbare Sicherheit. Daher können solche Hypervisoren nicht in sicherheitskritischen Domänen eingesetzt werden, weil sie keine strikte Trennung der Anwendungen verschiedener Kritikalität gewährleisten können. Um die Sicherheit in

solchen Domänen zu gewährleisten, wird eine Kombination von Hypervisoren und Mikrokerneln verwendet [59]. Diese Kombination (wie z. B. PikeOS) wird für Safety- oder Securityanwendungen verwendet (Seperationskernel [117]). Das Sicherheitsniveau solcher Separationskernel orientiert sich an verschiedenen Standards. Die Software-Zertifizierung in verschiedenen Industriebereichen orientiert sich an bestimmten Sicherheitsstandards. Diese definieren verschiedene Kritikalitätsebenen, die in den jeweiligen technischen Domänen unterschiedliche Notationen haben. Zum Beispiel sind bei EN-50128 und IEC 61508 die Safety Integrity Level (SIL) von 0 bis 4 eingestuft, während die Stufen in ISO 26262 von A bis E unterteilt sind. Das Safety-Konzept umfasst die Trennung der Anwendungen mit unterschiedlichen Kritikalitätsstufen sowie die Überwachung des Informationsflusses zwischen diesen. Das Echtzeitverhalten der OS ist ebenfalls ein safetyrelevanter Aspekt. Den safetykritischen Anwendungen muss ein detailliertes Schedulingschema zugeordnet werden, damit diese zuverlässig arbeiten können [59].

Die Security befasst sich mit dem Schutz des Systems vor ungewollten Manipulationen oder Datenzugriffen z.B. durch Cyberangriffe. Analog zu den Safetyanforderungen, basieren auch die Securityanforderungen auf Standards in verschiedenen Bereichen. Bei der Security geht es sowohl um die Trennung von Anwendungen auf derselben Hardware als auch um die Kontrolle des Informationsflusses zwischen diesen. IEC 15408 (gemeinsame Kriterien) und IEC 62443 sind zwei Beispiele für Securitystandards. Gemeinsame Kriterien beschreiben Securityanforderungen für allgemeine Zwecke und IEC 62443 beschreibt Securityanforderungen von Industrial Automation and Control System (IACS)s. Die Computersysteme haben unterschiedliche Prozessphasen. Spezifikation, Implementierung und Evaluation sind drei wichtige Phasen bei der Entwicklung von Computersystemen. Gemeinsame Kriterien stellen sicher, dass all diese Phasen gemäß dieser Normen durchgeführt werden. Dieser Standard definiert mehrere Evaluation Assurance Levels (EAL)s von 1 bis 7. Separationskernels (je nach Technologie) bieten eine höhere Sicherheit (EAL6) als Allzweckrechenplattformen (EAL4) [59].

Es werden verschiedene Technologien und Konzepte entwickelt, um Separationskernel zu entwerfen, welche die Anforderungen bestimmter Domänen erfüllen. Zum Beispiel definierten Chung-Wei, BaekGyu und Shinichi [65] einen Ansatz zur Hardware-Virtualisierung und Aufgabenzuweisung für Automobilsysteme und die Autoren in [81] definierten eine Architektur für leistungsarme eingebettete Echtzeitsysteme. Im nächsten Abschnitt werden einige Hypervisor- und Mikrokerneltechnologien gemeinsam mit der in dieser Arbeit verwendeten Technologie vorgestellt.

2.1.4 Hypervisor und Mikrokernel-Technologien

Im Folgenden werden einige Hypervisoren (Typ-1 und Typ-2) und Mikrokernel, die in der Industrie eingesetzt werden, präsentiert.

VirtualBox

VirtualBox ist ein Hypervisor vom Typ-2. Sie kann auf verschiedenen Host OSs wie Windows, Linux, Solaris installiert werden und unterstützt dabei eine unterschiedliche Anzahl von Guest OSs. Es handelt sich um eine von der US-amerikanischen Firma Oracle entwickelte Virtualisierungssoftware [76].

Xen Hypervisor

Xen ist ein Open-Source-Hypervisor. Xen umfasst eine Softwareschicht, die virtuelle Ressourcen implementiert und auch den I/O-Zugriff kontrolliert. Die VMs werden in Xen Domänen genannt. Xen besitzt eine Domäne null VM, die andere VMs erzeugen und löschen kann und die I/O-Geräte den VMs zuordnet [13].

PikeOS

PikeOS ist ein mikrokernelbasierter Hypervisor, der von der SYSGO GmbH entwickelt wurde. Es besteht aus einem Real Time Operating System (RTOS), einer Virtualisierungsplattform und einer Eclipse-basierten Entwicklungsumgebung. Dieser Hypervisor wird für Sicherheitskritische Anwendungen in den Bereichen der Luft- und Raumfahrt, der Verteidigung, der Fahrzeugtechnik, der industriellen Automatisierung, etc eingesetzt [96]. Er ermöglicht die Ausführung verschiedener Anwendungen, mit unterschiedlichen Sicherheitsstufen, auf derselben Hardware in verschiedenen Partitionen. Die Partitionen bieten unterschiedliche Portierungsmöglichkeiten. Diese Möglichkeiten werden als Personalities bezeichnet:

- **Native** ist eine direkte Verknüpfung der Anwendung mit PikeOS-Service-Schnittstellen. Diese erfordern eine minimale Anpassung der Anwendung. Da es sich jedoch um eine proprietäre Anwendung von PikeOS handelt, müssen andere Anwendungen, die nicht für PikeOS entwickelt wurden, modifiziert werden, um mit PikeOS kompatibel zu werden.
- **Die Portable Operating System Interface (POSIX)-Personality** verwendet das Standard Portable Operating System Interface POSIX. Viele UNIX OSs sind konform zu diesem Standard. POSIX wurde in der ISO/IEC 9945 normiert. Die POSIX-Personality für PikeOS entspricht dem PSE52-Profil des IEEE Std 1003.13-1998 mit zusätzlichen Echtzeit-Erweiterungen.
- **ElinOS** ist eine paravirtualisierte Linux-Distribution. Diese Personality kann durch die Installation von Softwarepaketen erweitert werden. Es ist der einfachste Weg, Anwendungen, die für Linux implementiert sind, zu portieren. Aber diese Schnittstelle ist wegen des Overheads auf einem eingebetteten System vergleichsweise langsam.

Das PikeOS RTOS basiert auf einem modularen Ansatz. Es besteht aus einem Mikrokernel, der die folgenden Dienste zur Verfügung stellt [80], [44]:

- Hardware-Abstraktion
- prioritätsbasiertes Echtzeit-Scheduling
- Ausführungseinheiten (Threads)
- getrennte Adressräume (Aufgaben)
- Kommunikationsprimitive
- Timer und Ausnahme- und Interruptbehandlung.

Scheduling und Zeitpartitionierung

Der PikeOS-Scheduler basiert auf dem Prinzip der Zeitpartitionen. Zeitpartition stellt dabei einen Mechanismus zur Verteilung von Rechenzeit auf Anwendungen (Partitionen) dar. Die getrennten Anwendungen in den Partitionen müssen, mit den für ihre Ausführung erforderlichen Ressourcen, versorgt werden. Dabei garantiert die Zeitpartitionierung, dass alle Partitionen einen bestimmten Anteil an Ausführungszeit erhalten. Die Ausführungszeit ist vordefiniert und kann während der Laufzeit nicht dynamisch verändert werden.

Die Zeitpartitionierung ist in Abb. 2.2 dargestellt. Die Aufteilung der Rechenzeit erfolgt in zwei Schritten. Im ersten Schritt werden die entsprechenden Partitionen einer Zeitpartition zugeordnet. Dann werden diese Partitionen einem oder mehreren Fenstern zugeordnet, wobei jedes Fenster eine eigene Dauer hat. Der Zeitplan setzt sich aus diesen vordefinierten Fenstern zusammen. Die Fenster können auf verschiedenen Central Processing Unit (CPU)-Kernen laufen. Aber an jedem Punkt ist auf jedem Kern nur ein Fenster aktiv. Die vorhandenen Threads innerhalb einer Partition können mit unterschiedlichen Prioritäten zugewiesen werden. Unter allen Threads wird jedes Mal der Thread mit der höchsten Priorität ausgeführt. Ein Thread enthält auch andere Informationen.

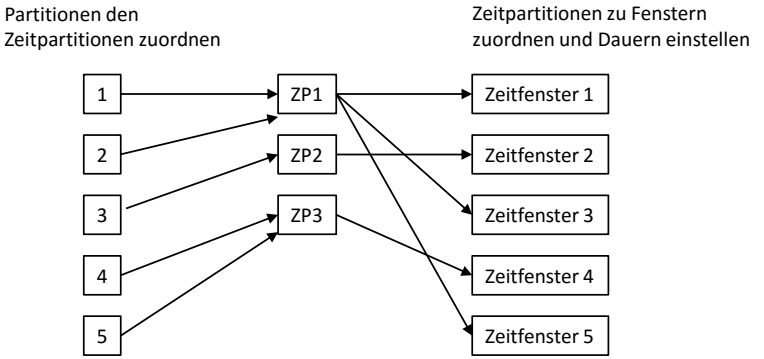

Abbildung 2.2: PikeOS-Zeitpartitionierung

Kommunikationen zwischen den Partitionen in PikeOS

PikeOS bietet verschiedene Möglichkeiten für die Kommunikation zwischen Partitionen. Die folgende Liste enthält die relevanten Kommunikationsmedien:

- Queuing Ports: Diese fungieren als First In First Out (FIFO)-Kommunikation, bei der Ports den Endpunkt der Kommunikation darstellen. Die Kommunikation mit Hilfe von Queuing Ports ist eine unidirektionale Kommunikation. Queueing ports ermöglichen Service Access Points (SAP) und eine paketbasierte Echtzeit-Kommunikation wie User Datagram Protocol (UDP)/Internet Protocol (IP).
- Shared memory: Dieser Kommunikationstyp bietet einen physischen Speicher für den Datenaustausch zwischen Partitionen. Die Zugriffsrechte auf den gemeinsamen Speicher müssen in der Engineering-Phase statisch zugewiesen werden.

- File System (FS): Diese Methode wird verwendet, um die aktuellen Daten auf einem Speichergerät abzubilden. Sie kann auch außerhalb einer Partition als Schnittstelle zum Lesen/Schreiben von Daten verwendet werden. Read Only Memory (ROM) und FS sind Beispiele für ein eingebautes Dateisystem.

WindRiver Hypervisor

Wie der Name schon suggeriert, besteht im WindRiver-Hypervisor die Integration mit WindRiver Linux. Er unterstützt auch andere OSs wie Windows 7 (32-bit und 64-bit Single-Core und Multi-Core) und Red Hat Linux. Es ist auch möglich, andere Betriebssysteme hinzuzufügen. Die unterstützten Prozessorarchitekturen sind Intel Atom, ARM und PowerPC. Die Zeitplanung kann prioritätsbasiert, zeitlich partitioniert oder benutzerdefiniert sein [112]. Einige Funktionen von WindRiver Hypervisor sind unten aufgeführt [113]:

- **Virtual Board Management:** Virtual Board Management ermöglicht die Verwaltung virtueller Boards, indem es Funktionen wie Create, Delete, Read zur Verfügung stellt.
- **Core Scheduling:** Priorität- und zeitbasiertes Scheduling.
- **Safety Profile:** Zertifizierung nach Normen, wie IEC 61508 (bis zu SIL 3).

Tabelle 2.1 gibt einen Überblick der Eigenschaften von PikeOS, Windriver und Xen Hypervisor.

Tabelle 2.1: Vergleich der Eigenschaften

PikeOS	WindRiver	Xen
CC EAL 3+	No CC EAL 3+	No certification
Certified Posix	No Certified Posix	No Certified Posix
Health Monitor	Health Monitor	no Health Monitor
Certifiable Hypervisor	Only ARMv8 und x86	no certification

Neben den genannten Technologien werden auch andere Separationskernel wie VxWorks 653, LynxSecure, INTEGRITY-178B und LynxOS-178 in verschiedenen Bereichen, wie der Fahrzeugtechnik und der Luft- und Raumfahrt eingesetzt. Einige Technologien haben Security- und Safetyfunktionalitäten in denselben Separationskernel integriert, wie z.B. PikeOS und LynxSecure.

2.2 Container-Technologien

Containertechnologien befassen sich mit der Isolierung und Kapselung von Anwendungen und notwendiger Bibliotheken in verschiedenen Containern. Das Ziel dieser Technologien ist, leichtgewichtige Container zu erzeugen, die wie Daten transportiert werden können. Aus diesem Grund enthalten Container keine OS und sie laufen alle auf dem Host OS. Dies kann auch als eine Virtualisierungsmethode angesehen werden. Der Vorteil von Containertechnologien besteht darin, dass die leichtgewichtigen Container und Container-Images

einfach von einem System auf ein anderes übertragen werden können. Dies reduziert den Aufwand für die Installation und Aktualisierung einer Software. Container verhalten sich ähnlich wie eine VM. Die in Abb. 2.1 dargestellte Container-Management-Einheit erlaubt eine Hardware-Abstraktion auf der Anwendungsebene. Dezentralisierung, Skalierbarkeit und dynamisches Deployment sind die Hauptvorteile der Containertechnologien [72].

Docker-Container

Docker ist eine Open-Source Plattform für die Ausführung und Entwicklung von Applikationen. Dort werden Applikationen in so genannte Container gekapselt. Docker-Container sind ein Beispiel für Containertechnologien [31]. Die Dockertechnologie ermöglicht das Konzept der Containerisierung auf dem Server. Applikationen können nach der Kapselung den anderen Anwendern zur Verfügung gestellt werden. Darüber hinaus stellen die Container Test-Umgebungen für die Applikationen bereit [85], [16]. Die Struktur der Dockertechnologie ist in Abb. 2.3 dargestellt. Docker besteht aus folgenden Hauptkomponenten [100]:

- **Docker Server und Klient**: Der Docker Server (Daemon) und Klient kommunizieren über ein RESTful API. Der Klient sendet Anfragen an den Server, um beispielsweise einen bestimmten Container herunterzuladen.
- **Docker Images**: Docker Images sind die Applikationen, die in Docker Containern gekapselt werden.
- **Docker Registry**: Die Docker Registry ist ähnlich, wie ein Repository. Images können beispielsweise durch Push, Pull und Build Befehle verwaltet werden.
- **Docker Container**: Docker Container beinhalten die Anforderungen und Abhängigkeiten für die Ausführung der Applikationen.

Diese Art der Virtualisierung spielt eine wichtige Rolle für das Cloud Computing [101].

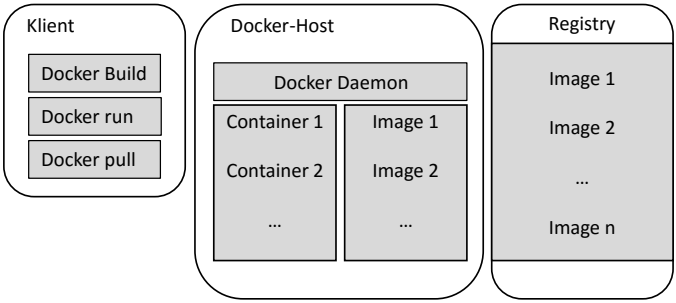

Abbildung 2.3: Docker-Technologie [32]

2.2.1 Virtualisierungsanwendungen in anderen industriellen Domänen

Echtzeit-Hypervisor werden in verschiedenen Bereichen wie der Avionik und der Fahrzeugtechnik eingesetzt [25], [83], [104]. Echtzeitfähigkeit, Verfügbarkeit, Safety/Security usw. sind wichtige Anforderungen, die den Hypervisor zu einer geeigneten Lösung in diesen Domänen machen. Nach der Entwicklung von AUTomotive Open System ARchitecture (AUTOSAR) und dem autonomen Fahren hat auch die Automobilindustrie begonnen, den Hypervisor für verschiedene Zwecke wie Safety und Security einzusetzen [66], [7]. Ähnlich wie die Controller in Automatisierungsdomänen (Speicherprogrammierbare Steuerung (SPS)en) muss auch der Bordcomputer der Fahrzeuge an die Cloud angeschlossen werden. Somit ist die Vernetzung auch eine Anforderung bei der Automatisierung von Fahrzeugen. In den nächsten Abschnitten wird die Anwendung des Hypervisors im Automobil- und Avionikbereich diskutiert.

Automotive Open System Architecture

AUTOSAR zielt darauf ab, eine hardwareunabhängige und standardisierte Anwendungssoftware für Electronic Control Units (ECU)s bereitzustellen. Bei den ECUs handelt es sich um Steuergeräte für die Fahrzeuge. AUTOSAR wurde 2003 gegründet und das erste Release des entsprechenden AUTOSAR-Konzepts wurde 2005 entwickelt. Seit 2005 wurde AUTOSAR kontinuierlich weiterentwickelt und an die unterschiedlichen Anforderungen in der Automobilindustrie angepasst (beispielsweise die Entwicklung der AUTOSAR Adaptive Platform [37]). Es besteht aus einer Basissoftware, einer Laufzeitumgebung und einer Anwendungssoftware. Abb. 2.4 zeigt die Architektur von AUTOSAR. Das Ziel dieser Architektur ist die Hardwareunabhängigkeit der Anwendungssoftware.

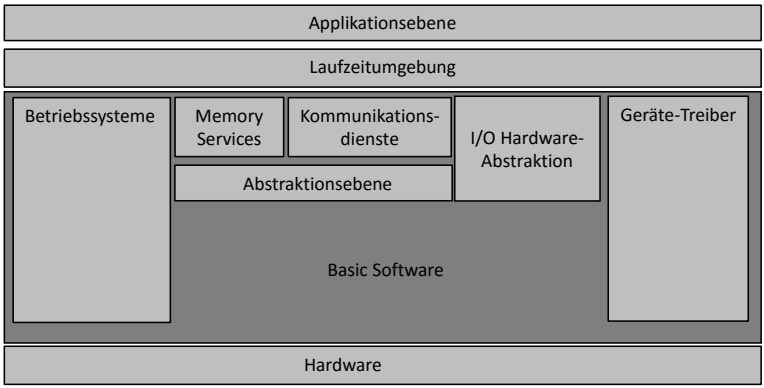

Abbildung 2.4: AUTOSAR-Architektur

Sie besteht aus:

- einer hardwareunabhängigen Anwendungsschicht
- einer Laufzeitumgebung, welche die Schnittstellen für die Anwendungen bereitstellt

- einer Basissoftware, welche Dienste und die Abstraktionsschicht enthält.

Kürzlich wurden einige neue Anforderungen für die AUTOSAR-Architektur definiert, die beim Entwurf der initialen Architektur nicht berücksichtigt wurden. Diese Architektur sollte auf derselben ECU-Hardware auch verschiedene Funktionen mit unterschiedlichem Sicherheitsniveau erfüllen. Dies entspricht einer Kombination verschiedener Automotive Safety Integrity Level (ASIL)-Ebenen auf derselben Hardware. Sie muss auch eine sichere Integration von Software verschiedener Anbieter ermöglichen. PharOS ist eine Lösung, die in [63] diskutiert wird, um eine Software-Partitionierung zur Bewältigung dieses Problems bereitzustellen. Wegen des zunehmenden Interesses am Betrieb von Anwendungen mit unterschiedlicher Kritikalität und Anforderungen auf derselben Hardware werden Hypervisoren in verschiedenen Domänen eingesetzt [27], [81]. Es gibt verschiedene Ansätze, um den Betrieb und die Trennung von Anwendungen unterschiedlicher Kritikalität sicherzustellen. Darüber hinaus können verschiedene Arten von Hypervisoren eingesetzt werden, um diese Trennung und die Ressourcenzuweisung zu gewährleisten. Als Beispiel wird in Abb. 2.5 der Betrieb von Anwendungen mit unterschiedlichem ASIL-Level auf derselben Hardware auf der Basis von VOSYSmonitor dargestellt [66]. Die dargestellte Architektur priorisiert die sicherheitskritischen (ASIL) Anwendungen, während nicht-kritische Anwendungen (keine ASIL) parallel auf derselben Hardware laufen.

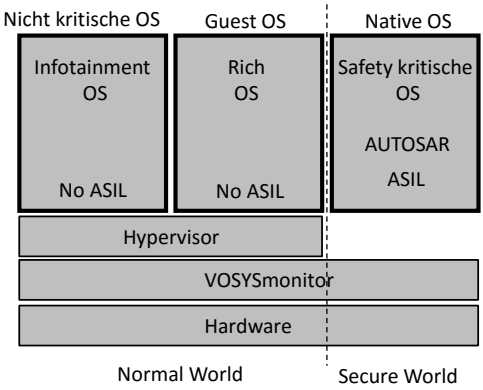

Abbildung 2.5: VOSYSmonitor

Den sicherheitskritischen Anwendungen sind bestimmte Ressourcen zuzuordnen. Diese Ressourcen sind für eine Nutzung anderer Anwendungen gesperrt, bis der VOSYSmonitor wieder eine Freigabe für diese erteilt. Anforderungen an ECUs und industrielle Steuerungsgeräte überschneiden sich in zentralen Aspekten. Beide erfordern die Integration von Anwendungen mit unterschiedlichen Anforderungen auf der gleichen Steuerungshardware, unter Berücksichtigung der Safety und Security. Die Safety und Security in beiden Bereichen orientiert sich an bestimmten Standards.

2.2.2 Virtualisierung in der Luftfahrt

Virtualisierung wird auch in der Luftfahrt eingesetzt, um eine sichere Trennung von Anwendungen zu gewährleisten. Abb. 2.6 zeigt ein Beispiel für eine solche Trennung zwischen Anwendungen. ARINC 653 ist ein Hypervisor, der für diese Trennung eingesetzt wurde.

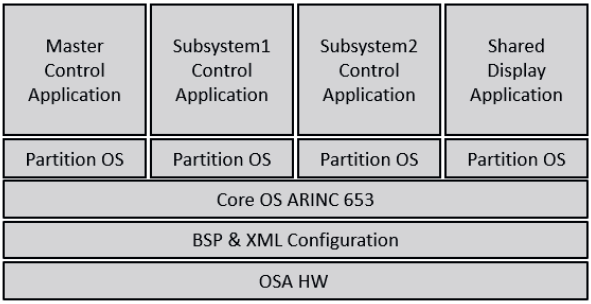

Abbildung 2.6: Virtualisierung in der Luftfahrt

2.2.3 Industrielle Automatisierung

Die Automatisierungspyramide bietet ein hierarchisches Ebenenmodell zur Kategorisierung der Aufgaben und Funktionalitäten innerhalb eines Produktionssystems von den Feld- bis zu den Geschäftsprozessen.

Die unterste Ebene stellt die Schnittstelle zu den Feldgeräten (Sensoren und Aktoren) dar. Die zweite Ebene wird als Prozessleitebene bezeichnet. Sie umfasst die SPS-Systeme, welche zur Implementierung der Prozessführungsanwendungen verwendet werden. Die dritte Ebene stellt Produktionsrezepte zur Verfügung und bietet Dienste, wie Scheduling, Predictive Maintenance und Ressourcenverwaltung an. Abschließend enthält die Enterprise Resource Planning (ERP)-Ebene die Geschäftsprozesse. Die Kommunikation in der Automatisierungspyramide erfolgt nur über definierte Schnittstellen zwischen den Ebenen [52], [57].

Zukünftige Automatisierungssysteme erfordern jedoch eine hohe Vernetzung, um Ziele wie Agilität zu erreichen. Aus diesem Grund wird die hierarchische Struktur der Automatisierungspyramide in eine hoch vernetzte Architektur aufgelöst, so dass alle Komponenten unabhängig von ihrem Automatisierungsgrad miteinander kommunizieren können. Die Auflösung betrifft nur die Kommunikationsperspektive. Das bedeutet, dass die klassischen Automatisierungsebenen weiter bestehen, während die Kommunikationsbeschränkungen aufgehoben werden [15], [43], [108]. Das bedeutet, dass alle beteiligten Komponenten direkt miteinander kommunizieren können. Die Vernetzung birgt neue Anforderungen an prozessnahe Komponenten.

Die zukünftigen Automatisierungssysteme müssen kooperativer mit IT-Systemen agieren. Die IT bietet Dienste zur Datenanalyse, Optimierung usw. an, während die Automatisierungstechnik weiterhin ihre klassischen Aufgaben beibehält. Ein wichtiger Aspekt

dieser Kooperation ist die Überwachung des Informationsflusses. Hierzu stellte NAMUR das Konzept einer Informationsdiode vor (NAMUR-Diode).

2.3 NAMUR Open Architecture

In der NAMUR Open Architecture (NOA) werden die Funktionen den Clustern Monitoring and Optimization (M+O) und Core Process Control (CPC) zugeteilt [74]. CPC enthält die klassischen Automatisierungsfunktionen. Sie sind direkt für die korrekte und sichere Steuerung der Prozesse zuständig und dürfen nicht durch externe Einflüsse gestört werden. M+O enthält alle Funktionen, die zur Optimierung und zum Management ergänzend angeboten werden. Das NOA-Konzept ist in Abb. 2.7 dargestellt.

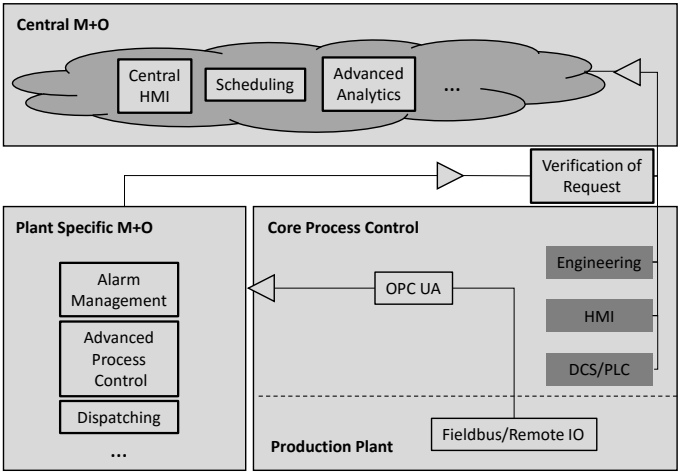

Abbildung 2.7: NAMUR Open Architecture [74]

Das CPC in der NOA enthält die zentralen Automatisierungskomponenten der ersten und zweiten Automatisierungsebene (wie SPSen, HMIs, DCS, Sensoren und Aktoren). Die Anlagenspezifische M+O enthält IPCs, Edge-Devices und M+O Sensoren. In ihr werden Informationen gespeichert, aggregiert, vorverarbeitet. Die Weiterverarbeitung der Informationen erfolgt in der zentralen M+O.

CPC und M+O kommunizieren über zwei verschiedenen Kommunikationsverbindungen, die jeweils unterschiedliche Anforderungen haben. Diese Kommunikationsverbindungen sind in Abb. 2.8 dargestellt.

Bei der Kommunikation zwischen der Automatisierung und der IT dürfen die Informationen in der Richtung von CPC nach M+O ungehindert fließen. Dies wird durch die Einführung des Konzepts der NAMUR-Diode realisiert. Die Anforderungen an die Diode wurden in [75] vorgestellt. Sie lauten wie folgt:

- Keine direkte Verbindung zwischen CPC und M+O

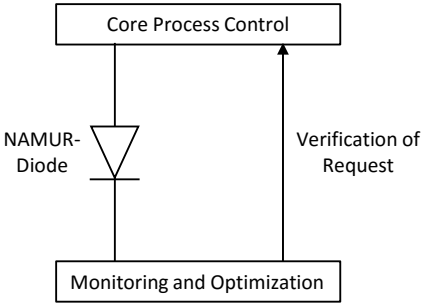

Abbildung 2.8: Kommunikation zwischen CPC und M+O

- Informationsfluss nur von CPC zu M+O und nicht umgekehrt
- Keine Rückwirkung auf die CPC-Parameter
- Keine Konfiguration oder Parameter-Manipulation des CPC aus der M+O-Domäne über die NAMUR-Diode

Nach dem Abruf und Verarbeitung der Daten in der M+O-Ebene wird ein Feedback erzeugt. Dieses kann in einem eigenen streng überwachten Kanal in die CPC-Ebene zurückgespielt werden (Verification of Request (VoR) in der Abb. 2.8). Die Integration des Feedbacks muss auf sichere Weise durchgeführt werden. Die zurückgeführten Daten müssen auf Plausibilität und Authentizität geprüft werden. Beispielsweise darf das Feedback keine Interlocks triggern. In [24] ist eine Komponente entworfen worden, welche das Feedback vor der Integration validiert. Diese Validierung findet nicht nur beim Feedback sondern auch darüber hinaus bei der deployten Komponenten statt.

2.4 Speicherprogrammierbare Steuerungen

Die SPS ist ein Gerät zur Steuerung eines Produktionssystems. Die Geschichte der SPSen lässt sich bis auf das Jahr 1968 zurückverfolgen. Fest verdrahtete Relaistafeln wurden zu diesem Zeitpunkt durch halbleiterbasierte sequentielle Logiksysteme ersetzt. SPSen bestehen jeweils aus Eingängen, Ausgängen und einem Betriebssystem (Abb. 2.9). Anwendungen bestimmen die Beziehung zwischen den Eingängen und Ausgängen. Sie können über eine Schnittstelle geladen werden. Darüber hinaus können sie hardwareunabhängig manipuliert, programmiert und umprogrammiert werden. Abgesehen von der Prozessführung kann eine SPS Laufzeitdaten überwachen, um je nach Anwendung die erforderlichen Maßnahmen zu ergreifen, wie z.B. Alarmgenerierungen oder Starten und Stoppen anderer Prozesse [111], [97].

2.4.1 Programmierung

Für SPSen wurden im Laufe der Zeit eine Vielzahl von Programmiersprachen entwickelt. Anweisungslisten, Kontaktpläne, Funktionsbausteinsprachen, Ablaufsprachen und Struk-

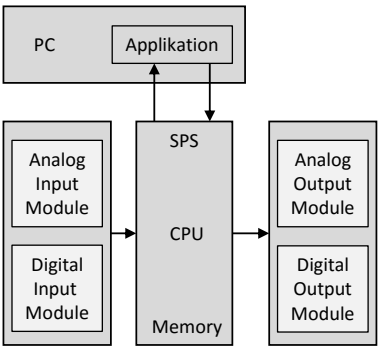

Abbildung 2.9: SPS-Struktur

turierte Texte sind ein paar Beispiele für diese Programmiersprachen. Die genannten Sprachtypen sind in IEC 61131-3 genormt, um eine standardisierte Programmierung für SPSen zu ermöglichen. SPSen werden je nach Anwendungsgebiet mit unterschiedlicher Anzahl von I/Os (binär oder analog), und zusätzlichen Funktionen, wie PID-Regler, Timer, Zähler angeboten. Je nach Anzahl der I/Os und internen Fähigkeiten werden sie als Micro SPS, Small, Medium und Large kategorisiert. SPSen, Industrie-PC (IPC)s und eingebettete Systeme werden alle in der industriellen Automatisierung eingesetzt. Ihre Basisstrukturen sind analog zueinander (Eingang, Ausgang und eine Logik). Im nächsten Abschnitt werden diese miteinander verglichen.

2.4.2 Entwicklung der speicherprogrammierbaren Steuerungen

SPSen wurden entwickelt, um fest verdrahtete Regelkreise zu ersetzen. Die Änderung der Steuerungsanwendung in solchen Regelkreisen erforderte zeitintensive Verfahren. Dies rief die Entwicklung softwarebasierter Steuerungsprogrammierung hervor, so dass die Steuerung unabhängig von der Hardware manipuliert werden kann. SPSen ermöglichen dies durch ihre Programmiersprachen wie Continuous Function Charts (CFC)s oder Sequential Function Charts (SFC)s. Die SPSen wurden im Laufe der Jahre kontinuierlich weiterentwickelt, und an die ständig wachsenden Anforderungen der Produktionssysteme angepasst.

Die ersten Entwurfskriterien für SPSen wurden von der General Motors Corporation spezifiziert. Diese Spezifikationen, die im Zusammenhang mit I3.0 betrachtet werden können, sind nachstehend aufgeführt [91]:

- Einfach zu programmieren und umzuprogrammieren
- Niedriger Wartungsaufwand
- Betriebsfähig in der Anlagenumgebung
- Kompakt
- Kommunikationsfähig mit höheren Automatisierungsebenen

- Niedrige Kosten

Im Jahr 1980 haben viele Unternehmen damit begonnen, SPSen für ihre Anwendungsfälle einzusetzen. In der Anfangszeit waren sie proprietäre und lokale Geräte. In den 90er Jahren begann die Standardisierung von SPSen. Ethernet-Netzwerke und die Entwicklung von Flash-Speichern waren die nächste Evolution der SPS-Systeme. Seit 1968 bis heute haben sich die Anforderungen und Spezifikationen von SPSen geändert und die Hersteller von SPSen haben versucht, die Technologie an die Fertigungsanforderungen dementsprechend anzupassen. Darüber hinaus wurden im Laufe der Jahre auch die Programmiersprachen der SPSen von der Leiterlogik zu den IEC 61131-3 Funktionsbaustein (FB)s und deren Erweiterung IEC 61499 weiterentwickelt. Eins der wichtigsten Ziele in der SPS-Evolution ist Flexibilität. Verschiedene Automatisierungsanforderungen haben im Laufe der Zeit eine neue Flexibilitätsniveau von SPSen gefordert. Anforderungen wie die Optimierung von Fertigungsprozessen, die Verkürzung der Time-To-Markt, die Massenproduktion in der dritten Automatisierungsrevolution und die hohe Agilität in I4.0 haben alle im Laufe der Zeit zu Neuinterpretationen des Begriffs Flexibilität geführt. Anfangs bezog sich Flexibilität auf die Entwicklung von Methoden zur softwarebasierten Manipulation der Steuerung. Im Zusammenhang mit I40 haben sich die Anforderungen an die Flexibilität drastisch geändert. Flexibilität wurde durch Agilität ersetzt. Ein agiles Produktionssystem muss in der Lage sein, auf Änderungen zu reagieren, die in der Entwurfsphase nicht berücksichtigt wurden. Dies unterscheidet die Agilität von der Flexibilität. Die Anforderungen und die Spezifikation von SPSen im Kontext von I4.0 werden im folgenden Abschnitt erörtert.

2.4.3 Neue Architekturen für speicherprogrammierbare Steuerungen

Aktuell werden einige neue Konzepte und Architekturen für SPSen definiert. Diese Architekturen zielen darauf ab, die Flexibilität der SPSen zu erhöhen. Die Ansätze besitzen unterschiedliche Schwerpunkte. Der Schwerpunkt kann zum Beispiel auf der Entwicklung neuer Software-Architekturen, der Integration neuer Hardwarekomponenten, strukturellen Änderungen des Programmierungsmodells (beispielsweise der Ansatz in [92]) oder der Entwicklung von real-time Betriebssystemen für die eingebetteten Systemen liegen [54], [4]. Einer dieser Ansätze ist die virtuelle SPS (vPLC) [39]. Das Konzept der vPLCs wird in [26], [38] erörtert. vPLCs werden als Anwendungen in einer Cloud implementiert und steuern die physische SPS-Hardware mit Hilfe von Cloud-Diensten. vPLCs liefern die Steuerungsfunktionen als Dienste an die Feldebene und steuern diese über das Netzwerk. Bei den vPLCs im Netzwerk handelt es sich um unterschiedliche Steuerungslogiken, die mit der Hardware-SPS kommunizieren. Dieses Konzept ist in Abb. 2.10 dargestellt. Beim Betrieb einer vPLC ist die Security ein entscheidender Faktor, der berücksichtigt werden muss. Um die Security des Produktionssystems zu gewährleisten und unberechtigte Zugriffe zu verhindern, wird eine private Cloud eingesetzt. Die Anbindung einer SPS an eine öffentliche Cloud kann zu Bedrohungen und unberechtigten Zugriffen führen. Ein ähnlicher Ansatz wurde in [49] unter Verwendung von AMAZON-Diensten realisiert. Cloud-basierte Ansätze werden auch im Bereich der Robotik verwendet. [23] setzt eine Cloud ein, um Steuerungsdienste für Roboter bereitzustellen. [114] bietet einen Überblick über den Stand der Technik der cloudbasierten Fertigung. Die Vor- und Nachteile der Cloud-basierten Strategie werden in [2] diskutiert. Kosteneffizienz, unbegrenzter Speicherplatz, einfacher Zugang sind einige Vorteile dieser Strategie. Die Security ist bei diesem Ansatz ebenfalls ein wichtiges Thema.

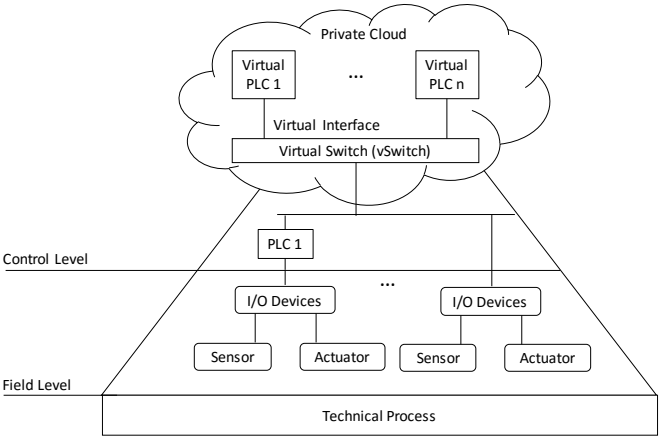

Abbildung 2.10: Control as a Service [39]

Die vPLCs und Cloud-as-a-service Applikationen haben allerdings hohe Latenzen. Dieses Problem wird in [82] geschildert. Eine Architektur basierend auf Miroservices wurde in [33] vorgestellt. In ihr werden Container-Technologien für die Kapselung der Applikationen eingesetzt. Die vorgestellte Architektur in [36] stellt ein Gateway bereit, das die Geräte mit der Cloud verbindet und die Rolle einer SPS spielt. Die Verbindung mit der Cloud erfolgt über MQTT-Nachrichten oder virtuellen Instanzen. Eine cloudbasierte Architektur namens soft-PLC wurde in [42] vorgestellt. Bei soft-PLC läuft die SPS als eine Software in einer Cloud und empfängt Prozessdaten mittels OPC UA. Die Architektur ermöglicht eine horizontale Skalierbarkeit. Der Nachteil dieser Methode ist die hohe Latenz zwischen der Cloud und den Feldgeräten. Die in dieser Arbeit vorgeschlagene Architektur ist im Gegensatz zu vPLCs keine cloudbasierte Architektur.

Eine weitere Strategie ist der Einsatz von Virtualisierungstechnologien. Die Herausforderungen beim Einsatz von der Virtualisierung und Hypervisoren im Bereich der eingebetteten Echtzeitsysteme und Legacy-Systeme werden in [86], [20] diskutiert. Die Hypervisor-Technologie bietet eine begrenzte Granularität [50]. Zukünftige Automatisierungssysteme müssen so gestaltet werden, dass sie eine höhere Granularität bieten. Um dieses Problem zu überwinden, wurde in [41] ein Ansatz zur Erhöhung der Flexibilität von SPSen auf der Grundlage der Container-Technologien vorgestellt. Dabei werden Container-Technologien eingesetzt, um neue Funktionen auf die SPS herunterzuladen. Der Schwerpunkt der Architektur liegt auf der Containerisierung und dem Containermanagement, sowie dem Container-Deployment (Funktionalitäten). Die Virtualisierung mit Hilfe von Container-Technologien ist eine Grundlage für diese. Abb. 2.11 stellt die diskutierte Architektur dar. Wie gezeigt, können verschiedene Anwendungen wie z.B. IEC 61131-Anwendungen von einem Container-Deployer auf die SPS heruntergeladen werden. Ein Container-Registry stellt eine Liste aller Container bereit. Die Ressourcen werden vor dem Deployment von einem Deployment-Koordinator auf Verfügbarkeit analysiert.

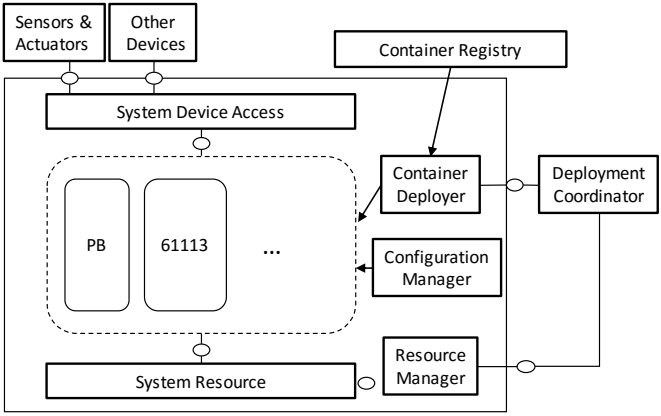

Abbildung 2.11: Multi Purpose Controller

Vorgeschlagene Architektur in dieser Arbeit und eigene Vorarbeiten

Die Grundlage der vorgeschlagenen Architektur wurde in [8] vorgestellt. Diese Architektur kombiniert Container- und Hypervisor-Virtualisierungsmethoden zur Erreichung der beiden wesentlichen Aspekte: strikte Trennung und dynamisches Deployment der Anwendungen zur Laufzeit. Ein weiteres Merkmal der Architektur ist der Entwurf eines Verwaltungssystems, das lokale Komponentenverwaltungsdienste anbietet. Diese Dienste bestehen unter anderem aus Kommunikationsdiensten (beispielsweise zwischen Anwendungen unterschiedlicher Kritikalität), Deploymentdiensten und Ressourcenzuteilungsdiensten. Die genannten Dienste und ihre relevanten Konzepte wurden in folgenden Veröffentlichungen erläutert. In [69] und [11] wurde die Realisierung einer rückwirkungsfreien Kommunikation basierend auf dem NAMUR-Diode-Konzept diskutiert. Dabei wurde eine unidirektionale FIFO-Kommunikation (Queue) für die Realisierung der rückwirkungsfreien Kommunikation eingesetzt. In [11] wurde die Validierung der deployten Komponenten vor Inbetriebnahme in den Fokus gestellt. Die vorhandenen Anwendungen können einen dynamischen Bedarf an Hardwareressourcen haben. Daher wurde in [10] eine dynamische Ressourcenzuteilung für die vorhandenen Anwendungen erläutert.

2.5 Digitale Zwillinge und Verwaltungsschalen

Simulationen haben sich im Laufe der Zeit zu Systemassistenten für den ganzen Lebenszyklus der Anlage entwickelt (Abb. 2.12). Das Konzept eines Digital Twin (DT)s wurde erstmals durch die Technologie-Roadmap der NASA vorgestellt [93]. Das Ziel von DT ist die Abbildung der verschiedenen Aspekte des Automatisierungssystems in die Informationswelt. Ein DT beinhaltet die Simulationsmodelle, Interaktionen und Schnittstellen eines Assets [5], [60]. Das DT-Konzept besteht aus drei Teilen, nämlich der physischen Welt, der virtuellen Welt und einer echtzeitfähigen Kommunikation zwischen diesen. Ein

DT kann für die Datenanalyse, die Optimierung und Fehlerdiagnose verwendet werden [45], [67]. Um die Vorteile eines DT nutzen zu können, muss ein Interaktionsmodell zwischen der Prozessführung und dem DT entwickelt werden. Die Entwicklung der Paradigmen der Herstellungsprozesse, sowie des proaktiven Manufacturing ist in [115], [119] diskutiert. Proaktives Manufacturing entspricht einem neuen Nutzungsgrad der Daten im Fertigungssystem. Die erste Stufe in dieser Entwicklung war die passive Strategie. In dieser Stufe wurden die Daten manuell gesammelt. Für die Verwaltung der Daten wurden die traditionellen Datenbanken eingesetzt. Traditionelle Datenbanken konnten die Anforderungen der kleinen Datenmengen erfüllen. In der zweiten Stufe wurde die Echtzeitdatenerfassung mittels RFID (Radio Frequency Identifikation), Barcodes, Ethernet, drahtloser Netzwerke usw. realisiert. In der nächsten Stufe wurden die Machine-Learning-Applikationen und künstliche Intelligenz (beispielsweise data mining, cloud computing und neuronale Netzwerke) eingesetzt, um das Systemverhalten vorhersagen zu können. Beispielsweise für diese sind predictive maintanence und predictive quality. In der aktuellen Stufe ist das Fertigungssystem in der Lage, anhand zur Verfügung stehender prädikativer Informationen, autonome Entscheidungen zu treffen. In der aktuellen Stufen spielt DT eine sehr wichtige Rolle. Er wird eingesetzt, um eine ausführliche Beschreibung (Verhalten, Eigenschaften, Funktionen usw.) der physischen Entitäten bereitzustellen. In den folgenden Abschnitten

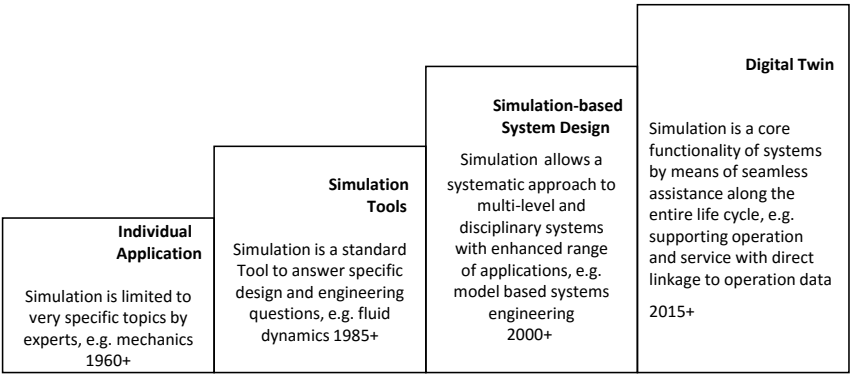

Abbildung 2.12: Entwicklung der Simulation [17]

werden verschiedene Anwendungen eines DT erläutert.

2.5.1 Digitaler Zwilling als Validierungskomponente

Die Validierung der Prozessführung kann in vier verschiedenen Systemkonfigurationen erfolgen:

Reale Anlage und reales Steuerungssystem Dabei handelt es sich um die traditionellen Automatisierungssysteme. Test und Verifikation werden in diesen Systemen während des realen Prozesses durchgeführt.

Hardware-in-the-Loop Bei der Hardware in the Loop-Technologie wird ein reales eingebettetes System über seine I/Os mit einer Nachbildung der realen Umgebung des Systems verbunden. Damit soll eine Plattform für Tests des Automatisierungssystems bereitgestellt werden. Dies ermöglicht das Austesten der entwickelten Software auf der Zielhardware mittels eines Hardware-Emulators [89], [12], [99].

Software-in-the-Loop Im Gegensatz zu Hardware-in-the-Loop wird bei der Software in the Loop-Technologie keine bestimmte Hardware verwendet [87]. Das bedeutet, dass die entwickelte Software nicht auf der Zielhardware, sondern auf einem Entwicklungsrechner getestet wird.

In dieser Arbeit wird, für die Validierung der deployten Komponenten und Optimierungsvorgänge eine Umgebung für Software-in-the-Loop-Tests eingesetzt. Die Software in the loop Applikation läuft auf der gleichen Hardware, wie die Prozessführungsapplikation.

2.5.2 Digitaler Zwilling für Beobachtung und Optimierung

Der Begriff des Beobachters ist aus der Regelungstechnik bekannt. Eine der Beobachtungsmethoden in der Regelungstechnik wird von Luenberger vorgestellt. Der so genannte Luenberger Beobachter besteht aus der Parallelschaltung eines Beobachters und eines Systemmodells. Dabei wird die Differenz der Zustandswerte an das Systemmodell geschickt. In dieser Arbeit zielt der Beobachter darauf ab, die Abweichung zwischen den Prozesswerten und den erwarteten Werten zu erkennen. Der Beobachter läuft parallel zum Prozess und überwacht sein Verhalten. Dies benötigt ein Simulationsmodell das prozessparallel laufen kann. Stehen solche Prozessmodelle zur Verfügung, dann können sie auch für eine prozessbegleitende Optimierung eingesetzt werden.

Verwaltungsschale

Die Verwaltungsschale (standardisiert in [30]) ist eine digitale Darstellung eines Assets während seines Lebenszyklus [109], [18], [77]. Das Ziel der Verwaltungsschale ist es konsistente Informationen über das Asset bereitzustellen. Beispielsweise kann die Verwaltungsschale einer Bohrmaschine u.a. Informationen über Hersteller, Drehgeschwindigkeit, Bohrtiefe enthalten. Die Verwaltungsschale ermöglicht sowohl eine einheitliche Informationsmodellierung als auch eine einheitliche Schnittstelle für den Zugriff auf die Daten. Verwaltungsschale und Asset bilden zusammen eine I40 Komponente. Der Begriff überlagert sich heute in weiten Bereichen mit dem Begriff des DT [110].

2.6 Laufzeitumgebungen

Laufzeitumgebungen sind die Grundlage für die Entwicklung und Ausführung von Modellen und Anwendungen in der Prozessautomatisierung. Eine der Hauptkomponenten der Laufzeitumgebungen in automatisierungstechnischen Anwendungen sind die FBs. Ein anwendungs- und herstellerneutraler Ansatz zur Modellierung und Beschreibung von Laufzeitumgebungen wird in [47] vorgestellt.

Dynamische Laufzeitumgebungen

Der Automatisierungsprozess ist in zwei Phasen unterteilt, nämlich der Entwicklung und dem Engineering. Diese Phasen sind in Abb. 2.13 dargestellt. In der Entwicklungsphase werden neue FB-Typen implementiert [118]. Die Implementierung kann dabei in verschiedenen Programmiersprachen durchgeführt werden. Die Engineering-Phase umfasst die Instanziierung, Konfigurierung und Verbindung von FBs zum Aufbau von Sequential State Charts (SSC)s und CFCs. Dynamische Laufzeitumgebungen bieten die Möglichkeit, die Prozessführungsanwendung auch während der Laufzeit zu manipulieren. Im Folgenden werden einige Technologien für Laufzeitumgebungen besprochen.

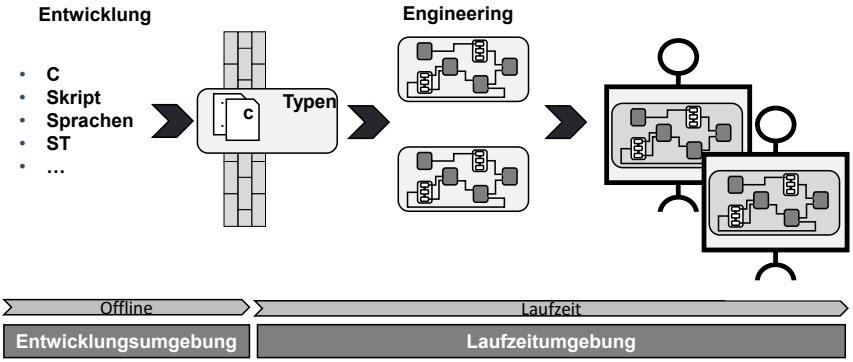

Abbildung 2.13: Dynamische Laufzeitumgebungen [34]

ACPLT/RTE

Die Open-Source-Laufzeitumgebung ACPLT/RTE (Aachener Prozessleittechnik Runtime Environment) wird am Lehrstuhl für Prozessleittechnik der RWTH Aachen entwickelt. Sie wird bereits in diversen Forschungs- und Industrieprojekten eingesetzt. Sie verfügt über ein eigenes Objektmanagementsystem und ein Metamodell, mit dessen Hilfe Objekte innerhalb des Systems erstellt werden können [70]. Die Kommunikation von Objekten und Metadaten wird durch das Kommunikationsprotokoll ACPLT/KS (Kommunikationssystem) oder Open Platform Communications Unified Architecture (OPC UA) realisiert [1], [53]. Das Objektmanagementsystem bietet ein, in ANSI C implementiertes, Object Oriented (OO)-Framework. Bekannte Features der OO-Programmierung wie Vererbung, Aggregation, Klasse werden von ACPLT/OV zur Verfügung gestellt. OV stellt eine Basisklasse von Objekten, die von Benutzern erweitert werden können, bereit. Jede Klasse gehört zu einer Bibliothek und besteht aus Variablen und Operationen. Die während der Entwicklungsphase in ACPLT/OV definierten Klassen können im OV-Laufzeitsystem instanziiert werden. Die instanziierten Klassen können verwendet werden, um gewünschte Anwendungen zu implementieren. Der ausführbare Code enthält die Metainformationen. Dies ermöglicht den Zugriff auf die Klassenschnittstellen während der Laufzeit, so dass eine Manipulation von Objekten während der Laufzeit möglich ist.

ACPLT/KS ACPLT/KS ist das Kommunikationssystem von ACPLT/RTE. Dieses Kommunikationssystem verwendet Transmission Control Protocol (TCP)/IP als Grundlage und bietet zusätzlich Dienste wie getVar (um einen Wert aufzurufen), setVar (um einen Wert zu definieren).

4DIAC FORTE

4Diac (The Framework for Distributed Industrial Automation and Control) ist ein Eclipse-Projekt, das eine Open-Source-Entwicklungs- und Laufzeitumgebung hervorgebracht hat. Diese Laufzeitumgebung ist IEC 61499-kompatibel [95]. Der modulare Aufbau von 4diac ermöglicht das Laden von Add-ons als Bibliotheken. Es unterstützt diverse OSs wie eCos, Cygwin und Linux und bietet eine ereignisbasierte Anwendungsentwicklung. Außerdem bietet 4diac eine erweiterbare Kommunikationsschicht zur Bereitstellung einer flexiblen Kommunikationsinfrastruktur. Es unterstützt verschiedene Kommunikationsprotokolle wie Ethernet, OPC UA und MQTT.

2.6.1 Industrie-PCs und eingebettete Systeme

Die eingebetteten Systeme und IPCs werden zunehmend im industriellen Bereich eingesetzt. Sie bringen viele Vorteile wie Robustheit, niedrige Preise, Effizienz etc. mit sich. Ähnlich wie eine SPS bestehen auch sie jeweils aus einer CPU, Speicher, Kommunikationsmodulen und I/O-Modulen. Es bestehen jedoch einige Unterschiede zwischen ihnen. Die Unterschiede sind unten aufgeführt [79]:

- **Modularität:** SPSen sind modular aufgebaut. Die Module einer SPS können nach Bedarf durch andere Module ersetzt werden.
- **Programmiersprache:** Die Programmiersprachen der SPSen basieren auf FBs. Eingebettete Systeme werden in höheren Programmiersprachen wie C oder C++ programmiert.
- **Safety**: SPSen bieten Kommunikationskanäle zur Überwachung der Vorgänge.
- **Robustheit:** SPSen haben keine beweglichen Komponenten. Dies ermöglicht den dauerhaften Betrieb in der Anlagenumgebung.
- **Operation:** SPSen besitzen ein eingebettetes RTOS. Sie erfüllen die Echtzeitanforderungen der Prozessführung. Sie sind für die Prozessautomatisierung konzipiert und laufen ohne weitere Dienstprogramme oder System-Updates.
- **Kosten:** SPSen sind kostenintensiver als eingebettete Systeme und IPCs.
- **Zertifizierung:** Ein weiterer Faktor betrifft die Zertifizierung. In vielen Projekten wird zertifizierte Hardware benötigt. In diesen Fällen haben SPSen gegenüber eingebetteten Systemen einen Vorteil.

2.6.2 Betriebsmittel und Maßnahmenmodell

Das Betriebsmittel und Maßnahmanmodell wurde in [68] vorgestellt. Dieses Modell beschreibt eine hierarchische Prozesssteuerung, bei der jede Ebene eine bestimmte Funktionalität aufweist. Die Hierarchie besteht aus folgenden Ebenen (Abb. 2.14):

Einzelsteuereinheiten enthalten die Steuerlogik einzelner Aktoren wie Ventile und Pumpen.

Gruppensteuereinheiten werden benutzt, um Einzelsteuereinheiten (ESE)s gemäß Rezepten zu orchestrieren.

Maßnahmen sind Produktionsrezepte.

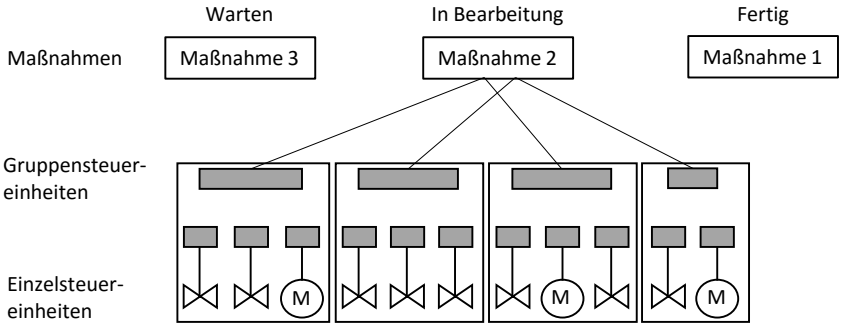

Abbildung 2.14: Steuerungshierarchie

Die Flexibilität in dieser Architektur basiert auf den Maßnahmen, welche dynamisch instanziiert und ausgeführt werden können.

3 Anforderung an zukünftige Automatisierungssysteme

3.1 Anforderungen

In diesem Kapitel werden die Anforderungen der zukünftigen Automatisierungssysteme, die in der vorgeschlagenen Architektur berücksichtigt werden, vorgestellt. Diese Anforderungen zielen darauf ab, die Vernetzung und Agilität der Steuerungssysteme zu erhöhen. Im Zuge der Digitalisierung müssen auch die SPS-Systeme in das I4.0 Produktionsumfeld integriert werden [61]. Die Vernetzung und das Zusammenspiel von IT und OT ändern auch die Anforderungen an SPS-Systeme. Diese Anforderungen können allgemein, wie folgt aufgelistet werden:

- Leistungsfähige Übertragung von Feld- und Automatisierungsdaten an überlagerte Anwendungen
- Prozessbegleitende Optimierung und Überwachung
- Effiziente interne Kommunikation
- Lokale Komponentenverwaltung
- Dynamisches Komponentenmanagement
- Explizite Verwaltung und Sicherstellung von QoS-Eigenschaften

Unabhängig von diesen zusätzlichen Aufgaben müssen die prozessbezogenen Komponenten ihre klassischen Management- und Prozessführungsaufgaben weiterhin zuverlässig und sicher erfüllen. Auch die für sie eingeführten Engineering- und Instandhaltungsmaßnahmen sollen in gewohnter Weise weiter möglich sein. Die aufgelisteten Anforderungen werden in den nächsten Abschnitten ausführlicher erörtert.

3.2 Leistungsfähige Übertragung von Feld- und Automatisierungsdaten an überlagerte Anwendungen

Die Übertragung von Feldinformationen in die Cloud (z. B. eine zentrale Optimierungseinheit) stellt eine Basis für weitere Optimierungen und datenbasierte Entscheidungen in Automatisierungssystemen bereit. Allerdings muss dies mit einer sicheren Methode erfolgen. Das bedeutet, dass durch diese Kommunikation keine Konfiguration- oder Parameteränderungen in der Prozessführung möglich werden darf. Die Informationen dürfen nur in eine Richtung, nämlich von Applikationen höherer Kritikalität (Prozessführung) zu

Applikationen niedrigerer Kritikalität (IT) fließen. Für diese Kommunikation gelten die Anforderungen der NAMUR-Diode. Abb. 3.1 stellt den unidirektionalen Informationsaustausch zwischen der Automatisierungstechnik und einer Cloud auf Basis der NAMUR-Diode dar [69], [11]. Die Realisierung dieses Konzeptes wird im weiteren Verlauf dieser Arbeit erläutert. Diese Kommunikation muss bestimmte Anforderungen erfüllen:

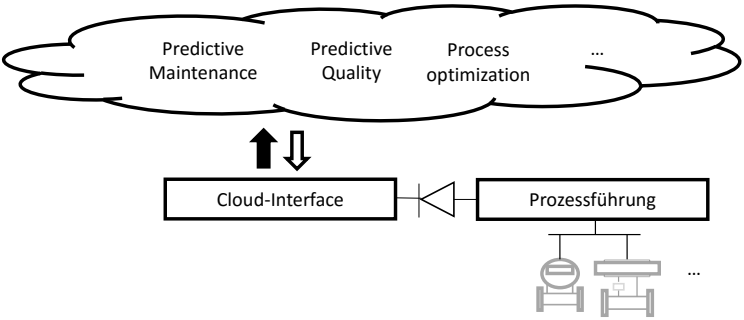

Abbildung 3.1: Übertragen von Feldinformationen

- **Rückwirkungsfrei:** Die Kommunikation darf die Echtzeitfähigkeit, Verfügbarkeit und weitere wichtige Anforderungen des Systems nicht beeinflussen.
- **Bandbreitig:** Die Kommunikation muss die Übertragung einer großen Menge an Daten ermöglichen.
- **QoS-Eigenschaften:** Die Kommunikation muss verschiedene QoS-Eigenschaften, beispielsweise Publish/Subscriber oder eine zyklische Übertragung der Daten, anbieten können.

3.3 Prozessbegleitende Optimierung und Überwachung

Während früher Simulation extern in speziellen Systemen realisiert wurde, geht heute der Trend dahin, Simulationsaufgaben modular in der Prozessumgebung zu realisieren. Die DT-Architektur unterstützt diese Vorgehensweise. Die Entwicklungen auf dem Gebiet der Simulation wurden in Kap. 2 erwähnt. Die neueste Generation von Simulationen wird digitaler Zwilling (DT) genannt. DT kann als eine Basis für Prozessoptimierungen, predictive maintenance und Fehlerdiagnosen verwendet werden. Dafür muss jedoch die Prozessführung mit dem DT interagieren können. Die Interaktion benötigt eine Infrastruktur, die eine sichere und echtzeitfähige Kommunikation mit dem DT erlaubt. Darüber hinaus muss eine modulare Infrastruktur für diverse Simulationsmodelle ermöglicht werden, welche zusammen agieren und Co-Simulationen realisieren. Die Anforderungen für eine Interaktion mit einer prozessparallelen Simulation können wie folgt gelistet werden:

- Eine strikte Trennung zwischen der Prozessführungsapplikation und der parallelen Simulation

- Eine echtzeitfähige Kommunikation zwischen der Prozessführungsapplikation und der parallelen Simulation
- Eine Infrastruktur für die Verwaltung und Ausführung von Simulationsmodellen.

3.4 Effiziente interne Kommunikation

Neue Anwendungen zur Datenanalyse, Optimierung, predictive maintenance erfordern eine breitbandige Anbindung an die Messung und Prozessführung. Mit der KAS-Architektur ergibt sich die Möglichkeit, diese Module – oder zumindest die notwendigen breitbandig anzubindenden Vorverarbeitungsmodule – auf einer Hardware zu konzentrieren. Damit steht die Effizienz der KAS-internen Komponenten-Komponenten-Kommunikation im Fokus. Diese Kommunikation muss einen minimalen Bedarf an Infrastruktur aufweisen, damit die Skalierbarkeit nicht eingeschränkt wird.

3.5 Lokale Komponentenverwaltung

Eine wesentliche Grundlage zur Erhöhung der Flexibilität und Agilität einer Automatisierungslösung ist die Möglichkeit das Komponentensystem in der Betriebsphase zu modifizieren. Dazu wird ein aktives Komponentenverwaltungssystem in der Betriebsphase benötigt. Aus Sicherheits- und aus Standardisierungsgründen (Zielsystemunabhängigkeit) fordert die KAS-Architektur ein lokales Verwaltungssystem, das die Komponentenverwaltungsdienste als Standarddienste anbietet. Darüber hinaus benötigt die Realisierung verschiedener Vorgänge wie das Deployment, die Kommunikation, die Ressourcenverwaltung eine entsprechende Orchestrierung der beteiligten Komponenten. Diese wird ebenfalls vom lokalen Verwaltungssystem durchgeführt. Komponenten haben je nach Anwendungsfall und Rolle unterschiedliche Mächtigkeit. Der Begriff Komponente umfasst in der I40-Terminologie sowohl physische Komponenten, als auch nicht-physische Komponenten wie z. B. Softwarekomponenten. Technische Komponenten sind vordefinierte, in sich geschlossene und individuell handhabbare Einheiten, die eine konkrete Rolle in einem technischen System erfüllen [94]. In der KAS Architektur versteht man unter Komponenten die nicht physischen Komponenten der Komponentenhierarchie:

- Funktionsbaustein und Prozessführung als Komponente
- Container als Komponente
- Partition als Komponente

Funktionsbaustein und Prozessführung als eine Komponente

Funktionsbausteine wurden in IEC 61131-3 und IEC 61499 standardisiert. Sie bestehen aus Eingängen, Ausgängen und einer internen Funktion (Logik), die das Verhalten des FB bestimmt. Sie können miteinander verbunden werden, um komplexe Funktionsblockdiagramme zu erstellen. In klassischen Leitsystemen sind die FBs gekapselt und können als Komponenten gehandhabt werden.

Im letzten Kapitel wurde eine komponentenbasierte Hierarchie für die Prozessführung vorgestellt. Diese Steuerungshierarchie besteht aus ESEs und Gruppensteuereinheiten (GSE)s, die wiederum als Prozessführungskomponenten betrachtet werden. Diese Komponenten bieten unterschiedliche Fähigkeiten und nehmen unterschiedliche Rollen ein. Sie können entsprechend den erforderlichen Rollen und Fähigkeiten erstellt und miteinander verknüpft werden. Wenn die Rolle in der Prozessführung nicht mehr erforderlich ist, wird die Komponente gelöscht. Abb. 3.2 zeigt die interne Architektur einer Prozessführungskomponente. Die interne Architektur enthält unterschiedliche Fahrweisen und Fähigkeiten sowie vier verschiedene Zustände.

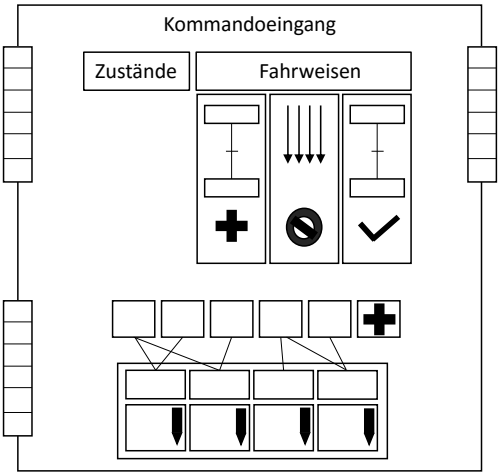

Abbildung 3.2: Prozessführungskomponente [46]

Container als eine Komponente

Containertechnologien wurden im Abschnitt 2 vorgestellt. Die Container müssen ebenfalls verwaltet werden, um das Produktionsziel zu erfüllen. Es existieren bereits einige Softwares zur Verwaltung von Containern, wie z.B. Kubernetes [21].

Partition als eine Komponente

Neben Docker-Containern müssen auch Hypervisor-Partitionen verwaltet werden. Diese werden ebenfalls als Komponenten betrachtet. Die Hypervisor Partitionen hosten die Container und FBs und bilden auf dieser Weise die unterste Ebene der Komponenten.

Zusammenfassung

Das Ziel ist ein Verwaltungssystem zu konzipieren, das diese Komponente verwaltet und orchestriert. Die Anforderungen an dieses Verwaltungssystem können wie folgt aufgelistet werden. Das Verwaltungssystem muss:

- lokale Komponentenverwaltungsdienste (create, read, update und delete),
- Dienste für die Verwaltung der Kommunikation zwischen Komponenten,
- Dienste für das Deployment neuer Komponenten und
- Dienste für die Ressourcenverwaltung anbieten.

3.6 Dynamisches Komponentenmanagement

Das dynamische Deployment zur Laufzeit erhöht die Fähigkeit des Systems, dynamisch auf Veränderungen und Anforderungen zu reagieren. Es ist dabei wichtig zwischen den folgenden Begriffen zu unterscheiden:

- **Deployment:** Als Deployment wird der Vorgang der Zuweisung einer Software-Einheit an einen Rechenknoten bezeichnet.
- **Redeployment:** Das Redeployment ist die Verlagerung einer Software von einem Rechenknoten auf einen anderen.

Die Ausstattung der Prozessführungsapplikation mit einem dynamischen Deploymentssystem erhöht die Agilität und Anpassbarkeit des Systems. In der KAS-Architektur können sowohl Container als auch FBs dynamisch deployt werden. Die Partitionen werden in der Engineeringphase statisch angelegt.

Für den Deploymentvorgang müssen die folgenden Vorbedingungen erfüllt sein:

- **ID**: Die Komponenten müssen eindeutig identifizierbar sein.
- **Erforderliche Ressourcen**: Die Ressourcen zur Ausführung der Komponente müssen auf der Hardware vorhanden und verfügbar sein.
- **Validierung**: Die deployten Komponenten müssen vor der Integration in die operative Ausführung validiert werden.

3.7 Explizite Verwaltung und Sicherstellung von QoS-Eigenschaften

Die Komponenten haben gemäß ihrer Aufgaben verschiedene Anforderungen an ihren QoS. Dazu zählen z.B. Anforderung an Verfügbarkeit, Echtzeit, Integrität. Beispielsweise haben die Komponenten, die in der Prozessführung (ähnlich wie CPC in der NAMUR-Architektur) eingesetzt sind, eine sehr hohe Anforderung an Verfügbarkeit. Bei einer Optimierungsapplikation hingegen ist dies nicht der Fall. Die Partitionen besitzen gemäß der Anwendungen, welche sie kapseln, unterschiedliche Fähigkeiten, Eigenschaften und Zugriffsrechte. Diese können in die folgenden Kategorien unterteilt werden:

- **Integrität:** Die Anwendungen können unterschiedliche Safety oder Securityniveaus haben. Im Safetybereich können die Anforderungen z. B. unterschiedlichen SIL-Ebenen entsprechen.

- **Echtzeitfähigkeit:** Die Anwendungen, die eine Anforderung an Echtzeit haben, können in einer Partition gekapselt werden, die mit einem echtzeitfähigen Betriebssystem ausgestattet ist. Echtzeitfähigkeit kann sowohl für die Ausführung der Prozesse als auch für die Kommunikation zwischen den Komponenten definiert werden. Dafür können die angeforderten Jitter-Bereiche für die Ausführung der Prozesse oder Deadlines für die Übertragung der Daten definiert werden.

- **Zugriffsrechte:** Die Partitionen haben gemäß ihrer Anwendungen unterschiedliche Zugriffsrechte und Treiber. Die Zugriffsrechte beinhalten u.a. den Zugriff auf:
  - **I/O-Geräte (Bus-System):** Nur die Partitionen, die ein Zugriffsrecht auf I/O-Geräte haben, können auf diese zugreifen.
  - **Scheduling-Tabelle und Memory Management Unit (MMU):** Die Partitionen, die einen Zugriff auf Scheduling-Tabelle und MMU haben, können das Schedulingsschema der Hardware-Ressourcen ändern.
  - **Kommunikationsverwaltungschnittstelle:** Die Partitionen, die einen Zugriff auf die Kommunikationsverwaltungschnittstelle haben, können die Kommunikationsverbindungen zwischen vorhandenen Komponenten verwalten.
  - **Kommponentenverwaltungsschnittstelle:** Die Partitionen, die einen Zugriff auf die Kommponentenverwaltungschnittstelle haben, können die vorhandenen Komponenten verwalten (beispielsweise, aktivieren und deaktivieren).

  Dabei können auch weitere Zugriffsrechte, wie der Zugriff auf GPU, zusätzlichen Speicher, Speicherdirektzugriff, Cloud-Schnittstelle im Betracht gezogen werden.

- **Fähigkeiten:** Die Partitionen können unterschiedliche Fähigkeiten besitzen. Die Fähigkeiten umfassen Read/Write-Rechte auf andere Partitionen, das Neustarten, die Aktualisierung und die Ausschaltung anderer Partitionen.

Alle Komponenten müssen den für sie relevanten QoS-Eigenschaften als Standardattribut zugeordnet sein. Die Partitionen müssen zeigen, welche QoS-Eigenschaften sie unterstützen können.

4 Konzept

4.1 Allgemeine Architektur

In diesem Kapitel wird eine Architektur vorgestellt, die als komponentenbasierte Architektur für Automatisierungssysteme (KAS) bezeichnet wird. In der KAS-Architektur werden die Anwendungen in Komponenten gemäß ihrer Anforderungen, Quality of Service (QoS)-Eigenschaften und Abhängigkeiten gekapselt. Die Architektur erlaubt eine strikte Trennung der Komponenten in verschiedenen virtuellen Umgebungen, so dass das Aktualisieren, Zurücksetzen oder auch Modifizieren einer Anwendung den Betrieb anderer Anwendungen in anderen Komponenten nicht beeinflussen kann. Die KAS-Architektur sieht eine Kapselung durch Komponenten auf drei unterschiedlichen Ebenen vor:

- der Kapselung in Partitionen
- der Kapselung in Containern
- der Kapselung in FBs

4.2 Komponentenhierarchie

Abb. 4.1 stellt die KAS Komponentenhierarchie formal dar. Die unterste und für das KAS-System wesentliche Kapselung ist die Kapselung in Partitionen. Eine Partitionen kann eine Anwendung beinhalten oder diverse Container (z. B. Docker Container), die jeweils wieder eine Anwendung oder ein FB-System kapseln. Die Kapselung in Containern bildet die nächste Komponentenebene. Die Container können eine Anwendung beinhalten oder diverse FBen, welche zusammen die Anwendung bilden. Die FBen bilden die nächste Komponentenebene. Alle drei Varianten können beliebig gemischt auf einer Hardwareplattform betrieben werden. Abb. 4.2 stellt diese Aufbaumöglichkeiten dar.

Die KAS-Architektur besteht grundsätzlich aus einem Verwaltungspartition, einer Interface-Partition und diversen Anwendungspartitionen. Die Verwaltungspartition beinhaltet ein Verwaltungssystem, das die Komponenten (Funktionsbausteine (FB), Prozessführungskomponenten (PF), Container und Partitionen) gemäß den Anforderungen orchestriert. Die Interface-Partition ist die einzige Partition, die mit überlagerten Anwendungen kommunizieren darf. Die KAS-Architektur kann eine beliebige Anzahl an Anwendungen beinhalten. Diese Anwendungen können, abhängig von der industriellen Domäne, unterschiedliche Funktionalitäten anbieten. Das Verwaltungssystem und die Interface-Partition werden in den nächsten Abschnitten detaillierter erläutert.

Der Hypervisor stellt eine Abstraktionsschicht zwischen der Hardware und den Partitionen bereit. Auf diese Weise reduziert er die Abhängigkeit zwischen Software und Hardware.

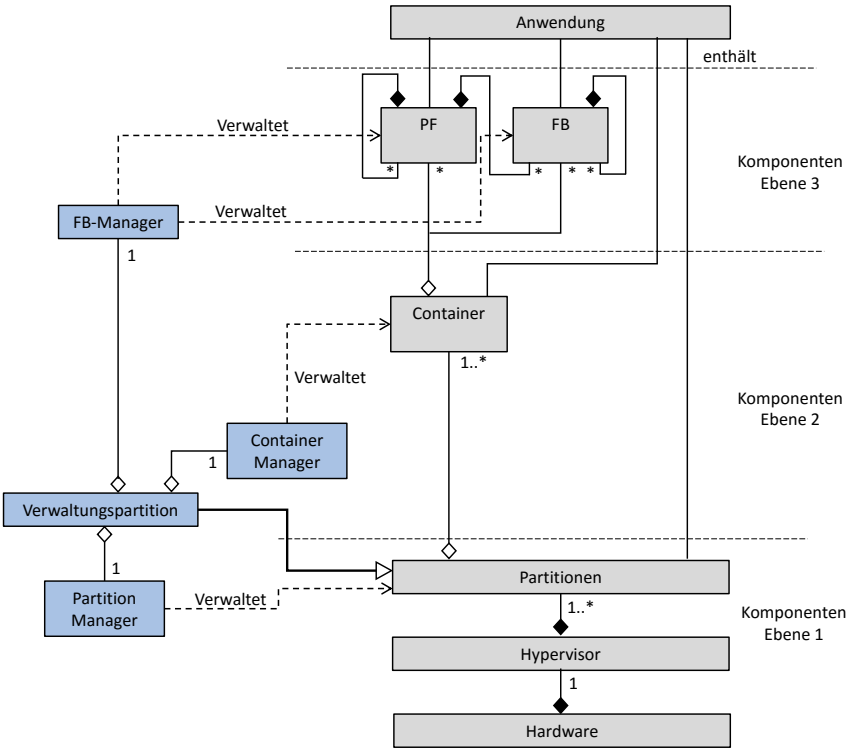

Abbildung 4.1: Metamodell der Komponentenhierarchie

Partitionen stellen den übergeordneten Komponentensystemen und Anwendungen eine virtuelle Umgebung (VM) zur Verfügung. Diese ist gekennzeichnet durch ihre QoS und ihre Abhängigkeiten:

- **QoS**: wie beispielsweise Verfügbarkeit, Echtzeitfähigkeit und Sicherheitsanforderungen
- **Abhängigkeiten**: wie beispielsweise Betriebssystem, Bibliotheken, Treiber und Zugriffsrechte auf I/Os.

4.2.1 Kommunikation zwischen den Partitionen

Das KAS-System sieht eine strenge Regulierung und Überwachung der Kommunikationskanäle zwischen den Partitionen vor. Prinzipiell stehen folgende Kommunikationsformen zur Verfügung:

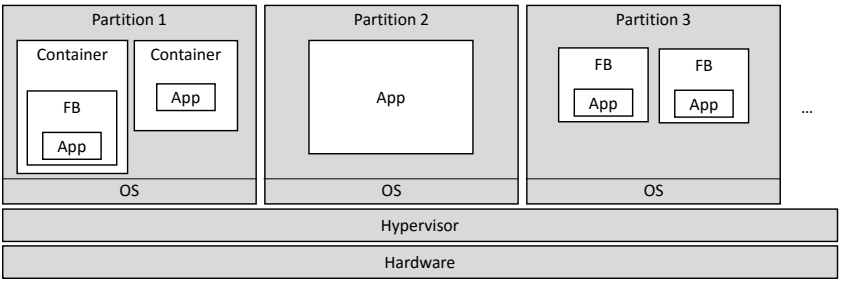

Abbildung 4.2: Interner Aufbau der Partitionen

Kommunikationsform 1: Unidirektionale echtzeitfähige Übertragung von Telegrammen von einem Senderport an einen Empfängerport. Diese Kommunikationsart ist eine synchrone Kommunikation, die aus folgenden Komponenten besteht:

- Senderports: Die Senderports können nur für das Senden der Daten verwendet werden.
- Empfängerports: Die Empfängerports werden eingesetzt, um Daten zu empfangen.
- Verbindungen: Die Verbindungen realisieren Kommunikationskanäle zwischen Sender- und Empfängerports für den Datenaustausch.

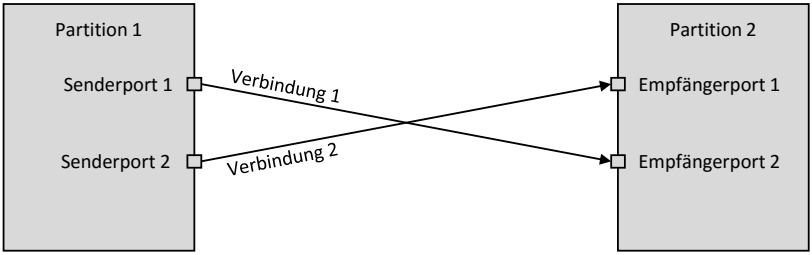

Abbildung 4.3: Unidirektionale Kommunikation

Abb. 4.3 stellt zwei Partitionen, die mittels Ports kommunizieren dar. In dieser Darstellung hat die Partition 1 zwei Senderports und die Partitionen 2 zwei Empfängerports. Diese sind durch Verbindung 1 und 2 miteinander verbunden.

Kommunikationsform 2: Die Komponenten schreiben ihre Nachrichten und Anfragen in einen geteilten Speicherbereich, auf welchen die anderen Partitionen Lese-Recht haben und diese Nachrichten abholen können. Diese Kommunikationsart ist in Abb. 4.4 dargestellt.

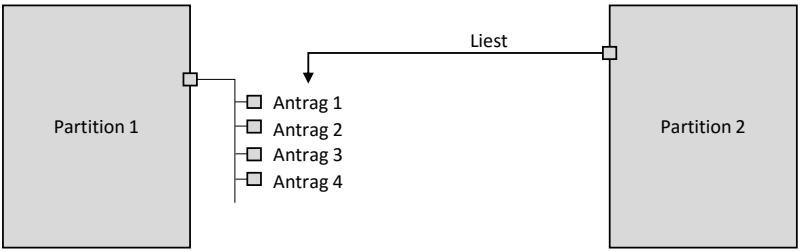

Abbildung 4.4: Kommunikationsform 2

Partition 1 schreibt ihre Anfragen in ihren Kommunikationsport. Partition 2 liest und bearbeitet die Anfragen der Reihenfolge nach. Diese Kommunikationsart ist eine asynchrone Kommunikation. Eine Synchronisierung der Komponenten ist nicht erforderlich. Diese Kommunikationsart ist besonders in einem Fall interessant, in dem eine Komponente niedrigerer Kritikalität mit einer Komponente höherer Kritikalität kommunizieren soll. Aufbau der Verbindungsarchitektur:

- Die statische Phase (Engineering-Phase): In dieser Phase wird die Anzahl der Partitionen festgelegt. Darüber hinaus werden die Kommunikationsports und die Verbindungen generiert. Diese können zur Laufzeit nicht mehr gelöscht oder erzeugt werden. Die Rechte und Fähigkeiten der Partitionen (Read/Write auf andere Partitionen oder Zugriffsrechte auf externe Geräte (z. B. I/Os)) werden ebenfalls in dieser Phase zugeordnet. Die zugewiesenen Rechte und Fähigkeiten sind prinzipiell vorhanden, aber nicht zwangsläufig alle aktiviert.
- Die dynamische Phase: In der dynamischen Phase können beispielsweise Verbindungen aktiviert oder deaktiviert werden.

4.3 Systempartitionen

In der KAS-Architektur wird ein Verwaltungssystem für die Verwaltung sämtlicher Komponenten und Kommunikationskanäle eingesetzt. Dieses Verwaltungssystem bietet Verwaltungsdienste an. Das Verwaltungssystem darf durch eine sichere Schnittstelle mit einem Planungssystem kommunizieren. Ziel dieser Kommunikation ist die Orchestrierung der Verwaltungsdienste [9]. Die Dienste und die Fähigkeiten, die durch das Verwaltungssystem zur Verfügung gestellt werden, werden im nächsten Abschnitt erläutert. Die Kommunikation kann entweder direkt zwischen den anderen Komponenten aufgebaut werden oder indirekt über das Verwaltungssystem abgewickelt werden. Im zweiten Fall, werden die Informationen zunächst an das Verwaltungssystem gesendet. Daraufhin leitet das Verwaltungssystem die Daten an den Empfänger weiter. Die andere Systempartition der KAS-Architektur ist das Interface. Die Interface-Partition ist die einzige Partition, die nach außen (den ungeschützten Bereich) kommunizieren darf. Die Partition stellt eine Möglichkeit für den Datenaustausch mit überlagerten Anwendungen bereit. Die KAS-Architektur beinhaltet auch

eine beliebige Anzahl an Anwendungspartitionen. Diese können unterschiedliche Aufgaben und Anforderungen haben. KAS stellt eine Infrastruktur für den Betrieb dieser Anwendungen, sowie deren Kommunikation mit anderen Komponenten bereit.

4.3.1 Verwaltungssystem

Das Verwaltungssystem überwacht und verwaltet die Operationen und die Komponenten in der KAS-Architektur. Sie bietet Verwaltungsdienste an, welche für die Komponentenverwaltung eingesetzt werden. Die Verwaltungsdienste können auch von Klienten (überlagerte Anwendungen) aufgerufen werden, um bestimmte Funktionalitäten auf dem System auszuführen. Die Kommunikation zwischen dem Verwaltungssystem und dem Planungssystem erfolgt über eine sichere Schnittstelle. Abb. 4.5 präsentiert das Verwaltungssystem. Die Verwaltungsdienste werden in diesem Abschnitt detaillierter erläutert.

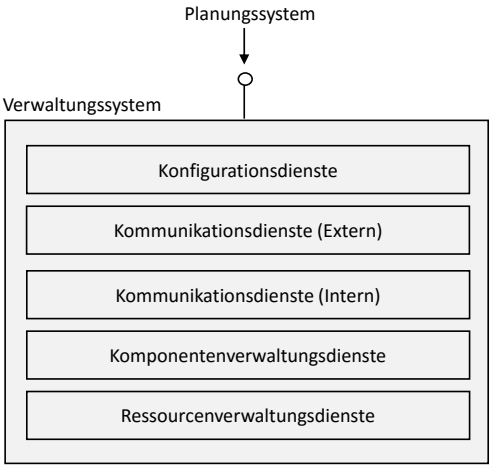

Abbildung 4.5: Interne Struktur des Verwaltungssystems

Das Verwaltungssystem umfasst folgende Konzepte:

- Kommunikationsdienste (Intern): werden zur Verwaltung der Kommunikationsverbindungen zwischen den Komponenten eingesetzt
- Kommunikationsdienste (Extern): verwalten die Kommunikation mit externen Komponenten. Das Verwaltungssystem agiert wie ein Gateway zwischen hoch kritischen Applikationen und Applikationen geringerer Kritikalität (NAMUR-Diode)
- Konfigurationsdienste: werden zur Validierung und Integration der deployten Komponenten eingesetzt
- Komponentenverwaltungsdienste: befassen sich mit Dienstleistungen für die Komponentenverwaltung. Es handelt sich dabei um Dienste wie Create, Delete, Copy und Update

- Ressourcenverwaltungsdienste: teilen die Hardware-Ressourcen einzelnen Anwendungen/Partitionen zu

Das Verwaltungssystem hat eine Eingangsschnittstelle für den Empfang von Kommandos. Wie zuvor beschrieben können diese Kommandos beispielsweise von einem Planungssystem stammen. Die Semantik der Kommandos lautet wie folgt:

[Command1];[PARAMETER1=VALUE1,PARAMETER2=VALUE2,. . .]

Tabelle 4.1 präsentiert einige Beispiele dieser Dienste.

Tabelle 4.1: Dienste der Komponentenverwaltung

Command	Parameter1	Parameter2
deploy	component name	docker container
upload	component name	docker container
create communication link	input port	output port
delete communication link	input port	output port
update resource allocation	aktivate scheduling schema X	
aktivate container	container X	
deaktivate container	container Y	

Das Verwaltungssystem hat das Recht auf die Interface-Partition sowie andere Partitionen zuzugreifen und Veränderungen vorzunehmen. Außerdem hat es das Zugriffsrecht zu den Hardware-Ressourcenverwaltungsdienste, um die Ressourcen den Komponenten optimal zuzuteilen.

4.3.2 Interface

Die interne Struktur der Interface-Partition ist in Abb. 4.6 dargestellt. Die Interface-Partition ist die einzige Partition, die mit externen Komponenten (z. B. Cloud) kommunizieren darf. Die Kommunikation aller lokalen Partitionen mit externen Komponenten erfolgt über das Interface. Darüber hinaus hostet die Interface-Partition die lokal geladenen Komponenten (Prozessführungskomponenten, Funktionsbausteine, Anwendungen) bevor diese schließlich in die anderen Partitionen integriert werden. Das Übersenden der Prozessdaten zu den externen Komponenten erfolgt durch das Interface. Die Interface Partition besitzt eine Schnittstelle, um Befehle zum Deployment neuer Komponenten zu erhalten. Vor der Kommunikation mit externen Komponenten authentifiziert die Interface-Partition den Kommunikationspartner. Danach wird die Komponente deployt. Die Komponente ist dann in der Liste der geladenen Komponenten der Interface-Partition vorhanden. Das Interface beinhaltet außerdem eine Schnittstelle für die Datenübertragung. Daten müssen durch die Interface-Partition zu den externen Komponenten gesendet werden. Die Übertragung kann einmalig oder zyklisch erfolgen. Die zyklische Übertragung wird über das Pub/Sub-Schema durchgeführt.

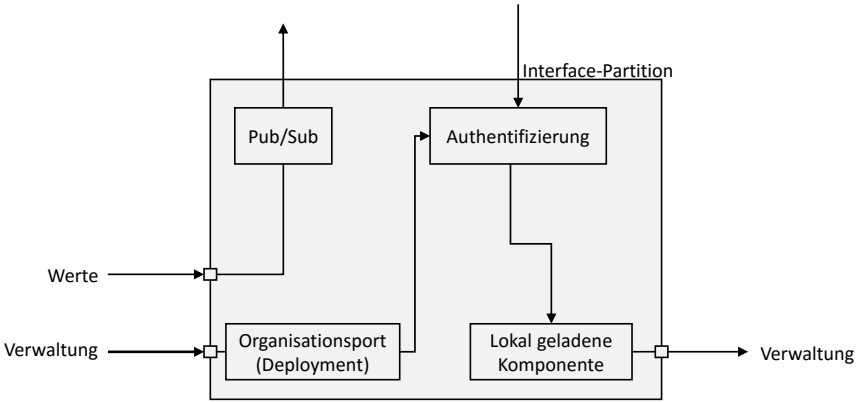

Abbildung 4.6: Interne Struktur der Interface-Partition

4.4 Verwaltungsdienste

Zur Realisierung der Konzepte stellt die Verwaltungskomponente entsprechende Dienste und Ablaufprozeduren zur Verfügung.

4.4.1 Interne Kommunikationsdienste

Die Kommunikation zwischen den Komponenten wird durch das Verwaltungssystem gesteuert. Falls keine direkte Kommunikation zwischen den Komponenten erlaubt ist, agiert das Verwaltungssystem als Gateway und leitet die Daten an den Empfänger weiter. Dazu verfügt es über verschiedene Kommunikationsprotokolle mit unterschiedlichen QoS-Eigenschaften. Bei der Weiterleitung der Daten können diese auch vom Verwaltungssystem modifiziert oder gefiltert werden. Wenn eine direkte Kommunikation erlaubt ist, baut das Verwaltungssystem eine direkte Kommunikationsverbindung zwischen den Komponenten auf. Diese werden zur Laufzeit den Komponenten, die eine Kommunikationsverbindung benötigen, zugeordnet. Aktuelle Kommunikationsverbindungen und Kommunikationsports werden vom Hypervisor verwaltet. Der Hypervisor besitzt eine Liste der zur Verfügung stehenden Kommunikationsports. Das Verwaltungssystem ist mit einer Schnittstelle zu dieser Liste ausgestattet und kann die Kommunikationsverbindungen aktivieren oder deaktivieren (Abb. 4.7).

Quality of Services

Die QoSs, die für die Kommunikation zwischen Komponenten in KAS eingesetzt werden können, sind:

- **History:** Diese QoS wird eingesetzt, um frühere Daten zu erhalten
- **Realtime:** Diese QoS wird für eine Realtime-Kommunikation zwischen den Komponenten eingesetzt

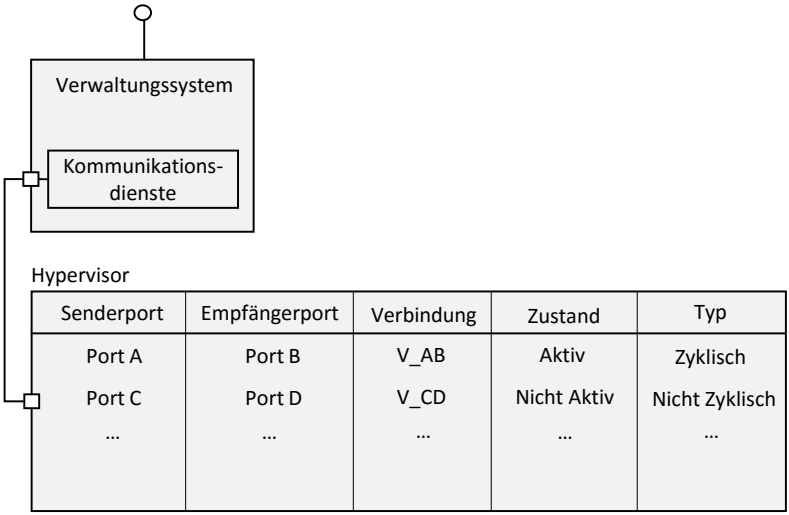

Abbildung 4.7: Verwaltung der Kommunikationsverbindungen

- **Filter:** Diese QoS wird für die Steuerung des Datenflusses eingesetzt.

Die Dienste, die für die Steuerung der Kommunikation eingesetzt werden, sind:

- **Kommunikationsport freischalten:**
  Dieser Dienst ermöglicht den Zugriff von Komponenten zu einem Port für die Kommunikation.
- **Kommunikationsport sperren:**
  Dieser Dienst sperrt den Zugriff einer Komponente zu einem Port.

Durch diese Dienste wird die Kommunikation zwischen verschieden Komponenten gesteuert.

4.4.2 Externe Kommunikationsdienste

Die Datenverarbeitung kann lokal oder zentral (durch eine überlagerte Anwendung z. B. Cloud) durchgeführt werden. Bei der zentralen Verarbeitung der Daten fordert zunächst die Cloud Informationen aus der Anwendung (z. B. einer hoch kritischen Partition) an. Aus Sicherheitsgründen läuft diese Anfrage durch die Interface-Partition. Diese schreibt die Liste der angeforderten Informationen in ihre Ports, wodurch dem Verwaltungssystem der Zugriff auf diese Liste ermöglicht wird. Das Verwaltungssystem liest daraufhin die Anforderungen und stellt die Daten der Interface-Partition zur Verfügung. Sollte es über die angeforderten Informationen nicht verfügen, fordert es diese von der entsprechenden Partition (Beispielsweise Anwendung 1 in Abb. 4.8) an. Die Anwendung sendet daraufhin die

angeforderten Informationen über eine unidirektionale Kommunikationsverbindung zum Verwaltungssystem.

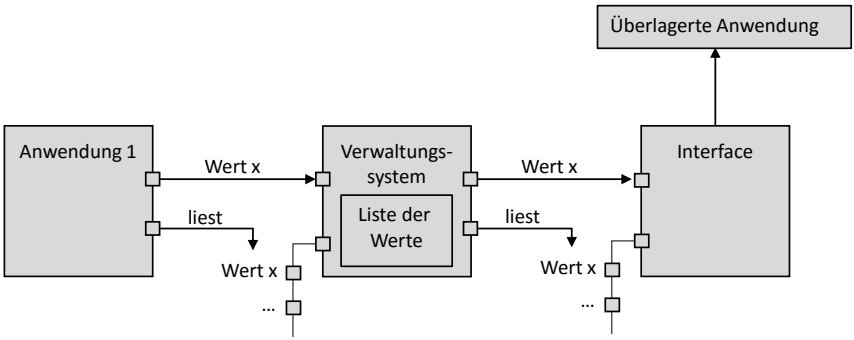

Abbildung 4.8: Verwaltung der Kommunikationsverbindungen

4.4.3 Konfigurationsdienste

Die Konfigurationsdienste beschäftigen sich mit dem Deployment und der Inbetriebnahme neuer Komponenten. Die neuen Komponenten werden deployt, um sich ändernden Anforderungen des Systems gerecht zu werden. Konfigurationsdienste bestehen aus zwei Schritten, nämlich dem Deployment und der Inbetriebnahme. Beim Vorgang des Deployments werden erforderliche Komponenten heruntergeladen. Dabei umfasst die Inbetriebnahme die Überprüfung und Synchronisation der deployten Komponenten. Das Verwaltungssystem ist in der Lage die Komponenten in jeder Partition zu deployen. Aus Sicherheitsgründen ist aber ein direktes Deployment der Komponenten in den Partitionen nicht erlaubt. Das Deployment läuft über das Interface. Das Verwaltungssystem triggert den Deploymentvorgang einer Komponente in das Interface. Folgende Komponenten können auf diese Weise deployt werden:

- FBs
- Prozessführungskomponenten
- Container

Die Konfigurationsdienste werden zur Integration der deployten Komponenten eingesetzt. Diese überprüfen deployte Komponenten vor der Integration ins System auf Plausibilität. Der Konfigurationsprozess besteht aus den folgenden Schritten:

- Die Komponente wird deployt. Dieser Vorgang wird vom Verwaltungssystem initiiert. Die Komponente wird aus der Cloud in die Interface-Partition geladen.
- Die Komponente wird auf Plausibilität getestet.
- Die Komponente wird in die Ressourcen der Zielpartition integriert.

- Die Komponente wird aktiviert und synchronisiert.

Das Diagramm des behandelten Vorgangs ist in Abb. 4.9 dargestellt. Das Verwaltungssystem triggert den Deploymentvorgang der Komponente X aus der Quelle Y. Die Komponente wird in der Interface-Partition deployt und in der Liste der lokal geladenen Komponenten abgelegt. Die geladene Komponente wird in einer Testplattform redeployt, um überprüft zu werden. Die Ergebnisse der Überprüfung werden daraufhin vom Verwaltungssystem analysiert. Schließlich wird die Komponente in die Zielpartition integriert.

Das Ressourcenmodell dieses Vorgangs ist in Abb. 4.10 dargestellt. Außer Deploymentdienste bietet die Verwaltungskomponente auch Redeploymentdienste zur Übertragung von Komponenten aus einer Partition in eine andere Partition an.

4.4.4 Ressourcenverwaltung

Die Verwaltungskomponente analysiert den aktuellen Ressourcenbedarf der Partitionen und ändert das Scheduling-Schema, um die Anforderungen dieser zu erfüllen. Dabei muss die jeweilige Priorität der Applikationen betrachtet werden. Abb. 4.12 präsentiert das Klassendiagramm der Ressourcenverwaltung für eine Anwendung auf Partitionsebene. Das Verwaltungssystem bietet Dienste zur Kommunikation, Speicherverwaltung und Aktivierung neuer Scheduling-Schemata an.

Ablaufprozedur und zur Verfügung stehende Dienste

Anwendungen, die mehr Ressourcen für die Ausführung benötigen, als ihnen zugewiesen wurde, müssen diese zur Laufzeit beim Verwaltungssystem beantragen. Abb. 4.11 präsentiert diese Prozedur für zwei Partitionen. Die Partitionen schreiben ihre Anfrage an längere Zeitpartitionierung (tp) in ihre Kommunikationsports. Das Verwaltungssystem liest die Anfragen und ordnet ihnen die Ressourcen zu. Die Partitionen können zur Laufzeit mehr Rechenzeit, mehr (Arbeits-)Speicher und Kommunikationsverbindungen anfordern:

- **Request(tp):**
  Durch diese Anfrage kann eine neue Zeitpartitionierungsdauer gefordert werden.
- **Request(Arbeitsspeicher):**
  Durch diese Anfrage kann ein größerer Anteil des Arbeitsspeichers gefordert werden.
- **Request(Kommunikationsverbindung):**
  Durch diese Anfrage kann eine Kommunikationsverbindung zu anderen Komponenten gefordert werden.

Die Ressourcen werden nur dann zugeteilt, wenn sie die Funktionsweise der Anwendungen höherer Kritikalität oder Priorität nicht beeinträchtigen und vom Verwaltungssystem genehmigt wurden. Folgende Dienste stehen zur Verfügung:

- **Scheduling-Schema generieren:**
  Dieser Dienst generiert ein neues Scheduling-Schema entsprechend der neuen Anforderungen.
- **Scheduling-Schema aktivieren:**
  Dieser Dienst aktiviert ein neues Schedulingschema

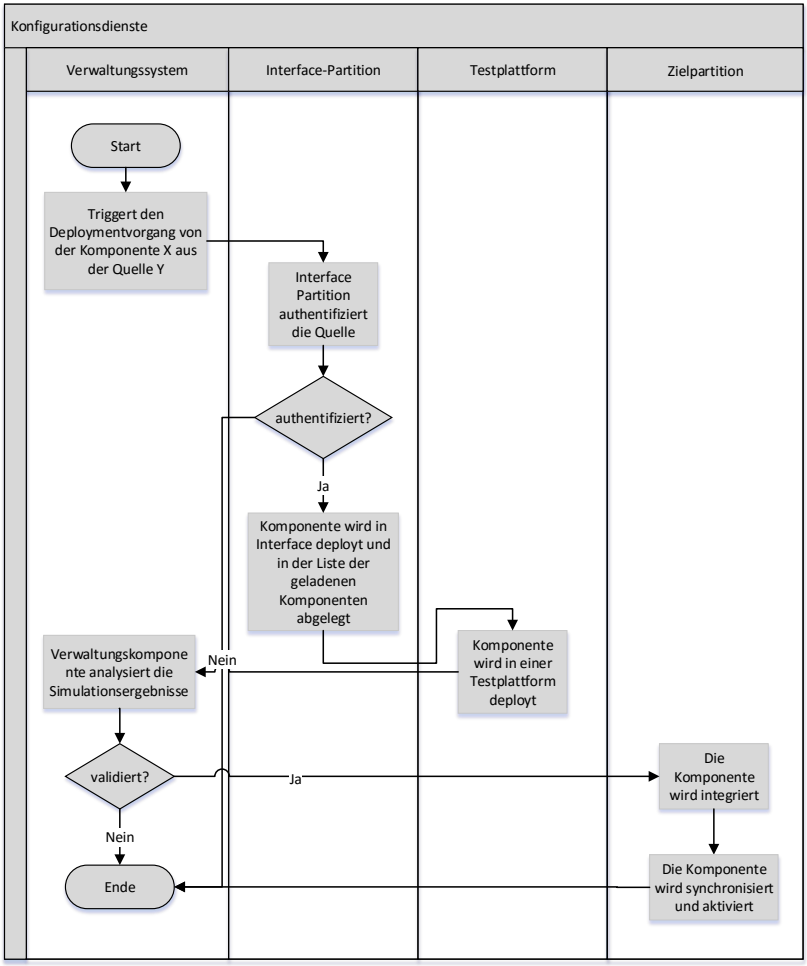

Abbildung 4.9: Deployment und Inbetriebnahme einer FB-Komponente

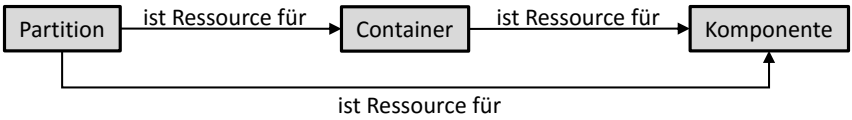

Abbildung 4.10: Ressourcenmodell

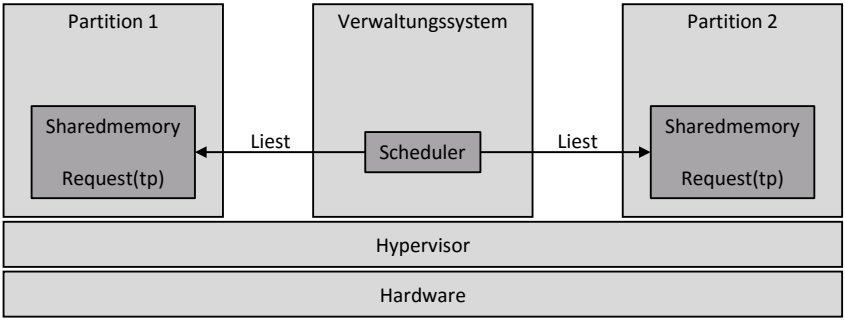

Abbildung 4.11: Ressourcenverwaltung

- **Speicherzuordnung aktualisieren:**
  Dieser Dienst aktualisiert die Speicherzuordnung entsprechend der neuen Anforderungen.

- **Kommunikationsverbindungen:**
  Diese sind bereits in Kap. 4.4.1 erläutert.

Die Ressourcenverwaltung wird vom Hypervisor verwaltet. Das Verwaltungssystem besitzt eine Schnittstelle zu den Schedulingschemata und kann gemäß der Anforderungen diese anpassen.

4.4.5 Komponentenverwaltungsdienste

Die Komponentenverwaltungsdienste bieten grundlegende Funktionen, um die Komponenten anzulegen und zu verwalten. Die zur Verfügung stehenden Dienste für die Komponentenverwaltung können wie folgt aufgelistet werden:

- **create:**
  Der create-Dienst wird zur Erstellung von Komponenten eingesetzt. FBs, Prozessführungskomponenten und Container sind die Komponenten, die dadurch erstellt werden können.

- **read:**
  Der read-Dienst wird für das Lesen der Zustände, Werte und Informationen der Komponenten eingesetzt.

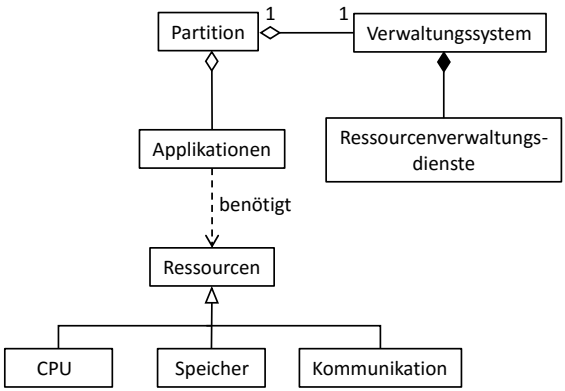

Abbildung 4.12: Klassendiagramm der Ressourcenverwaltung

- **delete:**
  Der delete-Dienst wird für das Löschen der FBs, Prozessführungskomponenten und Container eingesetzt. Die Partitionen können hingegen nach der Erstellung (in der statischen Phase) nicht mehr in der dynamischen Phase gelöscht werden.

- **update:**
  Der update-Dienst kann für die Aktualisierung der Komponenten eingesetzt werden.

- **reset:**
  Der reset-Dienst wird für den Neustart der Komponenten eingesetzt.

- **copy:**
  Der copy-Dienst wird zum Kopieren der Komponenten eingesetzt. Er kann für alle Komponenten mit Ausnahme der Partitionen eingesetzt werden.

- **aktivate:**
  Der aktivate-Dienst wird zur Aktivierung der Komponenten eingesetzt.

- **deaktivate:**
  Der deaktivate-Dienst wird zur Deaktivierung der Komponenten eingesetzt.

Die Komponenten registrieren sich beim Registrysystem des Hypervisors. Die Verwaltungskomponente hat Zugriff auf das Registrysystem und kann die Komponenten mithilfe von Diensten verwalten. Diese Dienste können für die Verwaltung von FBs, Prozessführungskomponenten und Containern eingesetzt werden. Allerdings können die drei Dienste create, read und copy nicht für die Verwaltung der Partitionen eingesetzt werden, da die Anzahl der Partitionen in der Engineering-Phase festgelegt wird.

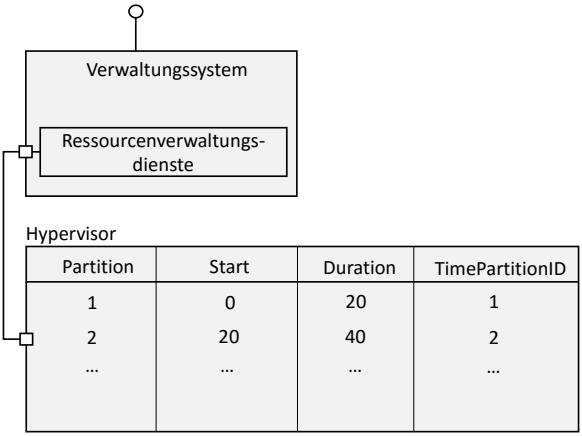

Abbildung 4.13: Verwaltung der Ressourcenverwaltung

4.5 Anwendungspartitionen

In diesem Kapitel wird auf Grundlage der KAS-Architektur eine Systemarchitektur für virtualisierte Steuerungsgeräte entworfen, um eine Plattform zur Erfüllung der in Kapitel 3 genannten Anforderungen bereitzustellen. Der Kern der Architektur besteht, wie in Abb. 4.15 dargestellt, aus unterschiedlichen Partitionen. Deren Trennung wird durch einen Hypervisor durchgeführt. Darüber hinaus überwacht der Hypervisor den Zugriff verschiedener Anwendungen aus den Partitionen auf Geräte, I/Os, andere Partitionen usw. sowie die Kommunikation zwischen den verschiedenen Partitionen. In Bild 4.15 ist beispielhaft ein System mit zwei Anwendungspartitionen dargestellt. Es handelt sich um die Anwendungsarten Control und Optimization and Management (O&M). Die Art der Anwendung einer Partition bestimmt ihre Zugriffsrechte (Zugriff auf andere Partitionen, Geräte und I/Os), Kommunikationskanäle, den internen Aufbau, QoS usw.

Die Anwendungen müssen, trotz unterschiedlicher Kritikalität, miteinander kommunizieren können. Hierfür wird ein Kommunikationsschema benötigt, das systemseitig die Anforderungen der Applikationen hoher Kritikalität sicherstellt. Das KAS-Konzept sieht vor, dass die Kommunikation nur über definierte Kommunikationskanäle stattfindet und dass der Abgriff der Daten zu keinerlei Störungen im abgegriffenen System (Zustandsänderung, Laufzeitverhalten, Integrität ...) führt. Abb. 4.16 präsentiert ein Beispiel für ein solches Kommunikationsschema für das in Bild 4.15 dargestelltes System. Die Control-Partition beinhaltet die klassische Prozessführungsanwendung. Sie ist eine kritische Anwendung und darf daher nicht durch unautorisierte Anwendungen verändert werden. Die Informationen aus der Control-Partition müssen trotzdem für die weiteren Verarbeitungen zu anderen Einheiten (beispielsweise Cloud und O&M) geschickt werden. Die Control-Partition darf nur mit dem Verwaltungssystem kommunizieren. Die Informationen werden über eine unidirektionale Kommunikation zum Verwaltungssystem geschickt und dieses leitet jene an die anderen Anwendungen weiter. Dabei dient die Interface-Partition der Anbindung ex-

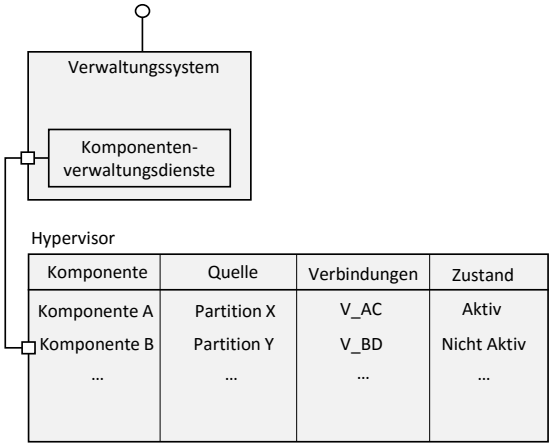

Abbildung 4.14: Komponentenverwaltung

terner Komponenten (z .B. Cloud). Da das Interface und die O&M-Partition unterschiedliche Kritikalitätsniveaus haben, werden zwei verschiedene Informationsdioden für diese Kommunikation eingesetzt. Die Anfragen der Applikationen mit einem niedrigeren Kritikalitätsniveau an Applikationen mit einem höheren Kritikalitätsniveau erfolgen durch den Typ 2-Kommunikationsport. Die Control-Partition hat eine Lese-Berechtigung für das Verwaltungssystem, um die Anfragen von diesem zu lesen. Das Verwaltungssystem hat wiederum eine Lese-Berechtigung für die O&M-Partition und das Interface. Dies verhindert die direkte Kommunikation von Applikationen unterschiedlicher Kritikalität. Das Kommunikationsschema ist in Abb. 4.16 dargestellt. In diesem Schema erfolgt die Kommunikation zwischen der Control-Partition und anderen Partitionen über das Verwaltungssystem.

Abb. 4.17 zeigt die Kommunikation zwischen der Interface-Partition und der O&M-Partition. Die Kommunikation erfolgt direkt zwischen diesen Partitionen.

4.6 Evaluation anhand der Anforderungen an die Architektur

In Kapitel 3 wurde eine Reihe von Anforderungen an die Architektur definiert. Im folgenden Abschnitt wird diskutiert, inwieweit diese durch das KAS erfüllt werden.

- **Sichere Übertragung von Feldinformationen in die Cloud:** Dank des Hypervisors wird eine Trennung der Komponenten ermöglicht, die über eine rückwirkungsfreie Kommunikationsverbindung miteinander kommunizieren können. Dies verhindert eine direkte Kommunikation der kritischen Komponenten mit überlagerten Anwendungen.

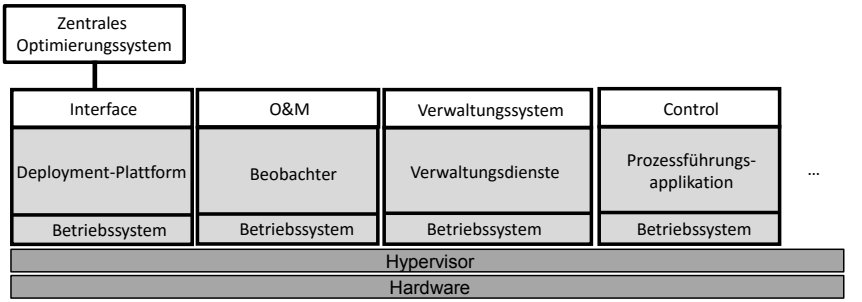

Abbildung 4.15: Systemarchitektur

- **Sichere Kommunikation zwischen Anwendungen mit unterschiedlicher Kritikalitätsstufe:** Die Kommunikation läuft über festgelegte Kommunikationsports und wird von einem Verwaltungssystem überwacht.
- **Implementierung zusätzlicher Funktionalitäten zur Analyse und Optimierung während der Laufzeit:** Neue Komponenten (FBs, Prozessführungskomponenten) können zur Laufzeit mittels der Kommunikation mit überlagerten Anwendungen heruntergeladen werden. Der Vorgang wird vom Verwaltungssystem getriggert.
- **Parallele Ausführung von zusätzlichen Applikationen auf der gleichen Hardware, wie z. B. lokalen Simulationsaufgaben:** Zusätzliche Komponenten mit unterschiedlichen Anforderungen können mittels Hypervisor und Virtualisierung auf der Hardware betrieben werden.
- **Unterstützung der lokalen Verwaltung und Überwachung der untergeordneten Komponenten:** Das Verwaltungssystem bietet Dienste für die Orchestrierung und Verwaltung der Komponenten an.

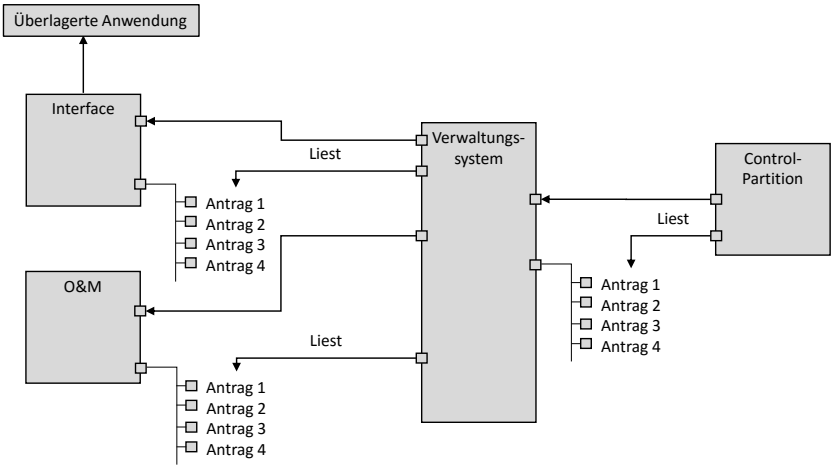

Abbildung 4.16: Indirekte Kommunikation zwischen der Control-Partition, der Interface-Partition und der O&M-Partition

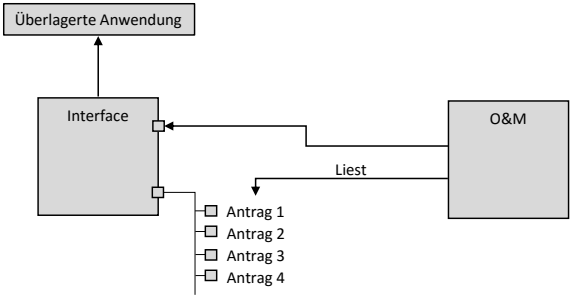

Abbildung 4.17: Direkte Kommunikation zwischen der O&M-Partition und der Interface-Partition

5 Anwendungsszenarien in der Automatisierungstechnik

In diesem Kapitel wird gezeigt, wie sich typische Anwendungsszenarien der Automatisierungstechnik auf die KAS-Struktur abbilden lassen.

5.1 Architektur der Automatisierungspyramide

Zur Gliederung der Automatisierungsaufgabe wird gerne das Bild der Automatisierungspyramide herangezogen. In klassischer Form hat die Pyramide sowohl einen funktionalen als auch einen hardwaretechnischen Aspekt. Der funktionale Aspekt ist im linken Teil von Abb. 5.1 dargestellt, der hardwaretechnische Aspekt im linken Teil von Abb. 5.2. Die Rechte Seite der Abbildungen hingegen präsentiert die funktionale und hardwaretechnischen Aspekt im Kontext von I4.0 [106]. Die Funktionalitäten (Abb. 5.1) lassen sich in zwei Hauptkategorien unterteilen, nämlich der O&M-Ebene und der Automatisierungsebene. Die O&M-Ebene beinhaltet Funktionalitäten wie Assetverwaltung, Datenanalyse. Die Funktionen, die operativ für die Prozessführung benötigt werden, gehören zur Automatisierungsebene. Abb. 5.2 (links) zeigt die klassische Hardware-Struktur in der Automatisierungstechnik. In dieser Architektur laufen die Anwendungen auf unterschiedlichen Hardware-Komponenten. In der virtualisierten Architektur hingegen (Abb. 5.2 rechts) besteht die Möglichkeit unterschiedliche Anwendungen auf der gleichen Hardware zu betreiben. Dabei kann eine Anwendung in eine prozessnahe und eine prozessferne Anwendungskomponente aufgeteilt werden. Beispielsweise kann, wie in Abb. 5.1 dargestellt, eine prozessnahe O&M-Anwendung definiert werden, die auf der gleichen Hardware, auf der Prozessführungsebene läuft und lokale Optimierungsfunktionen anbietet, während eine globale O&M-Anwendung auf der überlagerten Ebene betrieben wird. Eine solche virtualisierte Architektur kann auf unterschiedlichen Ebenen (beispielsweise Prozessführung und MES) eingesetzt werden, um verschiedene Anwendungen auf derselben Hardware voneinander zu trennen (beispielsweise globale O&M und Informationsmanagement (IM)).

5.2 Beispielhafte Anwendungspartitionen

In diesem Abschnitt werden einige typische automatisierungstechnische Anwendungen erläutert, die sich für die Kapselung in einer eigenen Partition anbieten.

5.2.1 Control-Partition

Die zu einer Anlage oder Teilanlage gehörenden Prozessführungskomponenten werden geeigneterweise in einer eigenen Partition zusammengefasst. Eine solche Partition wird

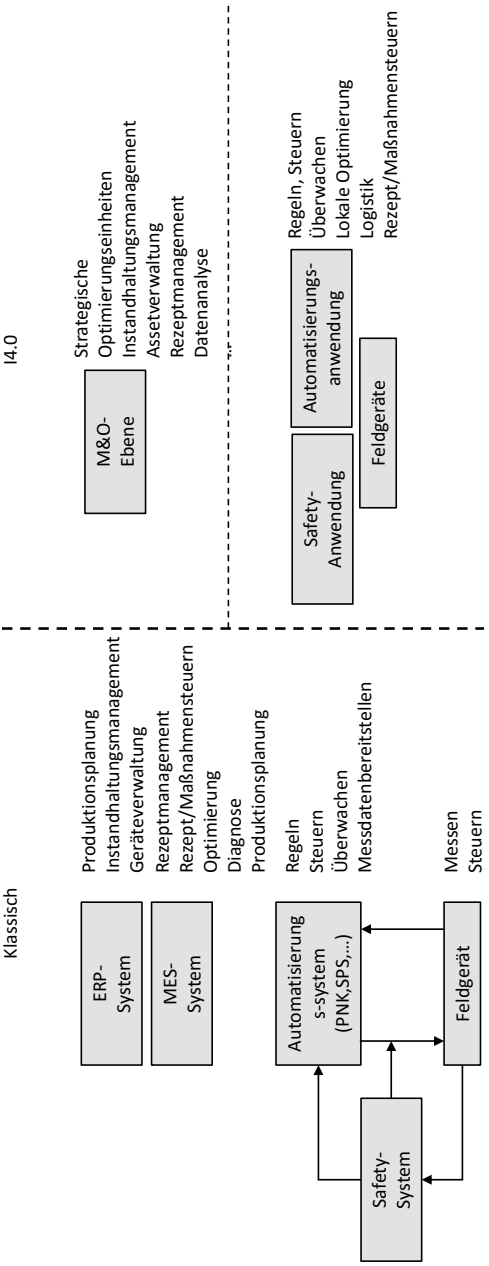

Abbildung 5.1: Funktionale Struktur des Automatisierungssystems

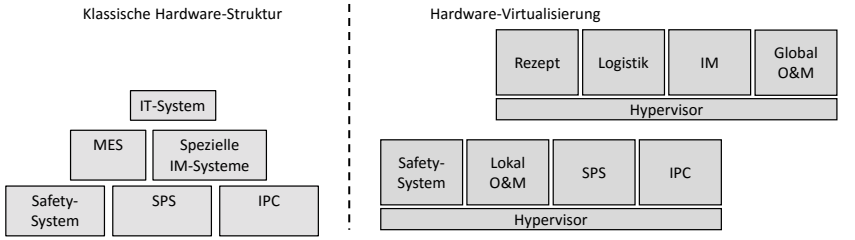

Abbildung 5.2: Hardwaretechnischer Aufbau des Automatisierungssystems

als Control-Partition bezeichnet. Diese muss hohe Anforderungen in Bezug auf Echtzeitfähigkeit, Robustheit, Handhabbarkeit und Sicherheit erfüllen. Die Prozessführung basiert beispielsweise auf dem in Kapitel 2 vorgestellten Betriebsmittel- und Maßnahmenmodell. Die Prozessführungsapplikationen werden beispielsweise in den Sprachen der IEC 61131-3 oder der IEC 61499 implementiert. Die Prozessführungsapplikation ist in einer Laufzeitumgebung implementiert, die für die Ausführung der Applikation zuständig ist. Diese muss von der umgebenden Partition oder einem umgebenden Container zur Verfügung gestellt werden. Die Control-Partition ist die einzige, die Zugriff auf die I/Os hat (Read/Write). Wie bereits beschrieben, bestehen einerseits die Möglichkeit des containerbasierten Aufbaus und andererseits die Möglichkeit der Verwendung von dedizierten Applikationen für die interne Struktur der Partition. Eine wandelbare Produktion ist das

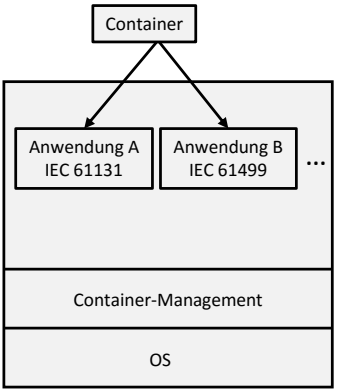

Abbildung 5.3: Struktur der Control-Partition

Ziel verschiedener Initiativen für zukünftige Automatisierungssysteme. Die vorgeschlagene Architektur stellt eine Basis für eine wandelbare Produktion bereit, in dem sie eine Verwaltung unterschiedlicher Prozessführungsapplikationen, sowie das Deployment neuer Applikationen, entsprechend der Anwendungsszenarien zulässt. Deployment zur Laufzeit

ermöglicht die Anpassung des Systems an die neuen Anforderungen und die Ausführung neuer Steuerungs- oder Verwaltungsaufgaben. Gemeinsam führen diese Eigenschaften zu einer erhöhten Dynamik des Systemverhaltens und bilden daher eine Basis für die Wandelbarkeit. Außerdem bieten die Partitionen eine hardwareunabhängige Umgebung zur Ausführung diverser Applikationen. Dies erhöht die Betriebsmöglichkeit von Applikation auf der selben Hardware und die Portabilität.

5.2.2 O&M-Partition

Die KAS-Architektur stellt eine Infrastruktur für den Betrieb einer prozessparallelen Simulation bereit. Dazu kann ein Simulationssystem in einer Partition realisiert werden. In vielen Fällen liegt das zu simulierende Modell in einer modularen Struktur vor, die nicht aufgelöst, sondern gemeinsam in einer Co-Simulation realisiert wird. Hier bietet es sich an, die Simulationsfragmente in eigenen Containern zu kapseln. Die Container-Technologie und die Orchestrierung dienen zur dynamischen Gestaltung der Simulationsapplikation. Verschiedene Simulationsmodelle sind in Form von Docker-Containern in O&M-Partition gehostet. Die Simulationsfragmente werden durch das Verwaltungssystem verwaltet, um Co-Simulationen zu konfigurieren (Abb. 5.4).

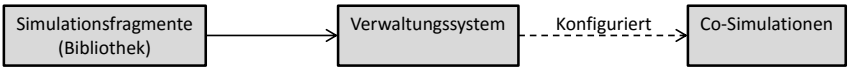

Abbildung 5.4: Konfigurationen von Co-Simulationen

6 Implementierung für eine Kaltwalzanlage

Als Anwendungsszenario dient die Optimierung des logistischen Durchlaufs von Paletten in einer Kaltwalzanlage. Dieses Szenario erfordert für die Durchführung von Optimierungsaufgaben und Deployment, dass das KAS zwischen der Prozessführung, der Simulation (O&M) und einer externen Cloud kooperiert.

6.1 Logistik

Die KAS-Architektur bietet eine Plattform zur Durchführung, Optimierung und Überwachung von Logistiksystemen. In dem hier verfolgten Kontext interessiert insbesondere die operative Steuerung von Intra-Logistiksystemen. Diese müssen eine effiziente Routen- und Ressourcenplanung anbieten. Die Routen- und Ressourcenplanung benötigt unter anderem die Zustände und Positionen der Logistikgeräte sowie die aktuelle Verkehrssituation, um eine effiziente Planung durchzuführen. Nach der Bearbeitung der Zustände werden Pläne generiert, die ins Logistiksystem integriert werden müssen. Abb. 6.1 präsentiert die Abbildung der Anwendungen auf die KAS-Architektur. Die Verwaltungs-

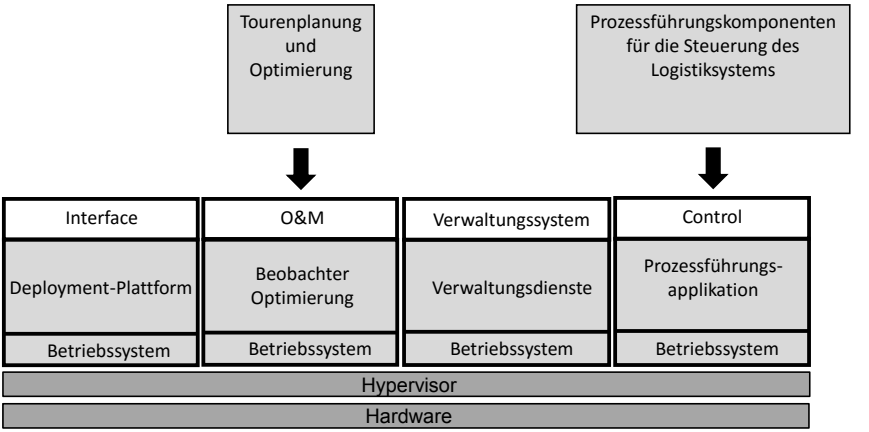

Abbildung 6.1: Logistiksystem

und Interface-Partition aus Abb. 6.1 entsprechen den zwei Systempartitionen der KAS-Architektur. Die Steuerung bildet die Control-Partition ab, die Tourenplanungs- und

Optimierungs-Partition bildet hingegen die O&M-Partition ab. Für die Optimierung und das Deployment sind die folgenden Schritten nötig:

- Optimierung
  - Abrufen der Informationen aus der Steuerung
  - Durchführung lokaler Optimierungen in der Tourenplanungs- und Optimierungs-Partition
- Deployment
  - Deployment in die Interface-Partition
  - Validierung
  - Integration in die Steuerung

6.2 SMS-Demonstrator

Die KAS-Architektur wird für die Steuerung des SMS-Demonstrators eingesetzt. Der SMS-Demonstrator simuliert das gesamte Transportsystem einer Kaltwalzanlage. Der Demonstrator besteht aus einem IPC für die Profibus-Anbindung, einem Embedded-System und einem Server für die MATLAB-Simulationsmodelle. Die Aktor- und Sensordaten werden über ein emuliertes Feldbussystem (Profibus) zwischen Simulator und Automatisierungssystem ausgetauscht (Abb. 6.2).

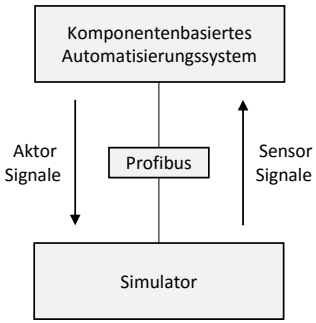

Abbildung 6.2: Aufbau

Die Hardwarekomponenten und der Aufbau der Anlage sind in Abb. 6.11 und 6.12 dargestellt. Ziel dieser Anlage ist die Realisierung einer virtuellen Inbetriebnahme. Sämtliche Anlagenfunktionen werden über einen Hybrid-Simulator simuliert. Der Simulator stellt alle Anlagensignale mit den Rechenschritten von $100\,\mu s$ bereit. Diese werden über emulierte Feldbussysteme an die Automatisierung übertragen.

Bestandteile

Abb. 6.3 zeigt das vom Demonstrator simulierte SMS-Kaltwalzwerk. Dieses besteht aus drei Hauptkomponenten [98], [35]:

- **Rollgänge**: Die Rollgänge bilden ein Förderband, um die Coils entlang der Anlage zu transportieren. Sie sind die einzigen aktiven Komponenten in der Anlage und werden mit Motoren angetrieben. Jeder Rollgang ist mit fünf Sensoren ausgestattet. Einer der Sensoren wird zur Erkennung der Palette eingesetzt, während die restlichen vier Sensoren die Aufgabe der Positionserkennung übernehmen. Das Simulationsmodell beinhaltet drei Arten von Rollgängen:
  - **Verschieber Wagen**: Sie können sich entlang der Y-Achse bewegen.
  - **Drehteller**: Sie können sich um die Z-Achse rotieren.
  - **Ofen**: Sie dienen zur Erwärmung der Coils.
- **Palette**: Sie werden zum Transport der Coils eingesetzt.
- **Coil**: Sie stellen die Aluminium-Coils dar.

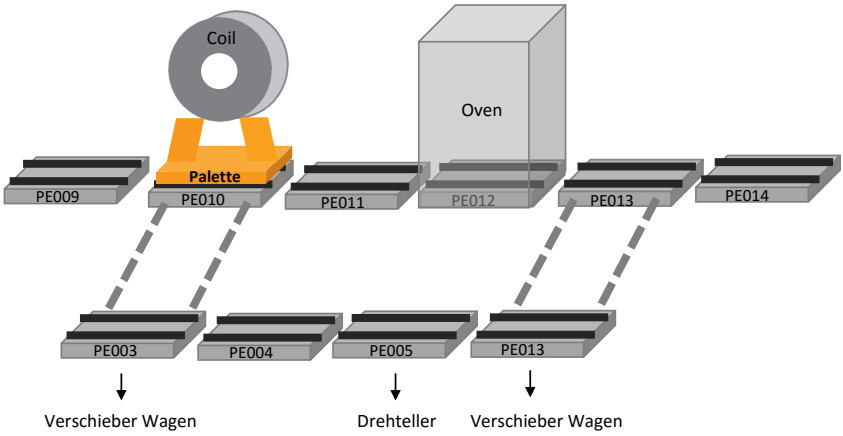

Abbildung 6.3: SMS-Demonstrator

Die Steuerungshierarchie dieser Anlage basiert auf dem in Kapitel 2 diskutierten Betriebsmittel- und Maßnahmenmodell. Die Rollgänge bilden die ESE-Ebene. Die Paletten orchestrieren diese, um die Produktionsaufträge umzusetzen. Das bedeutet, dass die Paletten in der Steuerungshierarchie die Rolle der GSEs übernehmen. Jede Palette ist einer GSE zugewiesen, um die ESEs (Rollgänge) entsprechend des derzeitigen Rezepts zu orchestrieren. Die Paletten können auch die entsprechenden ESEs belegen, um eine Kollision zu verhindern. Darüber hinaus bilden die Coils die Maßnahmen-Ebene. Diese Hierarchie ist in Abb. 6.4 dargestellt. Die GSEs agieren wie Klienten. Sie können auf verschiedenen Embedded-Systemen oder Hardware-Komponenten installiert werden, um die Anlage zu steuern. In dieser Arbeit wurden GSEs auf einem IMX6-Board implementiert.

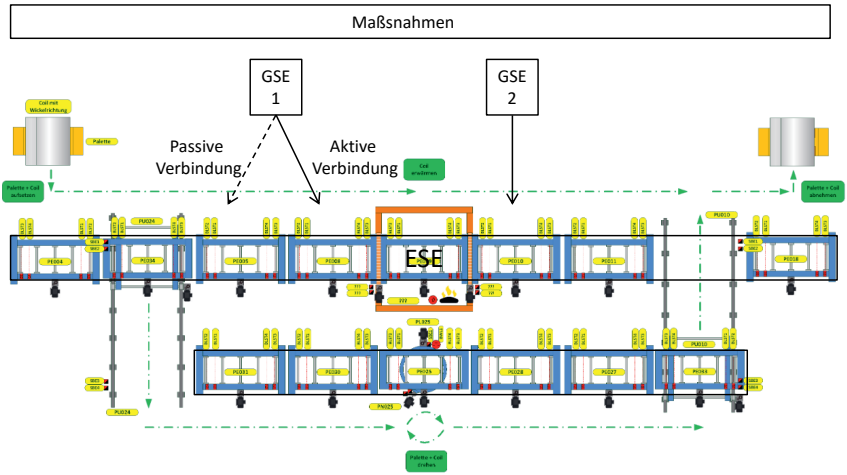

Abbildung 6.4: SMS-Demonstrator

6.3 Aufbau

Abb. 6.5 zeigt den Aufbau der Implementierung. Die linke Seite stellt das IMX6-Board und die darauf implementierte KAS-Architektur dar. Diese kommuniziert mit dem SMS-Demonstrator über eine TCP/IP-Kommunikationsschnittstelle. In diesem Aufbau sind die ESEs auf dem eingebetteten System des SMS-Demonstrators und die GSEs auf dem IMX6-Board implementiert.

6.4 Verification of Request

Für die prototypische Umsetzung des VoR werden fünf Partitionen auf einem IMX6-Board erstellt. Diese Partitionen (Verwaltungssystem, Control-Partition 1 und 2, Interface und O&M) beinhalten die in Kapitel 4 genannten Anwendungstypen. Darüber hinaus werden sie mit den in Kapitel 4 ebenfalls genannten Rechten, Eigenschaften und Kommunikationsports ausgestattet. Die Konfiguration der Partitionen für diesen Anwendungsfall ist im Anhang A dargestellt. Für die Implementierung wird angenommen, dass während des Betriebs eine neue Produktionslinie (Abb. 6.6) zur aktuellen Anlage (Abb. 6.4) hinzugefügt wird. Die neue Anlagenstruktur erfordert eine neue Prozessführungsanwendung. Das Deployment, die Integration und die Aktivierung der Prozessführung wird in diesem Abschnitt erläutert. Die Implementierung erfordert die folgenden Partitionen:

- **Das Interface:** Die Interface-Partition wird für das Deployment der Komponenten benötigt.
- **Die O&M-Partition:** Eine Simulationsanwendung validiert das Feedback, bevor es in die Prozessführung integriert wird.

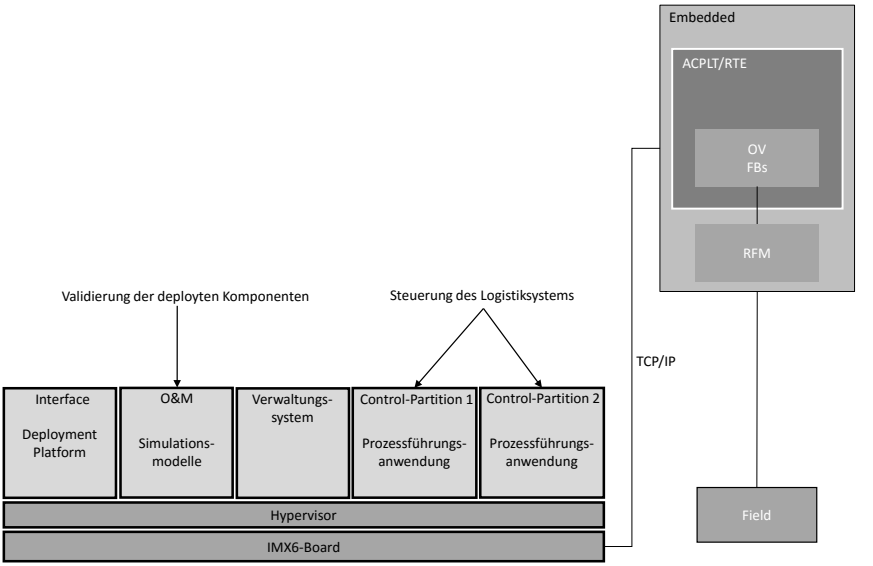

Abbildung 6.5: Aufbau

- **Das Verwaltungssystem:** Das Verwaltungssystem triggert die oben genannten Vorgänge.
- **Die Control-Partition:** Die Control-Partitionen werden für die Prozessführung benötigt. Die zweite Control-Partition wird für die Aktualisierung der Prozessführung eingesetzt.

Interface und Deployment

Das Verwaltungssystem triggert das Deployment der neuen Prozessführungsanwendung in der Interface-Partition. Abb. 6.7 präsentiert den Deployment- und Redeployment-FB. Er ermöglicht das Deployment von neuen Komponenten auf dem IMX6-Board. Darüber hinaus führt er das Redeployment von Komponenten durch. Für die Durchführung des Deployments und Redeployments werden die Ziel- und Quell-Informationen benötigt (Servername und Pfad zu den Komponenten) [34]. Dieser FB besteht aus einer getvar- und einer setvar-Funktion (Abb. 6.8, Abb. 6.9).

Validierung und Inbetriebnahme

Für die Inbetriebnahme werden zwei Control-Partitionen eingesetzt, zwischen denen gewechselt werden kann. Eine Partition beinhaltet die aktuelle Prozessführungsanwendung für die Steuerung der Anlage. Die zweite Partition wird für die Aktualisierung der Prozessführungsanwendung eingesetzt. Nach der Aktualisierung und Synchronisation der Pro-

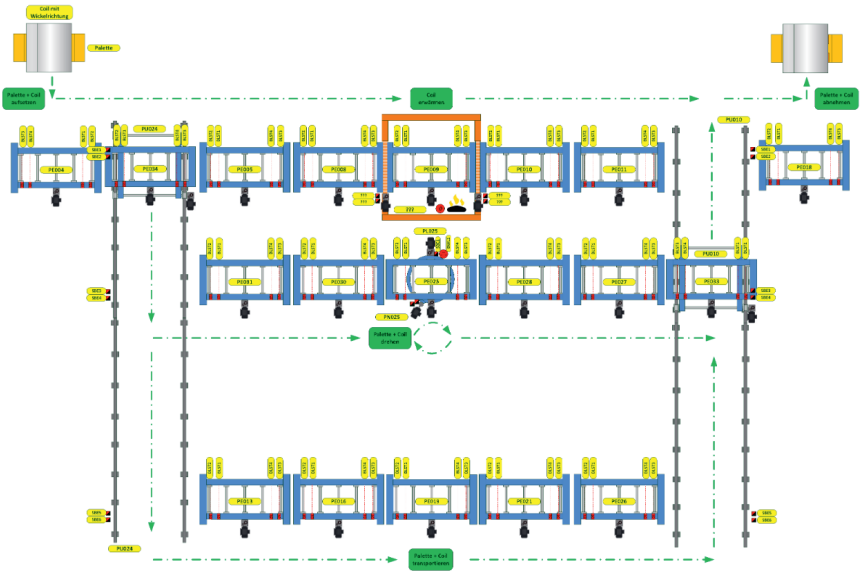

Abbildung 6.6: Die erweiterte Produktionsanlage

zessführungsanwendung in der zweiten Partition werden die Partitionen gewechselt, damit die zweite Partition die Steuerung der Anlage übernimmt. Dieser Prozess besteht aus den folgenden Schritten:

- Die neue Prozessführungsanwendung wird durch O&M-Partition validiert.
- Die neue Prozessführungsanwendung wird in der zweiten Control-Partition (deaktivierte Partition) implementiert.
- Wesentlichen Zustände der Anlage (z. B. die aktuelle Position der Paletten, die Zustände der Prozessführungskomponenten) werden in beiden Partitionen synchronisiert.
- Die aktuelle Prozessführung (Control-Partition) wird deaktiviert.
- Die neue Partition wird aktiviert.

Nach diesen Schritten übernehmen die neue Prozesssteuerungsanwendung und die neue Control-Partition die Steuerung der Anlage. Dies erfordert auch ein neues Schedulingschema. Die Ressourcen der aktuellen Partition müssen der neuen Partition zugeteilt werden. Da die aktuelle Control-Partition deaktiviert wurde, benötigt sie keine Ressourcen mehr. Abb. 6.10 zeigt die Ressourcenverwaltung.

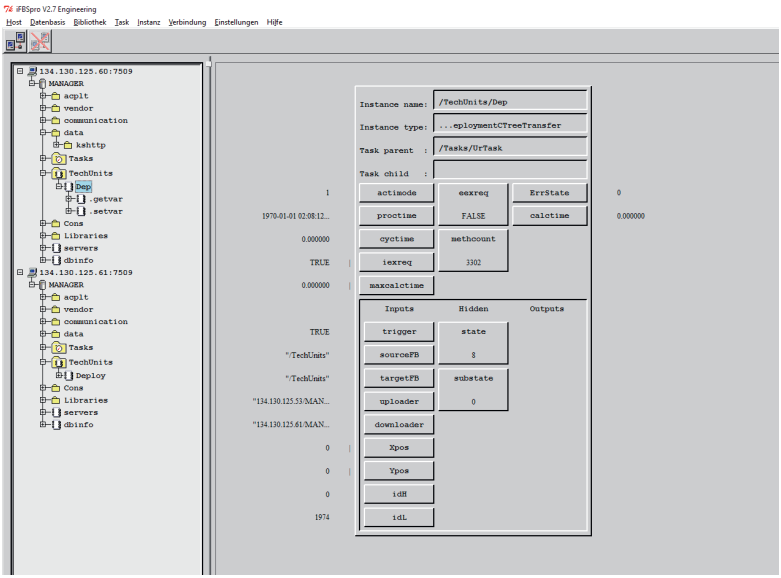

Abbildung 6.7: Deployment-Funktionsbaustein

6.4.1 Evaluation des VoR-Konzepts

Für VoR müssen die folgenden Schritte durchgeführt werden:

- Authentifizierung und Verifizierung des Feedbacks
- Plausibilitätscheck

In der NAMUR dürfen nur dann Änderungen in der CPC vorgenommen werden, wenn diese von einer VoR-Komponente stammen. In [24] werden die Anforderungen an VoR definiert. Die wichtigsten Anforderungen können wie folgt aufgelistet werden:

- Die Vertraulichkeit des Antrags: Der Antrag darf nicht für Drittparteien lesbar sein.
- Die Integrität des Antrags: Eine Änderung des Antrags während der Übertragung von der App in die VOR-Komponente muss erkannt werden.
- Verfügbarkeit der CPC-Domäne: Die VOR-Komponente darf die Verfügbarkeit des DCS/PCS nicht beeinträchtigen.
- Die Authentizität des Antrags: Nur beglaubigte und vertrauenswürdige Anwendungen dürfen Anfragen in die CPC-Domäne weiterleiten.
- Datenschutz der CPC: Keine internen Informationen von der CPC sollten durch die VoR-Komponente exponiert werden.

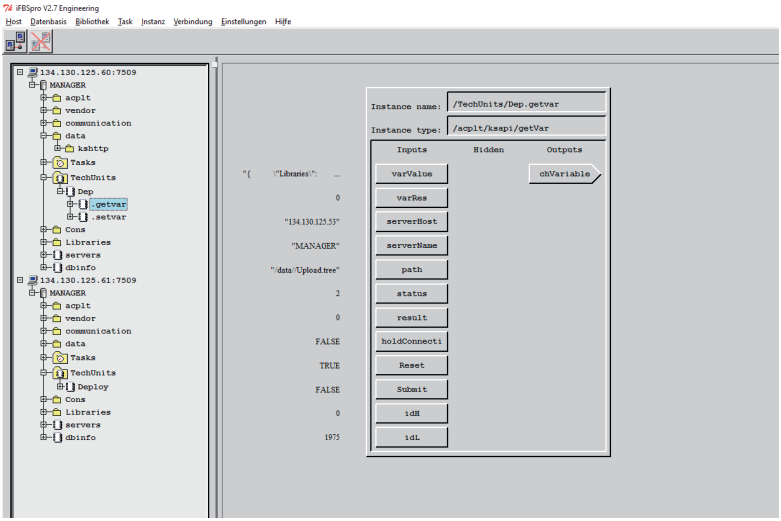

Abbildung 6.8: Deployment-Funktionsbaustein

Die VoR-Komponente entspricht der Validierungskomponente in der vorgeschlagenen Architektur. Einige Strategien zur Erfüllung der Anforderungen werden in den folgenden Punkten erläutert:

- Die Vertraulichkeit des Antrags: Der Antrag wird durch das Interface zum Verwaltungssystem weitergeleitet. Dieser kann nur gelesen werden, wenn die Applikationen ein Zugriffsrecht darauf haben. Allerdings muss die Kommunikation zwischen den externen Komponenten und dem Interface gesichert werden.

- Verfügbarkeit der CPC-Domäne: Die Anträge werden in die Prozessführung integriert, nach dem sie überprüft worden sind. Darüber hinaus ist die Prozessführung völlig isoliert und kann von anderen Partitionen nicht zugegriffen werden.

- Die Authentizität des Antrags: In der vorgeschlagenen Architektur schreibt das Verwaltungssystem die Anträge in seinen Kommunikationsport. Der Control-Partition ist es möglich auf diesen zuzugreifen. Dies verhindert eine direkte Verbindung zwischen der Control-Partition und den anderen Applikationen. Durch diesen Vorgang wird keine Anfrage an die Control-Partition weitergeleitet.

- Datenschutz der CPC: Die Kommunikationsverbindung der Validierungskomponente zum Verwaltungssystem ist nur mit einem Read-Recht ausgestattet. Durch diesen können nur Anträge im Verwaltungssystem-Kommunikationsport gelesen werden.

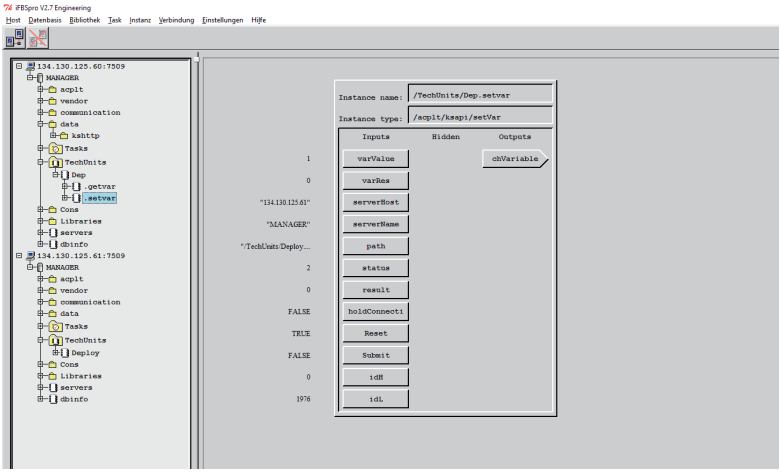

Abbildung 6.9: Deployment-Funktionsbaustein

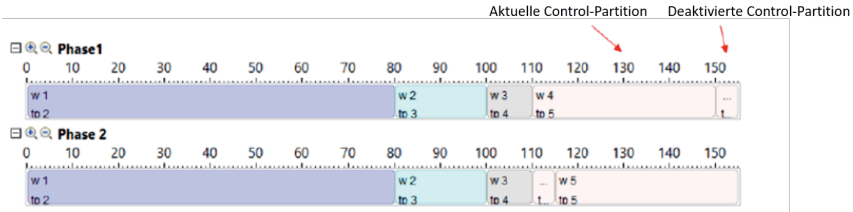

Abbildung 6.10: Aktuelle Ressourcenallokation

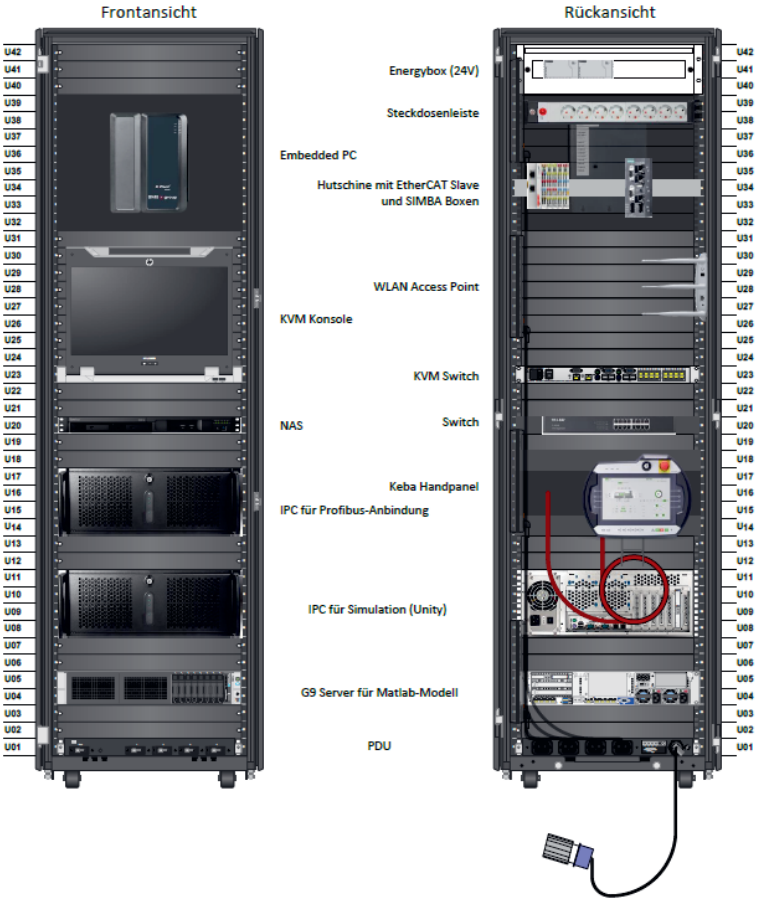

Abbildung 6.11: SMS-Demonstrator

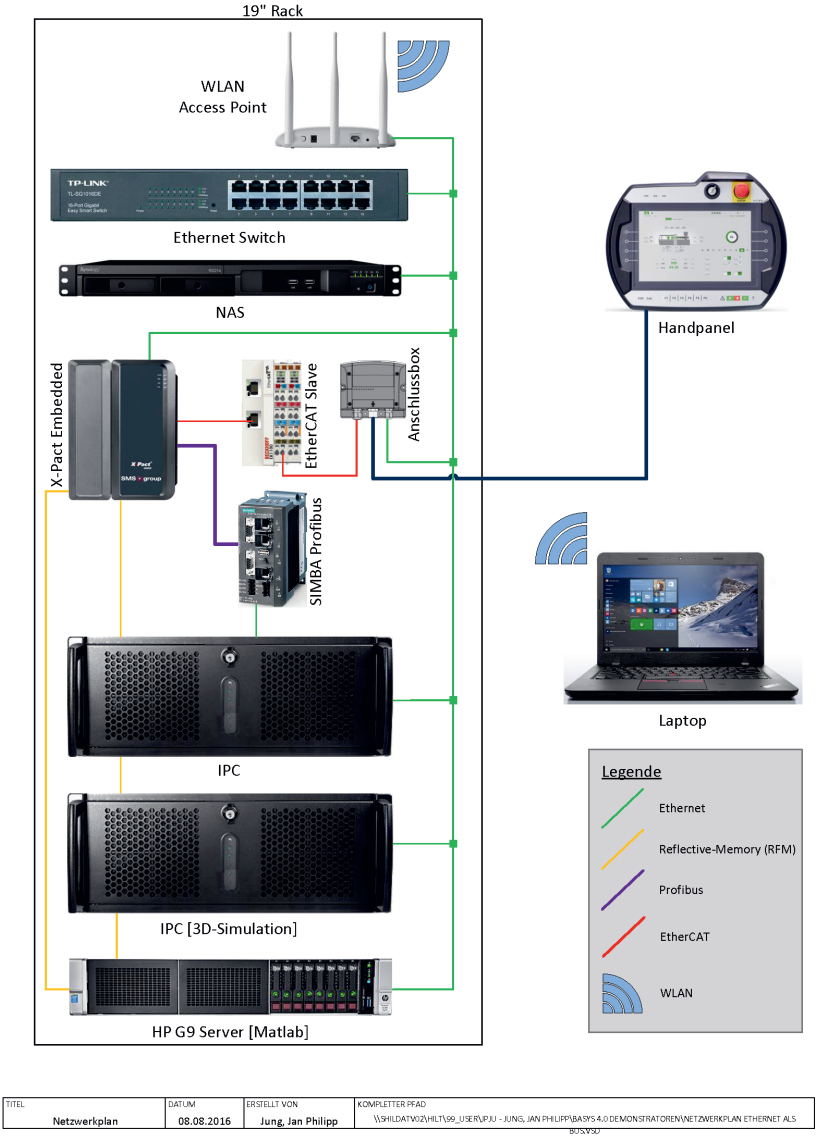

Abbildung 6.12: Struktur des Demonstrators

7 Validierung des Konzepts

7.1 Eingesetzte Technologien

Für die Implementierung werden ACPLT/RTE und ein PikeOS-Hypervisor verwendet. Zunächst ist eine Portierung von ACPLT/RTE auf das Echtzeit-OS (POSIX) des PikeOS-Hypervisors erforderlich. Die Portierung ermöglicht den Betrieb der Prozessführung auf einer virtualisierten Umgebung. Die Portierung von ACPLT/RTE und PikeOS wird im nächsten Abschnitt erläutert.

In diesem Abschnitt werden die Aufbaumöglichkeiten im Bezug auf existierende Anwendungen diskutiert. Diese sind in Abb. 7.1 grafisch dargestellt. Die erste Variante setzt Container-Technologien (z.B. Docker) ein, um eine Versions- und Variantenverwaltung sowie dynamisches Deployment mittels Containern zuzulassen. Abb. 7.1 (links) präsentiert einen Aufbau, in welchem Versionierung und Deployment von Applikationen mittels Container-Technologien umgesetzt sind. In diesem Fall kann zum Beispiel Linux als Betriebssystem in der Partition verwendet werden, um den Betrieb von Docker-Containern zu ermöglichen. Die überlagerte Ebene ist ein Container-Managementsystem zur Verwaltung der Container (beispielsweise Start und Stoppen). Die nächste Ebene beinhaltet Applikationen, die in mehreren Containern gekapselt sind.

Die zweite Variante ist in Abb. 7.1 (rechts) dargestellt. Sie präsentiert eine Möglichkeit, in der, im Kontrast zur ersten Variante, keine Container-Technologie eingesetzt wird. Im vorliegenden Fall können unterschiedliche Betriebssysteme benutzt werden (beispielsweise bietet POSIX-Interface den Vorteil leichtgewichtig und echtzeitfähig zu sein). Auf dem Betriebssystem werden die Anwendungen implementiert. Das Deployment neuer Komponenten kann mittels Serialisierung der Anwendungen und nötigen Bibliotheken auf POSIX erfolgen.

7.1.1 Portierung von ACPLT/RTE und PikeOS

ACPLT/RTE besteht aus einer Kernbibliothek (libov), die das Metamodell des Objektverwaltungssystems enthält, sowie aus zusätzlichen Bibliotheken, die bei Bedarf gelinkt werden können. Diese sind unten aufgeführt:

- **fb**: Die fb-Bibliothek beschreibt das Metamodell der FBs.
- **cshmi**: Die cshmi-Bibliothek enthält ein HMI-Modell für eine grafische Oberfläche.
- **TCPbind**: Die TCPbind-Bibliothek bietet eine Schnittstelle zum Netzwerk.
- **ksbase, ksxdr, kshttp**: Diese Bibliotheken stellen die Klassen und Funktionalitäten für die Kommunikation bereit.

Neben den genannten Bibliotheken gibt es außerdem noch einige zusätzliche Bibliotheken, die je nach Bedarf geladen werden können:

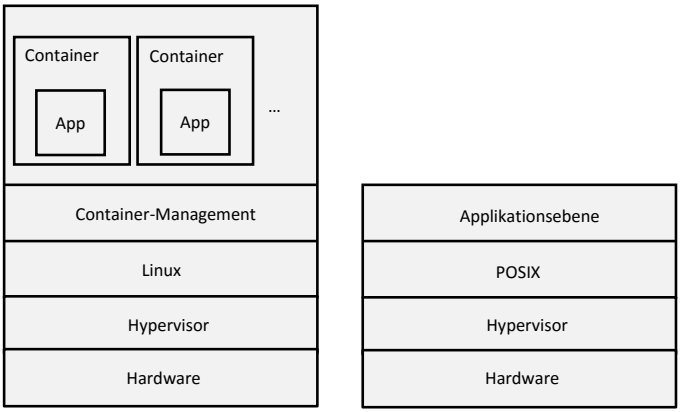

Abbildung 7.1: Interne Struktur der Partitionen

- **IEC61131stdfb**: Die IEC61131stdfb-Bibliothek enthält FBs nach der Norm IEC 61131.
- **vdivde3696**: Die vdivde3696-Bibliothek enthält FBs nach der Norm VDI.
- **Smrcs**: Die Smrcs-Bibliothek enthält Funktionen für die Interaktion mit dem SMS-Demonstrator.

ACPLT/RTE wurde in ANSI C implementiert und kann unter Linux und Windows ausgeführt werden. Die Liste der in ACPLT/RTE verwendeten libc-Funktionen sind in der Tabelle 7.1 aufgelistet. ACPLT/RTE kann sowohl auf Elinos (das einen vollständigen Linux-Kernel bereitstellt), als auch auf POSIX portiert werden. Die Tabelle 7.2 zeigt das Mapping und die Portierung von benötigten ACPLT/RTE-libc- und PikOS-libc-Funktionen.

Wie präsentiert sind die meisten Modifikationen für die Kommunikation, die Speicherzuteilung und die shared Objekte erforderlich. In dieser Arbeit wurden die in der Tabelle 7.2 präsentierten Modifikationen durchgeführt, damit ACPLT/RTE auf der Posix-Personality (Echtzeit OS) des PikeOS Hypervisors ausgeführt werden kann. Für eine Portierung auf ElinOS sind keine größeren Modifikationen erforderlich.

7.2 Prozessführung

Abb. 7.2 stellt das hierarchische Steuerungsmodell dar. Die unterste Ebene beinhaltet die auf dem eingebetteten System des SMS-Demonstrators implementierten ESEs. Die nächste Ebene präsentiert die auf dem IMX6-Board implementierten GSEs. Sie fungieren als Orchestrator und orchetrieren die ESEs.

Die Prozessführung ist nicht auf eine Partition beschränkt. Es können mehrere Partitionen erstellt werden, welche die Prozessführungsrolle übernehmen. Zu jedem Zeitpunkt kann nur eine davon aktiv sein. Die anderen Partitionen können für weitere Varianten und Versionen der Prozessführungsanwendungen eingesetzt werden. Wenn die verschiedenen

Tabelle 7.1: libc-Funktionen (ACPLT/RTE)

String/memory utility functions	atoi, atoll, memcmp, memcpy, memset snprintf, sprintf, strchr, strcmp, strcpy, strdup, strerror, strftime, strlen, strncmp, strncpy, strstr, strtod, strtol, strtoul, strtoull, tolower, toupper, vsnprintf
stdin/stdout	perror, stderr, stdout, puts
Socket/network IO	accept, bind, connect, freeaddrinfo, getaddrinfo, getnameinfo, getpeername, getsockname, listen, recv, setsockopt, socket, send
File handle API	close, lseek, open, read, select, write
Streams	close, clearerr, fclose, feof, ferror, fflush, fgets, flock, fopen, fprintf, fseek, fwrite
Memory allocation	calloc, free, malloc, realloc
Memory mapping	mmap, msync, munmap
Shared objects	dlclose, dlerror, dlopen, dlsym,
Time-related	gettimeofday, gmtime, localtime, mktime, nanosleep, setitimer, , time, timegm
Threads	pthread_create, pthread_join,

Tabelle 7.2: Portierung von benötigten ACPLT/RTE-libc- und PikOS-libc-Funktionen

String/memory utility functions	These functions can be replaced by PikeOS libc
stdin/stdout	PikeOS supports standard streams
Socket/network IO	POSIX specifies a Light Weight Internet Protocol (LWIP) some function should be modified. ACPLT/RTE should be provided by an interface to LWIP for the communication. LWIP is also TCP/IP compatible protocol.
File handle API	these funtions are supported
Streams	these functions are supported
Memory allocation	memory allocation is supported
Memory mapping	memory mapping functions must be modified.
Shared objects	shared libraries are not supported (*.so). The build process of ACPLT/RTE should modified to build all required libraries statically (*.a).
Time-related	important functions are supported
Threads	threads are supported

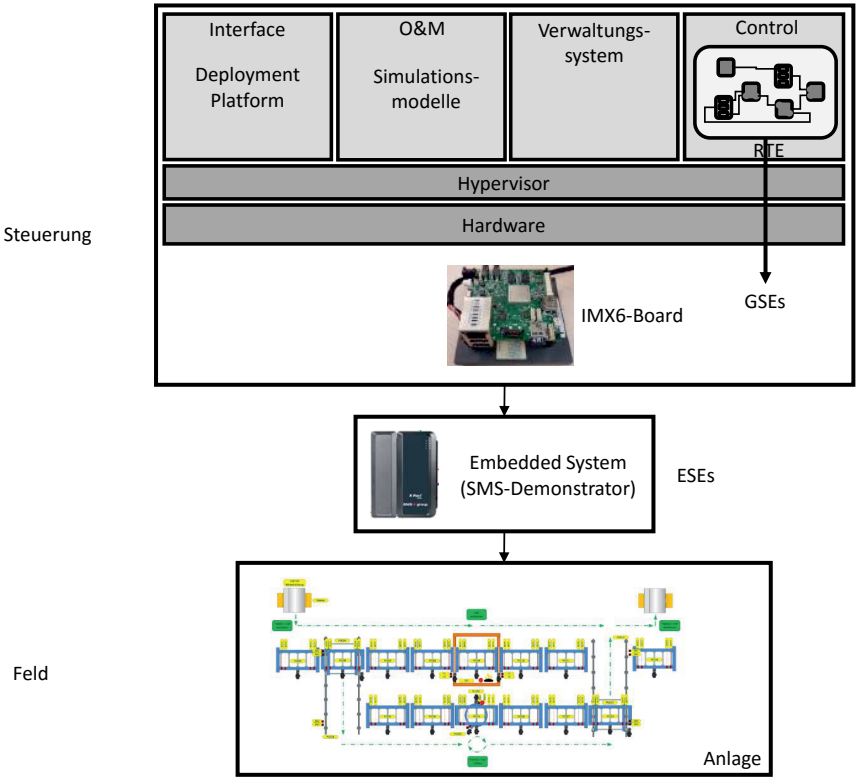

Abbildung 7.2: Prozessführungsebenen

Versionen sich nur in der Anwendungsebene unterscheiden, können sie auch in Docker-Containern innerhalb einer Partition gehostet werden. Wenn sie jedoch verschiedene Betriebssysteme benötigen, müssen sie in verschiedene Partitionen gekapselt werden. Zum Beispiel muss für den Wechsel zwischen Linux und POSIX die Partition gewechselt werden.

7.2.1 Laufzeitanalyse in virtualisierten und nicht virtualisierten Umgebungen

Eine Test-Applikation wurde erstellt, um die Prozessführung in verschiedenen Umgebungen zu vergleichen. In diesem Vergleich laufen die POSIX- und Linux-Prozesse auf einer virtualisierten Umgebung (IMX6-Board), wobei der Windows-Prozess auf einem PC ausgeführt wird. Wie in Abb. 7.3 präsentiert hat die Ausführung in POSIX keinen Jitter. Der Prozess in Linux (ElinOS Personality) hat einen geringen Jitter und der Windows-Prozess hat den höchsten Jitter.

In der nächsten Phase wurde die gleiche Analyse unter Hardwarebelastung durchgeführt. Der Prozess in POSIX hat einen bestimmten Anteil an Hardware-Ressourcen, welche nicht den anderen Applikationen zugeordnet werden können. Daher hat die Hardware-Belastung keinen Einfluss auf diesen Prozess und der Verlauf bleibt unverändert. Der Prozess in Windows hingegen weist einen erhöhten Jitter auf. Abb. 7.4 präsentiert den Vergleich zwischen POSIX und Windows.

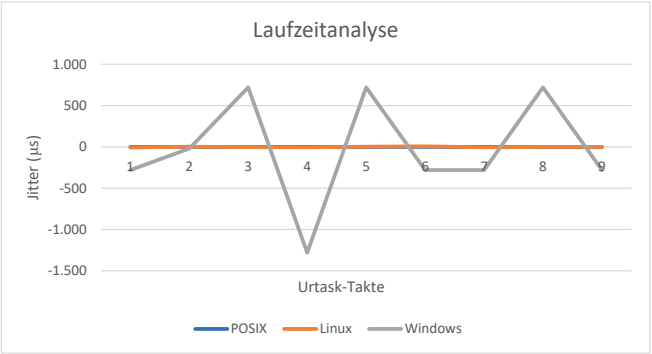

Abbildung 7.3: Laufzeitanalyse in POSIX, Linux und Windows

7.2.2 Kommunikation

Verschiedene Methoden zur Kommunikation zwischen Partitionen (die von PikeOS bereitgestellt werden) wurden in Kapitel 2 vorgestellt. Um diese Kommunikation echtzeitfähig und unidirektional zu gestalten, wurden bei der Implementierung Queueing-Ports verwendet. Die Queueing-Ports dienen auch als eine Basis für die NAMUR-Diode.

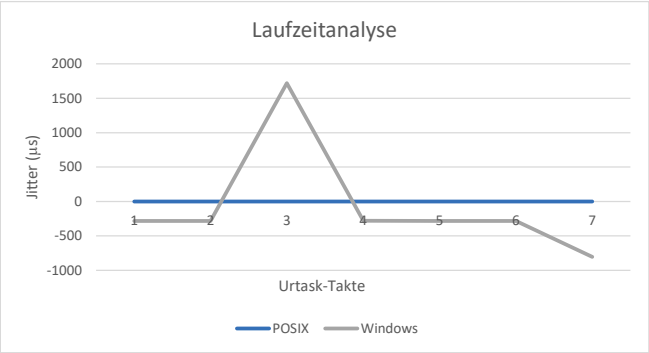

Abbildung 7.4: Laufzeitanalyse mit Hardware-Belastung in POSIX und Windows

Abb. 7.5 stellt ein Beispiel für die Implementierung der NAMUR-Diode dar. Die Implementierung besteht aus drei Partitionen, nämlich der CPC, der Simulation (ähnlich wie M+O) und dem Gateway. Die CPC und die Simulation kommunizieren miteinander indirekt durch das Gateway. Der Informationsfluss zwischen der CPC und dem Gateway findet

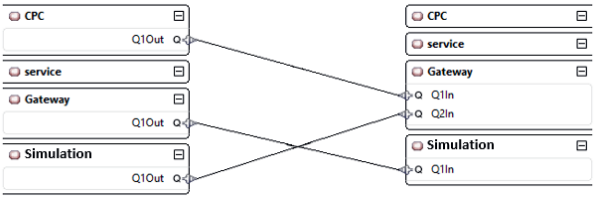

Abbildung 7.5: Kommunikationsports Zwischen der Partitionen

nur in einer Richtung statt, während die Simulation und das Gateway miteinander in beide Richtungen kommunizieren können. Die zweite Kommunikationsverbindung zwischen der Simulation und dem Gateway dient zur Sendung der Anfragen an das Gateway.

Die Queueing-Ports werden hauptsächlich für die Kommunikation zwischen Partitionen eingesetzt. Allerdings müssen diese für das vorgestellte Anwendungsszenario als ein Kommunikationsprotokoll zwischen FBs (in verschiedenen Partitionen) agieren. Aus diesem Grund ist auf dem ACPLT/RTE eine Schnittstelle zu Queueing-Ports implementiert, so dass die Kommunikation zwischen verschiedenen FBs über Queueing-Ports erfolgen kann. Um eine unidirektionale Kommunikation in ACPLT/RTE zu bewerkstelligen, müssen die Eingangs- und Ausgangsports entsprechend der Richtung des Informationsflusses definiert werden. Abb. 7.6 zeigt ein Beispiel für die Ports in ACPLT/RTE.

Die Kommunikation zwischen verschiedenen Servern in ACPLT/RTE erfolgte ursprünglich über fbcomlib. Abb. 7.7 präsentiert die fbcomlib-Latenzen in LINUX und POSIX. Die Ausführung in POSIX hatte eine konstante Latenz und keinen Jitter. Die

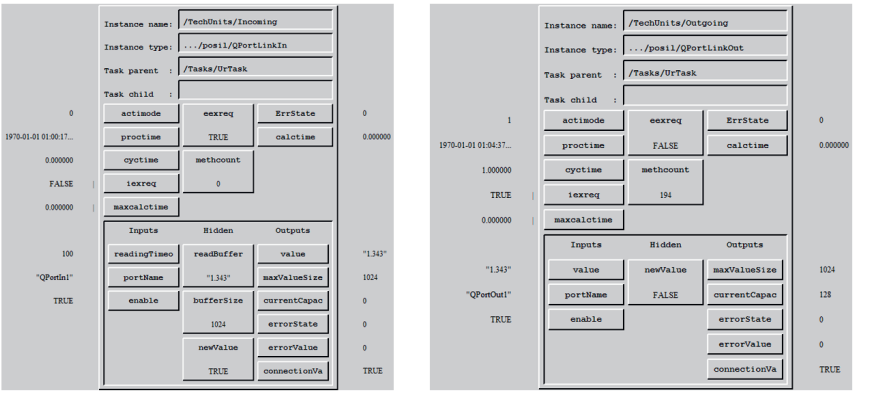

Abbildung 7.6: Unidirektional Kommunikation in ACPLT/RTE

Ausführung in LINUX hatte im Gegensatz zu POSIX einen Jitter und eine durchschnittliche Latenz von 3.410.054 μs (für die Analyse wurde $urtask = 1$ gesetzt). In einer virtua-

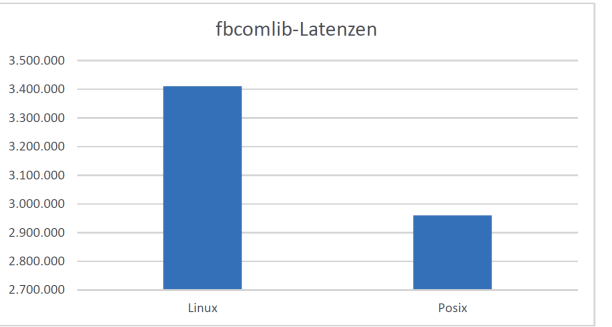

Abbildung 7.7: fbcomlib-Latenzen in POSIX und LINUX

lisierten Umgebung mit PikeOS werden jedoch Queueing-Ports für diese Kommunikation verwendet. Abb. 7.8 zeigt die Latenz dieser beiden Methoden (Zeit ist in μs gegeben, $urtask = 0.15$).

Die fbcomlib ist im Kern asynchron. Die Reduzierung des Urtask-Takts kann zum Packetverlust bei fbcomlib führen. Die durch beide Kommunikationsprotokolle empfangenen Daten (bei $urtask = 0.1$) sind in der folgenden Datei dargestellt. Eine Wertereihe von 4961 bis 4969 wurde durch beide Kommunikationsprotokolle gesendet. Wie präsentiert war der fbcomlib-Kommunikationsempfänger nicht in der Lage alle Werte zu empfangen (die Werte 4962, 4966 wurden nicht empfangen). Im Gegensatz zu fbcomlib hat der Queueing-Port-Empfänger alle Werte empfangen.

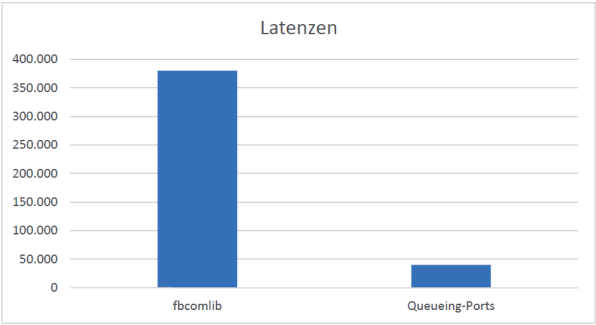

Abbildung 7.8: fbcomlib und Queueing Ports

```
fbcomlib:

[ACPLT/OV Info] Receiver: t=3699.719136; value=4961.000000
[ACPLT/OV Info] Receiver: t=3699.819136; value=4963.000000
[ACPLT/OV Info] Receiver: t=3699.919136; value=4964.000000
[ACPLT/OV Info] Receiver: t=3700.119136; value=4965.000000
[ACPLT/OV Info] Receiver: t=3700.219136; value=4967.000000
[ACPLT/OV Info] Receiver: t=3700.319136; value=4968.000000
[ACPLT/OV Info] Receiver: t=3700.519136; value=4969.000000

Queueing-Ports:

[ACPLT/OV Info] Receiver: t=4598.612313; value=4961.000000
[ACPLT/OV Info] Receiver: t=4598.712313; value=4962.000000
[ACPLT/OV Info] Receiver: t=4598.812313; value=4963.000000
[ACPLT/OV Info] Receiver: t=4598.912313; value=4964.000000
[ACPLT/OV Info] Receiver: t=4599.012313; value=4965.000000
[ACPLT/OV Info] Receiver: t=4599.112313; value=4966.000000
[ACPLT/OV Info] Receiver: t=4599.212313; value=4967.000000
[ACPLT/OV Info] Receiver: t=4599.312313; value=4968.000000
[ACPLT/OV Info] Receiver: t=4599.412313; value=4969.000000
```

7.2.3 Verwaltungssystem

Das Verwaltungssystem dient zur Überwachung der Ressourcenzuweisung, der Kommunikation und dem Deployment. Zudem bietet er Komponentenverwaltungsdienste an.

Ressourcenverwaltung

Die Hardware-Ressourcenzuweisung kann vom Verwaltungssystem den Anforderungen entsprechend geändert werden. Abb. 7.9 zeigt zwei verschiedene, vom Verwaltungssystem

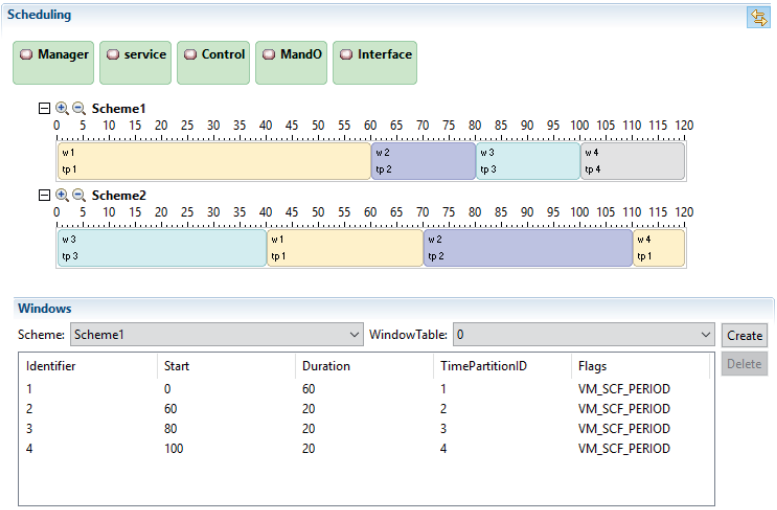

Abbildung 7.9: Scheduling Schema

generierte Schedulingschemata, welche zur Laufzeit angewendet werden können. Die Bedingungen für eine Änderung des Schedulingschemas werden erfüllt wenn:

- Partitionen 2, 3 und 4 mehr CPU-Zeit benötigen als ihnen ursprünglich zugewiesen wurde und Partition 1 weniger CPU-Zeit als vorgesehen benötigt.
- die Anwendungen in den Partitionen 2, 3 und 4 eine höhere Priorität als Partition 1 besitzen.

Verwaltungssystem als Gateway

Abb. 7.10 stellt die Kommunikationsports zwischen der Control-Partition, dem Verwaltungssystem und dem Interface dar.

Wenn die vom Interface benötigten Informationen nicht an dieses weitergeleitet werden dürfen, wird die Kommunikation verweigert. Dieser Vorgang ist in der folgenden Datei dargestellt. Die Meldung „Permission denied“ wird erzeugt, wenn die Daten nicht weitergeleitet werden dürfen, andernfalls findet die Kommunikation statt.

```
[1970/01/01 00:04:37.340041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:37.340041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:37.340041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:37.340041] [INFO] Man: transferred 8 bytes of data
```

```
[1970/01/01 00:04:37.340041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:37.360041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:37.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:38.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:39.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:40.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:41.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:42.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:43.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:44.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:45.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:46.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:47.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:48.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:49.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:50.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:51.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:52.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:04:53.840041] [INFO] Man: Permission Denied
[1970/01/01 00:04:54.840041] [INFO] Man: Permission Denied
[1970/01/01 00:04:55.840041] [INFO] Man: Permission Denied
[1970/01/01 00:04:56.840041] [INFO] Man: Permission Denied
[1970/01/01 00:04:57.840041] [INFO] Man: Permission Denied
[1970/01/01 00:04:58.840041] [INFO] Man: Permission Denied
[1970/01/01 00:04:59.840041] [INFO] Man: Permission Denied
[1970/01/01 00:05:00.840041] [INFO] Man: Permission Denied
[1970/01/01 00:05:01.840041] [INFO] Man: Permission Denied
[1970/01/01 00:05:02.840041] [INFO] Man: Permission Denied
[1970/01/01 00:05:03.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:04.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:05.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:06.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:07.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:08.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:09.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:10.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:11.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:12.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:13.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:14.840041] [INFO] Man: transferred 8 bytes of data
[1970/01/01 00:05:15.840041] [INFO] Man: transferred 8 bytes of data
```

Der Gateway-FB und seine Eingangs- (Prozessführung) und Ausgangsports (Interface) sind in Abb. 7.11 dargestellt.

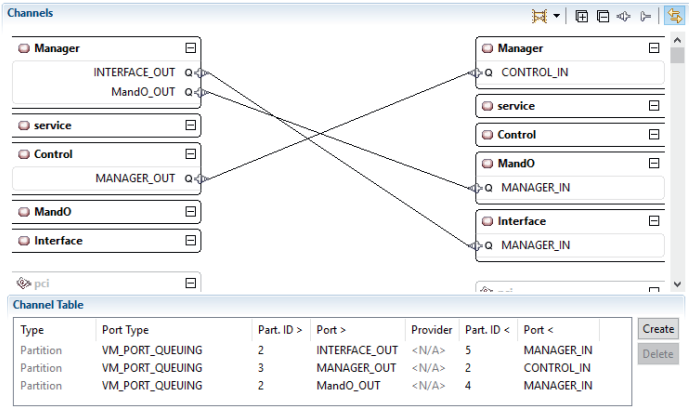

Abbildung 7.10: Kommunikationsports zwischen der Control-Partition und dem Interface

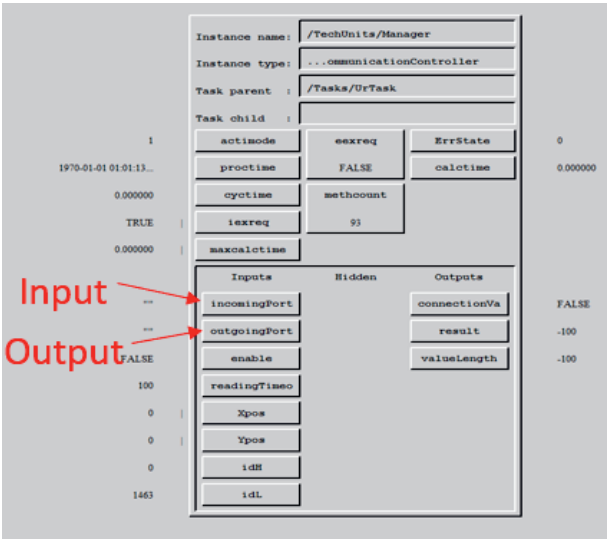

Abbildung 7.11: Verwaltungssystem als ein Gateway zwischen der Control-Partition und dem Interface

8 Fazit

Industrie 4.0 konfrontiert die industriellen Domänen mit neuen Herausforderungen. Die Geräte kommunizieren miteinander für eine erhöhte Effizienz, Optimierungsgrad und Wandelbarkeit. Wichtig ist, dass diese Vernetzung keinerlei Auswirkungen auf die Anforderungen der kritischen Anwendungen, wie z. B. die Verfügbarkeit, haben darf. Abgesehen von der Vernetzung, werden Analysefunktionen und KI-Algorithmen zunehmend in der Industrie Domänen eingesetzt. Die Anwendungen haben unterschiedliche Anforderungen an Echtzeit und Verfügbarkeit im Vergleich zu kritischen Anwendungen, müssen aber mit diesen kommunizieren. Die Vorteile des Betriebs dieser Anwendungen auf der gleichen Hardware können wie folgt aufgelistet werden:

- Skalierbarkeit: Durch die Verwaltung von Applikationen mit unterschiedlichen Anforderungen auf derselben Hardware reduziert sich die Anzahl der erforderlichen Hardware-Ressourcen. In der virtualisierten Architektur können sich mehrere Anwendungen eine Hardwareressource teilen. Dies führt zu einer besseren Skalierbarkeit.
- Freie Zuordnung auf die Hardwareressourcen: Die Partitionen können frei auf die zur Verfügung stehenden Hardware-Ressourcen übertragen werden. Dies erlaubt ein dynamisches Deployment und die Wiederverwendbarkeit der Partitionen.
- Vereinfachte Kommunikationsinfrastruktur: Innerhalb der Umgebung eines Hypervisors kann die Kommunikation zwischen den Partitionen einfach und effizient gestaltet werden. Der gesamte Netzwerk-Overhead entfällt.
- Reduzierung der Abhängigkeit zwischen Software und Hardware: Die Software wird auf dem OS des Hypervisors spezifiziert und kann auf beliebigen Instanzen des Hypervisors realisiert werden.
- Implementierung von Security-Aspekten auf der Systemebene: Security-Aspekte, wie Zugriffsrechte, Kommunikation, Ressource-Allokation können auf der Systemebene definiert werden.

In dieser Dissertation wurde ein Architekturkonzept für Steuerungsgeräte, bezeichnet als Komponentenbasierte Architektur für Automatisierungssysteme, vorgestellt. Die vorgestellte Architektur verwendet eine Hardware-Virtualisierung, um verschiedene Anwendungen auf Steuerungsgeräten zu trennen und zu integrieren. Die Systemfunktionen, die in dieser virtualisierten Umgebung implementiert wurden, bilden ein Verwaltungssystem und ein Interface. Diese beiden Systemfunktionen bilden den Kern der KAS-Architektur und wurden jeweils in eigenen Partitionen implementiert. Anwendungen können je nach QoS oder Strukturierungsanforderungen in gemeinsamen oder getrennten Partitionen oder Containern realisiert werden. Das KAS-System stellt ein leistungsfähiges internes Kommunikationssystem für den Datenaustausch zwischen den Partitionen und Containern einer Ressource bereit. Die KAS-Mechanismen erlauben eine effiziente und übersichtliche Überwachung

des Informationsflusses zwischen den internen Komponenten. Die Kommunikation zwischen den Anwendungen soll jedoch sowohl hinsichtlich der Kommunikationsrichtung als auch des Informationsflusses überwacht werden. Die Überwachung und Steuerung des Informationsflusses wird durch das Verwaltungssystem durchgeführt. Darüber hinaus bietet das Verwaltungssystem sowohl Dienste für das Ressourcen- und Komponentenmanagement als auch Konfigurationsdienste (Deployment und Inbetriebnahme) an. Durch Virtualisierung und Hypervisor-Technologie werden unabhängige Umgebungen (VMs) generiert, die gemäß der Anforderungen der Anwendungen konfiguriert werden können. Beispielsweise haben die Datenanalysefunktionen andere Anforderungen an Echtzeit und Verfügbarkeit im Vergleich zu Prozessführungsanwendungen. Daher müssen die Partitionen, welche diese Anwendungen kapseln auch unterschiedlich konfiguriert sein. Die KAS-Architektur ermöglicht den Betrieb, die Kooperation und die Vernetzung von Geräten in dem Industrie 4.0 Umfeld, ohne die Anforderungen der kritischen Applikationen zu gefährden.

KAS implementiert eine neue Architektur eines hierarchischen Komponentensystems. KAS vereinfacht und strukturiert die Implementierung von modularen wandelbaren Anwendungsstrukturen unter Berücksichtigung der jeweiligen QoS. Es erscheint erstrebenswert, ein solches Konzept zu verallgemeinern und als neue generische Struktur für die Automatisierungsarchitektur zu standardisieren. Die Partitionen kapseln Anwendungen mit unterschiedlichen Anforderungen. Standardisierte Konfigurationen der Partitionen gemäß der Anwendungen wird auch als eine zukünftige Arbeit betrachtet. Diese Konfigurationen beinhalten die Fähigkeiten der Partitionen, die Zugriffsrechte, die QoS-Eigenschaften, die Kommunikationsports usw. Dies ermöglicht auch eine Wiederverwendbarkeit dieser Partitionen.

Im bisherigen System werden die Daten direkt zwischen den Partitionen 1:1 übertragen. Für zukünftige Anwendungen erscheint es hilfreich im Verwaltungssystem eine Datenhaltung, in Form eines Publisher/Subscriber-Systems, zu implementieren. Abb. 8.1 illustriert die Kommunikation zwischen verschiedenen Anwendungen. Für die Kommunikation wird eine Methode zur Identifikation, sowie der Speicherung der Daten benötigt. Die Informationen müssen dazu mit einem Topic versehen werden. Dies inkludiert auch die Verwaltung der historischen Daten. Zustände, die von der Verwaltungskomponente abgerufen werden, werden in diesem System unter diversen Topics gespeichert. Die anderen Partitionen können diese Topics nach Bedarf abonnieren.

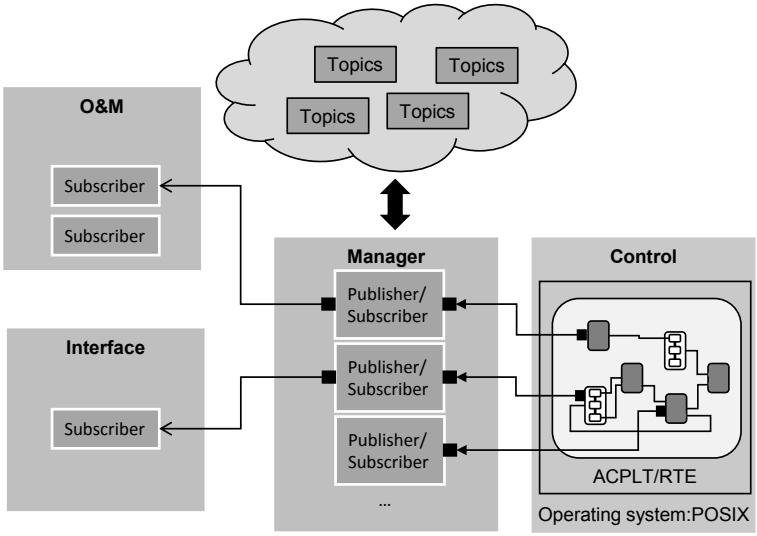

Abbildung 8.1: Aufrüstung mit einem Publisher/Subscriber-System

A Anhang

Dieser Anhang stellt die Partition-Konfigurationen dar.

```
<?xml version="1.0" encoding="us-ascii" standalone="no"?>
<Project xmlns="http://www.sysgo.com/xsd/prj/project-4.8.xsd" xmlns:xsi="
   http://www.w3.org/2001/XMLSchema-instance" productversion="4.2">
  <Integration name="posix-devel" profile="integration"
    target="arm_v7hf">
    <PathTable>
      <Path id="PIKEOS" location="F:\SYSGO\opt\pikeos-4.2\" />
    <Path id="PIKEOS_POOL"
        location="F:\SYSGO\opt\pikeos-4.2\target\arm\v7hf\" />
      <Path id="CUSTOM_POOL" location="F:\SYSGO\POSIX4\POOL" />
    </PathTable>
    <ConfigurationDomainTable>
      <!--start here-->
      <!--use CTRL + SPACE for suggestions-->
      <!--use 'validate' from the right click menu to validate your code
         -->
      <Group name="Build">
        <ComponentInstance name="Compilation Parameters"
          ref="Compilation Parameters" />
      </Group>
      <Group name="Application">
        <ComponentInstance name="POSIX Partition"
          ref="POSIX Partition">
          <ParameterValue name="PARTNAME" value="Manager" />
          <VmitConfigurationTable>
            <VmitConfiguration condition="true"
              isReference="true">
              <Partition
                Abilities="VM_AB_CACHE_CHANGE VM_AB_TRACE VM_AB_PSP_RESET
                    VM_AB_MONITOR VM_AB_MEM_CREATE VM_AB_PSP_CONSOLE
                   VM_AB_HM_INJECT_OTHER" CpuMask="-1"
                Identifier="$(PARTID)" MaxChildTaskCount="1" MaxFDCount="
                   20"
                MaxPrio="62" MultiPartitionHMTableID="0" Name="$(PARTNAME
                   )"
                SchedChangeAction="VM_SCHED_CHANGE_IGNORE"
                StartupMode="VM_PART_MODE_COLD_START" TimePartitionID="0"
                   >
                <FileAccessTable>
                  <FileAccess
                    FileName="rfs:ov_server_manager.conf"
                    AccessMode="VM_O_RD VM_O_MAP">
                  </FileAccess>

                  <FileAccess
                    FileName="shm:MandO_SHAREDMEMORY"
```

```
40                    AccessMode="VM_O_RD VM_O_WR VM_O_MAP">
41                  </FileAccess>
42                  <FileAccess
43                    FileName="shm:INTERFACE_SHAREDMEMORY"
44                    AccessMode="VM_O_RD VM_O_WR VM_O_MAP">
45                  </FileAccess>
46
47                  <ComponentReference
48                    ref="POSIX Process" />
49                </FileAccessTable>
50                <MemoryRequirementTable>
51                  <ComponentReference
52                    ref="POSIX Process" />
53                </MemoryRequirementTable>
54                <ProcessTable>
55                  <ComponentReference
56                    ref="POSIX Process" />
57                </ProcessTable>
58                <QueuingPortTable>
59                  <QueuingPort Name="INTERFACE_OUT"
60                    MaxMessageCount="4294900000" MaxMessageSize="
                        4294900000" Direction="VM_PORT_SOURCE">
61                  </QueuingPort>
62                  <QueuingPort Name="CONTROL_IN"
63                    MaxMessageCount="4294900000" MaxMessageSize="
                        4294900000" Direction="VM_PORT_DESTINATION">
64                  </QueuingPort>
65                  <QueuingPort Name="Mand0_OUT"
66                    MaxMessageCount="4294900000" MaxMessageSize="
                        4294900000" Direction="VM_PORT_SOURCE">
67                  </QueuingPort>
68                  <ComponentReference
69                    ref="POSIX Process" />
70                </QueuingPortTable>
71                <SamplingPortTable>
72                  <ComponentReference
73                    ref="POSIX Process" />
74                </SamplingPortTable>
75                <HMTable>
76                  <DefaultSwitch>
77                    <Default Action="P4_HM_PAC_IDLE"
78                      Code="0" Level="P4_HM_LEVEL_USER" Notify="0" />
79                  </DefaultSwitch>
80                </HMTable>
81              </Partition>
82            </VmitConfiguration>
83          </VmitConfigurationTable>
84        </ComponentInstance>
85        <ComponentInstance name="POSIX Process"
86          ref="POSIX Process">
87          <ParameterValue name="LWIP_CONFIG" value="true" />
88          <ParameterValue name="LWIP_TARGETIP_IF1"
89            value="134.130.125.60" />
90          <ParameterValue name="RAMSIZE" value="0x04000000" />
91          <ParameterValue name="FILE"
92            value="CUSTOM_POOL/ov_runtimeserver" />
```

```
93            <ParameterValue name="MUXA_ALL" value="true" />
94            <ParameterValue name="LWIP_GATEWAY_IF1"
95              value="192.168.0.11" />
96            <ParameterValue name="DESTNAME" value="Manager" />
97            <ParameterValue name="POSIX_TUNE" value="true"/><ParameterValue
                  name="POSIX_TUNE_VM" value="true"/><ParameterValue name="
                POSIX_TUNE_ARCH" value="true"/><ParameterValue name="
                POSIX_TUNE_FS" value="true"/><ParameterValue name="
                POSIX_TUNE_HEAP" value="true"/><ParameterValue name="
                POSIX_TUNE_P4_PRIO" value="true"/><ParameterValue name="
                POSIX_TUNE_SCHED" value="true"/><ParameterValue name="
                POSIX_TUNE_PTHREAD" value="true"/><ParameterValue name="
                POSIX_TUNE_MQ" value="true"/><ParameterValue name="
                POSIX_TUNE_PARAMS" value="true"/><ParameterValue name="
                POSIX_TUNE_TTY" value="true"/><AssignedDependencyTable>
98
99              <AssignedDependency cmp="imx_fec-vchan1"
100               provideId="channel" dependId="LWIP_DEVICE_IF1" />
101             <AssignedDependency cmp="muxa"
102               provideId="CHANNEL_02" dependId="IOFILE" />
103           </AssignedDependencyTable>
104         </ComponentInstance>
105       </Group>
106       <Group filename="driver/misc/devel.dom" name="devel"
107         path_id="PIKEOS_POOL">
108         <ComponentInstance name="monitor" ref="monitor">
109           <AssignedDependencyTable>
110             <AssignedDependency cmp="muxa" dependId="MONBIN"
111               provideId="monitor" />
112             <AssignedDependency cmp="muxa" dependId="MONCON"
113               provideId="mon_con" />
114             <AssignedDependency cmp="Monitor Master"
115               dependId="MON_MASTER" provideId="imon-master" />
116           </AssignedDependencyTable>
117         </ComponentInstance>
118         <ComponentInstance name="traceserver" ref="traceserver">
119           <AssignedDependencyTable>
120             <AssignedDependency cmp="muxa"
121               dependId="MUXA Channel" provideId="traceserver" />
122           </AssignedDependencyTable>
123         </ComponentInstance>
124         <ComponentInstance name="muxa" ref="muxa">
125           <ParameterValue name="HostIP" value="134.130.125.53" />
126           <ParameterValue name="TargetIP"
127             value="134.130.125.90" />
128           <ParameterValue name="GatewayIP"
129             value="134.130.125.126" />
130           <ParameterValue name="Netmask" value="255.255.255.0" />
131           <ParameterValue name="Channel2_Protocol"
132             value="telnet" />
133           <AssignedDependencyTable>
134             <AssignedDependency cmp="imx_fec-vchan0"
135               provideId="channel" dependId="FILE" />
136           </AssignedDependencyTable>
137         </ComponentInstance>
138       </Group>
```

```
139       <Group name="Default">
140         <ComponentInstance name="service.partition"
141           ref="service.partition">
142           <VmitConfigurationTable>
143             <VmitConfiguration condition="true"
144               isReference="true">
145               <Partition
146                 Abilities="VM_AB_TIMEPART_CHANGE VM_AB_MONITOR
                      VM_AB_PSP_CONSOLE VM_AB_TIMEPART_SETUP
                      VM_AB_HM_INJECT_OTHER VM_AB_PART_SET_MODE
                      VM_AB_MEM_CREATE VM_AB_PSP_RESET VM_AB_CACHE_CHANGE
                      VM_AB_TRACE"
147                 CpuMask="-1" Identifier="1" MaxChildTaskCount="20"
148                 MaxFDCount="128" MaxPrio="102" MultiPartitionHMTableID="0
                      "
149                 Name="service" SchedChangeAction="VM_SCHED_CHANGE_IGNORE"
150                 StartupMode="VM_PART_MODE_COLD_START" TimePartitionID="0"
                      >
151                 <FileAccessTable>
152                   <FileAccess
153                     AccessMode="VM_O_RD VM_O_WR" FileName="con:" />
154                   <ComponentReference ref="monitor" />
155                   <ComponentReference ref="traceserver" />
156                   <ComponentReference ref="muxa" />
157                   <ComponentReference
158                     ref="imx_uart-base" />
159                   <ComponentReference
160                     ref="imx_fec-base" />
161                 </FileAccessTable>
162                 <MemoryRequirementTable>
163                   <MemoryRequirement
164                     AccessMode="VM_MEM_ACCESS_RD VM_MEM_ACCESS_WR
                        VM_MEM_ACCESS_EXEC"
165                     Alignment="-1" CacheMode="VM_MEM_CACHE_CB" Contiguous
                        ="false"
166                     IsPool="true" MemRegionID="-1" MemRegionPartition="-1
                        "
167                     Name="_RAM_" PhysicalAddress="-1" Size="0x200000"
168                     Type="VM_MEM_TYPE_RAM" ZeroCount="0" />
169                   <MemoryRequirement
170                     AccessMode="VM_MEM_ACCESS_RD VM_MEM_ACCESS_WR
                        VM_MEM_ACCESS_EXEC"
171                     Alignment="-1" CacheMode="VM_MEM_CACHE_CB" Contiguous
                        ="false"
172                     IsPool="false" MemRegionID="-1" MemRegionPartition="
                        -1"
173                     Name="_KMEM_" PhysicalAddress="-1" Size="0x320000"
174                     Type="VM_MEM_TYPE_KMEM" ZeroCount="0" />
175                   <ComponentReference ref="monitor" />
176                   <ComponentReference ref="traceserver" />
177                   <ComponentReference ref="muxa" />
178                   <ComponentReference
179                     ref="imx_uart-base" />
180                   <ComponentReference
181                     ref="imx_fec-base" />
182                 </MemoryRequirementTable>
```

```
183                 <ProcessTable>
184                   <ComponentReference ref="monitor" />
185                   <ComponentReference ref="traceserver" />
186                   <ComponentReference ref="muxa" />
187                   <ComponentReference
188                     ref="imx_uart-base" />
189                   <ComponentReference
190                     ref="imx_fec-base" />
191                 </ProcessTable>
192                 <QueuingPortTable>
193                   <ComponentReference ref="monitor" />
194                   <ComponentReference ref="traceserver" />
195                   <ComponentReference ref="muxa" />
196                   <ComponentReference
197                     ref="imx_uart-base" />
198                   <ComponentReference
199                     ref="imx_fec-base" />
200                 </QueuingPortTable>
201                 <SamplingPortTable>
202                   <ComponentReference ref="monitor" />
203                   <ComponentReference ref="traceserver" />
204                   <ComponentReference ref="muxa" />
205                   <ComponentReference
206                     ref="imx_uart-base" />
207                   <ComponentReference
208                     ref="imx_fec-base" />
209                 </SamplingPortTable>
210                 <HMTable>
211                   <DefaultSwitch>
212                     <Default Action="P4_HM_PAC_IDLE"
213                       Code="0" Level="P4_HM_LEVEL_USER" Notify="0" />
214                   </DefaultSwitch>
215                 </HMTable>
216               </Partition>
217             </VmitConfiguration>
218           </VmitConfigurationTable>
219         </ComponentInstance>
220       </Group>
221       <Bsp align="0x00001000" arch="arm" boot="uboot"
222         bootstrats="uboot,uboot_unc,raw" endian="little"
223         filename="board/imx6q_sabrelite.bsp.dom" name="imx6q_sabrelite"
224         path_id="PIKEOS_POOL" proc="v7hf" wrdsz="32">
225         <Description>
226           Boundary Devices BD-SL-i.MX6 (formerly the Freescale
227           SABRE Lite board).
228         </Description>
229         <Group name="Monitor Kernel Drivers">
230           <ComponentInstance name="Monitor Master"
231             ref="imon-master" />
232           <ComponentInstance name="Monitor PSSW" ref="imon-ssw" />
233           <ComponentInstance name="Monitor APEX"
234             ref="imon-apex" />
235         </Group>
236         <Group name="iMX6 Serial User Level Driver">
237           <Description>i.MX Serial Driver</Description>
238           <ComponentInstance name="imx_uart-base"
```

```
239             ref="imx_uart-fp_ext">
240             <ParameterValue name="MAX_FD_COUNT" value="5" />
241             <ParameterValue name="PROVIDER" value="ser0" />
242             <ParameterValue name="USE_CLK_MGR" value="true" />
243             <ParameterValue name="HEAP_SIZE" value="0x00300000"/><
                    AssignedDependencyTable>
244               <AssignedDependency cmp="iMX Clock Manager"
245                 dependId="CLKMNGR" provideId="driver" />
246             </AssignedDependencyTable>
247           </ComponentInstance>
248           <ComponentInstance name="imx_uart-port0"
249             ref="imx_uart-device">
250             <ParameterValue name="IOADDR_LINK" value="UART1" />
251             <ParameterValue name="IRQ_LINK" value="UART1" />
252             <ParameterValue name="CLOCK_SPEED"
253               value="80000000" />
254             <ParameterValue name="CLK_NAME"
255               value="uart_serial_clk_gate" />
256             <ParameterValue name="CLK_FREQ" value="80000000" />
257             <ParameterValue name="FILE_NAME" value="0" />
258             <ParameterValue name="IO_ID" value="0" />
259             <AssignedDependencyTable>
260               <AssignedDependency cmp="imx_uart-base"
261                 dependId="PROVIDER" provideId="driver" />
262               <AssignedDependency cmp="imx_uart-base"
263                 dependId="USE_CLK_MGR" provideId="USE_CLK_MGR" />
264             </AssignedDependencyTable>
265           </ComponentInstance>
266           <ComponentInstance name="imx_uart-port1"
267             ref="imx_uart-device">
268             <ParameterValue name="IOADDR_LINK" value="UART2" />
269             <ParameterValue name="DEVICE" value="1" />
270             <ParameterValue name="IRQ_LINK" value="UART2" />
271             <ParameterValue name="CLOCK_SPEED"
272               value="80000000" />
273             <ParameterValue name="CLK_NAME"
274               value="uart_serial_clk_gate" />
275             <ParameterValue name="CLK_FREQ" value="80000000" />
276             <ParameterValue name="FILE_NAME" value="1" />
277             <ParameterValue name="IO_ID" value="1" />
278             <AssignedDependencyTable>
279               <AssignedDependency cmp="imx_uart-base"
280                 dependId="PROVIDER" provideId="driver" />
281               <AssignedDependency cmp="imx_uart-base"
282                 dependId="USE_CLK_MGR" provideId="USE_CLK_MGR" />
283             </AssignedDependencyTable>
284           </ComponentInstance>
285           <ComponentInstance name="imx_uart-port2"
286             ref="imx_uart-device">
287             <ParameterValue name="IOADDR_LINK" value="UART3" />
288             <ParameterValue name="DEVICE" value="2" />
289             <ParameterValue name="IRQ_LINK" value="UART3" />
290             <ParameterValue name="CLOCK_SPEED"
291               value="80000000" />
292             <ParameterValue name="CLK_NAME"
293               value="uart_serial_clk_gate" />
```

```
294             <ParameterValue name="CLK_FREQ" value="80000000" />
295             <ParameterValue name="FILE_NAME" value="2" />
296             <ParameterValue name="IO_ID" value="2" />
297             <AssignedDependencyTable>
298               <AssignedDependency cmp="imx_uart-base"
299                 dependId="PROVIDER" provideId="driver" />
300               <AssignedDependency cmp="imx_uart-base"
301                 dependId="USE_CLK_MGR" provideId="USE_CLK_MGR" />
302             </AssignedDependencyTable>
303           </ComponentInstance>
304           <ComponentInstance name="imx_uart-port3"
305             ref="imx_uart-device">
306             <ParameterValue name="IOADDR_LINK" value="UART4" />
307             <ParameterValue name="DEVICE" value="3" />
308             <ParameterValue name="IRQ_LINK" value="UART4" />
309             <ParameterValue name="CLOCK_SPEED"
310               value="80000000" />
311             <ParameterValue name="CLK_NAME"
312               value="uart_serial_clk_gate" />
313             <ParameterValue name="CLK_FREQ" value="80000000" />
314             <ParameterValue name="FILE_NAME" value="3" />
315             <ParameterValue name="IO_ID" value="3" />
316             <AssignedDependencyTable>
317               <AssignedDependency cmp="imx_uart-base"
318                 dependId="PROVIDER" provideId="driver" />
319               <AssignedDependency cmp="imx_uart-base"
320                 dependId="USE_CLK_MGR" provideId="USE_CLK_MGR" />
321             </AssignedDependencyTable>
322           </ComponentInstance>
323           <ComponentInstance name="imx_uart-port4"
324             ref="imx_uart-device">
325             <ParameterValue name="IOADDR_LINK" value="UART5" />
326             <ParameterValue name="DEVICE" value="4" />
327             <ParameterValue name="IRQ_LINK" value="UART5" />
328             <ParameterValue name="CLOCK_SPEED"
329               value="80000000" />
330             <ParameterValue name="CLK_NAME"
331               value="uart_serial_clk_gate" />
332             <ParameterValue name="CLK_FREQ" value="80000000" />
333             <ParameterValue name="FILE_NAME" value="4" />
334             <ParameterValue name="IO_ID" value="4" />
335             <AssignedDependencyTable>
336               <AssignedDependency cmp="imx_uart-base"
337                 dependId="PROVIDER" provideId="driver" />
338               <AssignedDependency cmp="imx_uart-base"
339                 dependId="USE_CLK_MGR" provideId="USE_CLK_MGR" />
340             </AssignedDependencyTable>
341           </ComponentInstance>
342           </Group>
343         <Group name="iMX6_FEC Ethernet User Level Driver">
344           <ComponentInstance name="imx_fec-base"
345             ref="imx_fec-fp_ext">
346             <ParameterValue name="MAX_FD_COUNT" value="6" />
347             <ParameterValue name="PROVIDER" value="eth0" />
348             <ParameterValue name="MAX_TRANSFER_SIZE"
349               value="1522" />
```

```
350             <ParameterValue name="MAX_FILE_COUNT" value="6" />
351             <ParameterValue name="USE_CLK_MGR" value="true" />
352             <AssignedDependencyTable>
353               <AssignedDependency cmp="iMX Clock Manager"
354                 dependId="CLKMNGR" provideId="driver" />
355             </AssignedDependencyTable>
356           </ComponentInstance>
357           <ComponentInstance name="imx_fec-device"
358             ref="imx_fec-device">
359             <ParameterValue name="USE_PSP_NODE" value="false" />
360             <ParameterValue name="IO_ADDR" value="0x02188000" />
361             <ParameterValue name="IO_OFFSET" value="0" />
362             <ParameterValue name="IO_IRQ" value="150" />
363             <ParameterValue name="CLK_FREQ" value="66000000" />
364             <ParameterValue name="CLK_NAME"
365               value="enet_clk_gate" />
366             <ParameterValue name="FILE_NAME" value="dev0" />
367             <ParameterValue name="IO_SIZE" value="0x400" />
368             <AssignedDependencyTable>
369               <AssignedDependency cmp="imx_fec-base"
370                 dependId="USE_CLK_MGR" provideId="USE_CLK_MGR" />
371               <AssignedDependency cmp="imx_fec-base"
372                 dependId="PROVIDER" provideId="driver" />
373             </AssignedDependencyTable>
374           </ComponentInstance>
375           <ComponentInstance name="imx_fec-vchan0"
376             ref="hlnet-vchan">
377             <ParameterValue name="VCHAN" value="0" />
378             <ParameterValue name="FILE_NAME" value="0" />
379             <AssignedDependencyTable>
380               <AssignedDependency cmp="imx_fec-device"
381                 dependId="DEVICE" provideId="file" />
382               <AssignedDependency cmp="imx_fec-base"
383                 dependId="PROVIDER" provideId="driver" />
384             </AssignedDependencyTable>
385           </ComponentInstance>
386           <ComponentInstance name="imx_fec-vchan1"
387             ref="hlnet-vchan">
388             <ParameterValue name="VCHAN" value="1" />
389             <ParameterValue name="FILE_NAME" value="1" />
390             <AssignedDependencyTable>
391               <AssignedDependency cmp="imx_fec-device"
392                 dependId="DEVICE" provideId="file" />
393               <AssignedDependency cmp="imx_fec-base"
394                 dependId="PROVIDER" provideId="driver" />
395             </AssignedDependencyTable>
396           </ComponentInstance>
397           <ComponentInstance name="imx_fec-vchan2"
398             ref="hlnet-vchan">
399             <ParameterValue name="VCHAN" value="2" />
400             <ParameterValue name="FILE_NAME" value="2" />
401             <AssignedDependencyTable>
402               <AssignedDependency cmp="imx_fec-device"
403                 dependId="DEVICE" provideId="file" />
404               <AssignedDependency cmp="imx_fec-base"
405                 dependId="PROVIDER" provideId="driver" />
```

```
406             </AssignedDependencyTable>
407           </ComponentInstance>
408           <ComponentInstance name="imx_fec-vchan3"
409             ref="hlnet-vchan">
410             <ParameterValue name="VCHAN" value="3" />
411             <ParameterValue name="FILE_NAME" value="3" />
412             <AssignedDependencyTable>
413               <AssignedDependency cmp="imx_fec-device"
414                 dependId="DEVICE" provideId="file" />
415               <AssignedDependency cmp="imx_fec-base"
416                 dependId="PROVIDER" provideId="driver" />
417             </AssignedDependencyTable>
418           </ComponentInstance>
419           <ComponentInstance name="imx_fec-vchan4"
420             ref="hlnet-vchan">
421             <ParameterValue name="VCHAN" value="4" />
422             <ParameterValue name="FILE_NAME" value="4" />
423             <AssignedDependencyTable>
424               <AssignedDependency cmp="imx_fec-device"
425                 dependId="DEVICE" provideId="file" />
426               <AssignedDependency cmp="imx_fec-base"
427                 dependId="PROVIDER" provideId="driver" />
428             </AssignedDependencyTable>
429           </ComponentInstance></Group>
430         <ComponentInstance name="PikeOS Kernel"
431           ref="PikeOS Kernel">
432           <ParameterValue name="PIKEOS_KERNEL_DIR"
433             value="PIKEOS_POOL/object/bsp/imx6" />
434         </ComponentInstance>
435         <ComponentInstance name="System Software"
436           ref="generic-pssw">
437           <ParameterValue name="PIKEOS_PSSW_BIN"
438             value="PIKEOS_POOL/pssw/object/standard/pssw.elf" />
439         </ComponentInstance>
440         <ComponentInstance name="sabrelite-config"
441           ref="sabrelite-config" />
442         <ComponentInstance name="imx6.psp" ref="imx6.psp">
443           <ParameterValue name="PSP_CONSOLE_PORT" value="2" />
444           <ParameterValue name="GPU_VOLT" value="false" />
445         </ComponentInstance>
446         <ComponentInstance name="PCI Manager"
447           ref="PCI Manager KDEV" />
448         <ComponentInstance name="HM Event Subscription"
449           ref="hmev" />
450         <ComponentInstance name="Standard Console"
451           ref="Standard Console" />
452         <ComponentInstance name="iMX Clock Manager"
453           ref="i.MX Clock Manager" />
454         <Info>
455           <Cpu name="i.MX6DL, i.MX6Q" />
456           <Vendor data="Boundary Devices" />
457           <Platman
458             manual="documentation/platform/platform-manual-ARM.pdf" />
459           <Uri
460             link="http://boundarydevices.com/products/sabre-lite-imx6-sbc
                  " />
```

```
461           <Project label="PSSW Fusion Project"
462             profile="fusion-pssw" template="standard" variable="
                  FUSION_PSSW" />
463           <Project label="Kernel Fusion Project"
464             profile="fusion-kernel" template="imx6" variable="
                  FUSION_KERNEL" />
465           <Project label="PSP Project" profile="psp"
466             template="imx6" variable="PSP" />
467         </Info>
468       </Bsp>
469       <Group name="Application2">
470         <ComponentInstance name="POSIX Partition2"
471           ref="POSIX Partition">
472           <ParameterValue name="PARTNAME" value="Control" />
473           <ParameterValue name="PARTID" value="3" />
474           <VmitConfigurationTable>
475             <VmitConfiguration condition="true"
476               isReference="true">
477               <Partition
478                 Abilities="VM_AB_CACHE_CHANGE VM_AB_TRACE VM_AB_PSP_RESET
                      VM_AB_PSP_CONSOLE VM_AB_HM_INJECT_OTHER
                     VM_AB_MEM_CREATE" CpuMask="-1"
479                 Identifier="$(PARTID)" MaxChildTaskCount="1" MaxFDCount="
                    20"
480                 MaxPrio="62" MultiPartitionHMTableID="0" Name="$(PARTNAME
                    )"
481                 SchedChangeAction="VM_SCHED_CHANGE_IGNORE"
482                 StartupMode="VM_PART_MODE_COLD_START" TimePartitionID="0"
                    >
483                 <FileAccessTable>
484                   <FileAccess
485                     AccessMode="VM_O_RD VM_O_MAP" FileName="rfs:
                        ov_server_manager2.conf" />
486                   <FileAccess FileName="shm:MANAGER_SHAREDMEMORY"
487                     AccessMode="VM_O_RD VM_O_WR VM_O_MAP">
488                   </FileAccess>
489                   <ComponentReference
490                     ref="POSIX Process2" />
491                 </FileAccessTable>
492                 <MemoryRequirementTable>
493                   <ComponentReference
494                     ref="POSIX Process2" />
495                 </MemoryRequirementTable>
496                 <ProcessTable>
497                   <ComponentReference
498                     ref="POSIX Process2" />
499                 </ProcessTable>
500                 <QueuingPortTable>
501                   <QueuingPort Name="MANAGER_OUT"
502                     MaxMessageCount="4294900000" MaxMessageSize="
                        4294900000" Direction="VM_PORT_SOURCE">
503                   </QueuingPort>
504                   <ComponentReference
505                     ref="POSIX Process2" />
506                 </QueuingPortTable>
507                 <SamplingPortTable>
```

```
508                 <ComponentReference
509                   ref="POSIX Process2" />
510               </SamplingPortTable>
511               <HMTable>
512                 <DefaultSwitch>
513                   <Default Action="P4_HM_PAC_IDLE"
514                     Code="0" Level="P4_HM_LEVEL_USER" Notify="0" />
515                 </DefaultSwitch>
516               </HMTable>
517             </Partition>
518           </VmitConfiguration>
519         </VmitConfigurationTable>
520       </ComponentInstance>
521       <ComponentInstance name="POSIX Process2"
522         ref="POSIX Process">
523         <ParameterValue name="LWIP_CONFIG" value="true" />
524         <ParameterValue name="LWIP_TARGETIP_IF1"
525           value="134.130.125.61" />
526         <ParameterValue name="RAMSIZE" value="0x04000000" />
527         <ParameterValue name="FILE"
528           value="CUSTOM_POOL/ov_runtimeserver" />
529         <ParameterValue name="MUXA_ALL" value="true" />
530         <ParameterValue name="LWIP_GATEWAY_IF1"
531           value="192.168.0.11" />
532         <ParameterValue name="DESTNAME" value="Control" />
533         <AssignedDependencyTable>
534
535
536
537           <AssignedDependency cmp="imx_fec-vchan2"
538             provideId="channel" dependId="LWIP_DEVICE_IF1" />
539           <AssignedDependency cmp="muxa"
540             provideId="CHANNEL_03" dependId="IOFILE" />
541         </AssignedDependencyTable>
542       </ComponentInstance>
543     </Group>
544     <Group name="Application3">
545       <ComponentInstance name="POSIX Partition3"
546         ref="POSIX Partition">
547         <ParameterValue name="PARTNAME" value="Mand0" />
548         <ParameterValue name="PARTID" value="4" />
549         <VmitConfigurationTable>
550           <VmitConfiguration condition="true"
551             isReference="true">
552             <Partition
553               Abilities="VM_AB_CACHE_CHANGE VM_AB_TRACE" CpuMask="-1"
554               Identifier="$(PARTID)" MaxChildTaskCount="1" MaxFDCount="
                    20"
555               MaxPrio="62" MultiPartitionHMTableID="0" Name="$(PARTNAME
                    )"
556               SchedChangeAction="VM_SCHED_CHANGE_IGNORE"
557               StartupMode="VM_PART_MODE_COLD_START" TimePartitionID="0"
                    >
558               <FileAccessTable>
559                 <FileAccess
560                   AccessMode="VM_O_RD VM_O_MAP"
```

```
561                     FileName="rfs:ov_server_manager3.conf" />
562                   <FileAccess FileName="shm:INTERFACE_SHAREDMEMORY"
563                     AccessMode="VM_O_RD VM_O_WR">
564                   </FileAccess>
565                   <ComponentReference
566                     ref="POSIX Process3" />
567                 </FileAccessTable>
568                 <MemoryRequirementTable>
569                   <ComponentReference
570                     ref="POSIX Process3" />
571                 </MemoryRequirementTable>
572                 <ProcessTable>
573                   <ComponentReference
574                     ref="POSIX Process3" />
575                 </ProcessTable>
576                 <QueuingPortTable>
577                   <QueuingPort Name="MANAGER_IN"
578                     MaxMessageCount="4294900000" MaxMessageSize="
                          4294900000" Direction="VM_PORT_DESTINATION">
579                   </QueuingPort>
580                   <ComponentReference
581                     ref="POSIX Process3" />
582                 </QueuingPortTable>
583                 <SamplingPortTable>
584                   <ComponentReference
585                     ref="POSIX Process3" />
586                 </SamplingPortTable>
587                 <HMTable>
588                   <DefaultSwitch>
589                     <Default Action="P4_HM_PAC_IDLE"
590                       Code="0" Level="P4_HM_LEVEL_USER" Notify="0" />
591                   </DefaultSwitch>
592                 </HMTable>
593               </Partition>
594             </VmitConfiguration>
595           </VmitConfigurationTable>
596         </ComponentInstance>
597         <ComponentInstance name="POSIX Process3"
598           ref="POSIX Process">
599           <ParameterValue name="LWIP_CONFIG" value="true" />
600           <ParameterValue name="LWIP_TARGETIP_IF1"
601             value="134.130.125.64" />
602           <ParameterValue name="RAMSIZE" value="0x04000000" />
603           <ParameterValue name="FILE"
604             value="CUSTOM_POOL/ov_runtimeserver" />
605           <ParameterValue name="MUXA_ALL" value="true" />
606           <ParameterValue name="LWIP_GATEWAY_IF1"
607             value="192.168.0.11" />
608           <ParameterValue name="DESTNAME" value="Mand0" />
609           <ParameterValue name="POSIX_TUNE" value="false"/><
                AssignedDependencyTable>
610
611
612
613
614
```

```
615             <AssignedDependency cmp="muxa"
616               provideId="CHANNEL_07" dependId="IOFILE" /><
                     AssignedDependency
617               cmp="imx_fec-vchan3" provideId="channel"
618               dependId="LWIP_DEVICE_IF1" /></AssignedDependencyTable>
619           </ComponentInstance>
620         </Group>
621         <Group name="Application4">
622           <ComponentInstance name="POSIX Partition4"
623             ref="POSIX Partition">
624             <ParameterValue name="PARTNAME" value="Interface" />
625             <ParameterValue name="PARTID" value="5" />
626             <VmitConfigurationTable>
627               <VmitConfiguration condition="true"
628                 isReference="true">
629                 <Partition
630                   Abilities="VM_AB_CACHE_CHANGE VM_AB_TRACE" CpuMask="-1"
631                   Identifier="$(PARTID)" MaxChildTaskCount="1" MaxFDCount="
                        20"
632                   MaxPrio="62" MultiPartitionHMTableID="0" Name="$(PARTNAME
                        )"
633                   SchedChangeAction="VM_SCHED_CHANGE_IGNORE"
634                   StartupMode="VM_PART_MODE_COLD_START" TimePartitionID="0"
                        >
635                   <FileAccessTable>
636                     <FileAccess
637                       AccessMode="VM_O_RD VM_O_MAP"
638                       FileName="rfs:ov_server_manager4.conf" />
639                     <ComponentReference
640                       ref="POSIX Process4" />
641                   </FileAccessTable>
642                   <MemoryRequirementTable>
643                     <ComponentReference
644                       ref="POSIX Process4" />
645                   </MemoryRequirementTable>
646                   <ProcessTable>
647                     <ComponentReference
648                       ref="POSIX Process4" />
649                   </ProcessTable>
650                   <QueuingPortTable>
651                     <QueuingPort Name="MANAGER_IN"
652                       MaxMessageCount="4294900000" MaxMessageSize="
                            4294900000" Direction="VM_PORT_DESTINATION">
653                     </QueuingPort>
654                     <ComponentReference
655                       ref="POSIX Process4" />
656                   </QueuingPortTable>
657                   <SamplingPortTable>
658
659                     <ComponentReference
660                       ref="POSIX Process4" />
661                   </SamplingPortTable>
662                   <HMTable>
663                     <DefaultSwitch>
664                       <Default Action="P4_HM_PAC_IDLE"
665                         Code="0" Level="P4_HM_LEVEL_USER" Notify="0" />
```

```
                </DefaultSwitch>
              </HMTable>
            </Partition>
          </VmitConfiguration>
        </VmitConfigurationTable>
      </ComponentInstance>
      <ComponentInstance name="POSIX Process4"
        ref="POSIX Process">
        <ParameterValue name="LWIP_CONFIG" value="true" />
        <ParameterValue name="LWIP_TARGETIP_IF1"
          value="134.130.125.66" />
        <ParameterValue name="RAMSIZE" value="0x04000000" />
        <ParameterValue name="FILE"
          value="CUSTOM_POOL/ov_runtimeserver" />
        <ParameterValue name="MUXA_ALL" value="true" />
        <ParameterValue name="LWIP_GATEWAY_IF1"
          value="192.168.0.11" />
        <ParameterValue name="DESTNAME" value="Interface" />
        <AssignedDependencyTable>









          <AssignedDependency cmp="imx_fec-vchan4"
            provideId="channel" dependId="LWIP_DEVICE_IF1" /><
              AssignedDependency
            cmp="muxa" provideId="CHANNEL_01" dependId="IOFILE" /></
              AssignedDependencyTable>
      </ComponentInstance>
    </Group></ConfigurationDomainTable>
  <Vmit>
    <!--the master VMIT-->
    <Configuration PartitionID="0"
      Version="VM_VMIT_VERSION_CURRENT">
      <ConnectionTable>
        <PartitionChannelTable>
          <Channel PortType="VM_PORT_QUEUING">
            <DestinationPortRef PortName="MANAGER_IN"
              PartitionID="5">
            </DestinationPortRef>
            <SourcePortRef PortName="INTERFACE_OUT" PartitionID="2"></
              SourcePortRef></Channel>
          <Channel PortType="VM_PORT_QUEUING">
            <DestinationPortRef PortName="CONTROL_IN"
              PartitionID="2">
            </DestinationPortRef>
            <SourcePortRef PortName="MANAGER_OUT" PartitionID="3"></
              SourcePortRef></Channel>
          <Channel PortType="VM_PORT_QUEUING">
            <DestinationPortRef PortName="MANAGER_IN"
              PartitionID="4">
            </DestinationPortRef>
            <SourcePortRef PortName="MandO_OUT" PartitionID="2"></
```

```
                SourcePortRef></Channel>
718             <ComponentReference ref="PikeOS Kernel" />
719             <ComponentReference ref="System Software" />
720             <ComponentReference ref="sabrelite-config" />
721             <ComponentReference ref="imx6.psp" />
722             <ComponentReference ref="PCI Manager" />
723             <ComponentReference ref="Monitor Master" />
724             <ComponentReference ref="Monitor PSSW" />
725             <ComponentReference ref="Monitor APEX" />
726             <ComponentReference ref="HM Event Subscription" />
727             <ComponentReference ref="Standard Console" />
728             <ComponentReference ref="imx_uart-base" />
729             <ComponentReference ref="imx_uart-port0" />
730             <ComponentReference ref="imx_uart-port1" />
731             <ComponentReference ref="imx_uart-port2" />
732             <ComponentReference ref="imx_uart-port3" />
733             <ComponentReference ref="imx_uart-port4" />
734             <ComponentReference ref="imx_fec-base" />
735             <ComponentReference ref="imx_fec-device" />
736             <ComponentReference ref="imx_fec-vchan0" />
737             <ComponentReference ref="imx_fec-vchan1" />
738             <ComponentReference ref="imx_fec-vchan2" />
739             <ComponentReference ref="imx_fec-vchan3" />
740             <ComponentReference ref="iMX Clock Manager" />
741             <ComponentReference ref="POSIX Partition2" />
742             <ComponentReference ref="POSIX Process2" />
743           <ComponentReference ref="POSIX Partition3"/><ComponentReference
                ref="POSIX Process3"/><ComponentReference ref="imx_fec-
              vchan4"/><ComponentReference ref="POSIX Partition4"/><
              ComponentReference ref="POSIX Process4"/></
              PartitionChannelTable>
744           <ExtensionChannelTable>
745             <ComponentReference ref="PikeOS Kernel" />
746             <ComponentReference ref="System Software" />
747             <ComponentReference ref="sabrelite-config" />
748             <ComponentReference ref="imx6.psp" />
749             <ComponentReference ref="PCI Manager" />
750             <ComponentReference ref="Monitor Master" />
751             <ComponentReference ref="Monitor PSSW" />
752             <ComponentReference ref="Monitor APEX" />
753             <ComponentReference ref="HM Event Subscription" />
754             <ComponentReference ref="Standard Console" />
755             <ComponentReference ref="imx_uart-base" />
756             <ComponentReference ref="imx_uart-port0" />
757             <ComponentReference ref="imx_uart-port1" />
758             <ComponentReference ref="imx_uart-port2" />
759             <ComponentReference ref="imx_uart-port3" />
760             <ComponentReference ref="imx_uart-port4" />
761             <ComponentReference ref="imx_fec-base" />
762             <ComponentReference ref="imx_fec-device" />
763             <ComponentReference ref="imx_fec-vchan0" />
764             <ComponentReference ref="imx_fec-vchan1" />
765             <ComponentReference ref="imx_fec-vchan2" />
766             <ComponentReference ref="imx_fec-vchan3" />
767             <ComponentReference ref="iMX Clock Manager" />
768             <ComponentReference ref="POSIX Partition2" />
```

```
769             <ComponentReference ref="POSIX Process2" />
770           <ComponentReference ref="POSIX Partition3"/><ComponentReference
                  ref="POSIX Process3"/><ComponentReference ref="imx_fec-
                vchan4"/><ComponentReference ref="POSIX Partition4"/><
                ComponentReference ref="POSIX Process4"/></
                ExtensionChannelTable>
771           <GateChannelTable>
772             <ComponentReference ref="PikeOS Kernel" />
773             <ComponentReference ref="System Software" />
774             <ComponentReference ref="sabrelite-config" />
775             <ComponentReference ref="imx6.psp" />
776             <ComponentReference ref="PCI Manager" />
777             <ComponentReference ref="Monitor Master" />
778             <ComponentReference ref="Monitor PSSW" />
779             <ComponentReference ref="Monitor APEX" />
780             <ComponentReference ref="HM Event Subscription" />
781             <ComponentReference ref="Standard Console" />
782             <ComponentReference ref="imx_uart-base" />
783             <ComponentReference ref="imx_uart-port0" />
784             <ComponentReference ref="imx_uart-port1" />
785             <ComponentReference ref="imx_uart-port2" />
786             <ComponentReference ref="imx_uart-port3" />
787             <ComponentReference ref="imx_uart-port4" />
788             <ComponentReference ref="imx_fec-base" />
789             <ComponentReference ref="imx_fec-device" />
790             <ComponentReference ref="imx_fec-vchan0" />
791             <ComponentReference ref="imx_fec-vchan1" />
792             <ComponentReference ref="imx_fec-vchan2" />
793             <ComponentReference ref="imx_fec-vchan3" />
794             <ComponentReference ref="iMX Clock Manager" />
795             <ComponentReference ref="POSIX Partition2" />
796             <ComponentReference ref="POSIX Process2" />
797           <ComponentReference ref="POSIX Partition3"/><ComponentReference
                  ref="POSIX Process3"/><ComponentReference ref="imx_fec-
                vchan4"/><ComponentReference ref="POSIX Partition4"/><
                ComponentReference ref="POSIX Process4"/></GateChannelTable
                >
798         </ConnectionTable>
799         <PartitionTable>
800           <ComponentReference ref="POSIX Partition" />
801           <ComponentReference ref="service.partition" />
802           <ComponentReference ref="POSIX Partition2" />
803           <ComponentReference ref="POSIX Partition3" /><
                ComponentReference
804             ref="POSIX Partition4" /></PartitionTable>
805         <ScheduleTable>
806           <ScheduleScheme Name="Scheme1">
807             <WindowTable>
808               <Window Identifier="1" Start="0" Duration="60"
809                 TimePartitionID="1" Flags="VM_SCF_PERIOD">
810               </Window>
811               <Window Identifier="2" Start="60" Duration="20"
812                 TimePartitionID="2" Flags="VM_SCF_PERIOD">
813               </Window>
814               <Window Identifier="3" Start="80" Duration="20"
815                 TimePartitionID="3" Flags="VM_SCF_PERIOD">
```

```
816                </Window>
817                <Window Identifier="4" Start="100" Duration="20"
818                  TimePartitionID="4" Flags="VM_SCF_PERIOD">
819                </Window></WindowTable></ScheduleScheme>
820              <ScheduleScheme Name="Scheme2">
821                <WindowTable>
822                  <Window Identifier="1" Start="40" Duration="30"
823                    TimePartitionID="1" Flags="VM_SCF_PERIOD">
824                  </Window>
825                  <Window Identifier="2" Start="70" Duration="40"
826                    TimePartitionID="2" Flags="VM_SCF_PERIOD">
827                  </Window>
828                  <Window Identifier="3" Start="0" Duration="40"
829                    TimePartitionID="3" Flags="VM_SCF_PERIOD">
830                  </Window>
831                  <Window Identifier="4" Start="110" Duration="10"
832                    TimePartitionID="1" Flags="VM_SCF_PERIOD">
833                  </Window></WindowTable></ScheduleScheme>
834              <ComponentReference ref="PikeOS Kernel" />
835              <ComponentReference ref="System Software" />
836              <ComponentReference ref="sabrelite-config" />
837              <ComponentReference ref="imx6.psp" />
838              <ComponentReference ref="PCI Manager" />
839              <ComponentReference ref="Monitor Master" />
840              <ComponentReference ref="Monitor PSSW" />
841              <ComponentReference ref="Monitor APEX" />
842              <ComponentReference ref="HM Event Subscription" />
843              <ComponentReference ref="Standard Console" />
844              <ComponentReference ref="imx_uart-base" />
845              <ComponentReference ref="imx_uart-port0" />
846              <ComponentReference ref="imx_uart-port1" />
847              <ComponentReference ref="imx_uart-port2" />
848              <ComponentReference ref="imx_uart-port3" />
849              <ComponentReference ref="imx_uart-port4" />
850              <ComponentReference ref="imx_fec-base" />
851              <ComponentReference ref="imx_fec-device" />
852              <ComponentReference ref="imx_fec-vchan0" />
853              <ComponentReference ref="imx_fec-vchan1" />
854              <ComponentReference ref="imx_fec-vchan2" />
855              <ComponentReference ref="imx_fec-vchan3" />
856              <ComponentReference ref="iMX Clock Manager" />
857              <ComponentReference ref="POSIX Partition2" />
858              <ComponentReference ref="POSIX Process2" />
859            <ComponentReference ref="POSIX Partition3"/><ComponentReference
                 ref="POSIX Process3"/><ComponentReference ref="imx_fec-vchan4
                 "/><ComponentReference ref="POSIX Partition4"/><
                 ComponentReference ref="POSIX Process4"/></ScheduleTable>
860            <SharedMemoryTable>
861              <MemoryRequirement Name="MANAGER_SHAREDMEMORY" Size="0x00001000
                   "
862                Alignment="-1" PhysicalAddress="-1" Contiguous="false"
863                IsPool="false" Type="VM_MEM_TYPE_IO_MEM" CacheMode="
                     VM_MEM_CACHE_CB"
864                AccessMode="VM_MEM_ACCESS_RD VM_MEM_ACCESS_WR
                     VM_MEM_ACCESS_EXEC">
865              </MemoryRequirement>
```

```
866           <MemoryRequirement Name="MandO_SHAREDMEMORY" Size="0x00001000"
867             Alignment="-1" PhysicalAddress="-1" Contiguous="false"
868             IsPool="false" Type="VM_MEM_TYPE_IO_MEM" CacheMode="
                    VM_MEM_CACHE_CB"
869             AccessMode="VM_MEM_ACCESS_RD VM_MEM_ACCESS_WR
                    VM_MEM_ACCESS_EXEC">
870           </MemoryRequirement>
871           <MemoryRequirement Name="INTERFACE_SHAREDMEMORY" Size="0
                  x00001000"
872             Alignment="-1" PhysicalAddress="-1" Contiguous="false"
873             IsPool="false" Type="VM_MEM_TYPE_IO_MEM" CacheMode="
                    VM_MEM_CACHE_CB"
874             AccessMode="VM_MEM_ACCESS_RD VM_MEM_ACCESS_WR
                    VM_MEM_ACCESS_EXEC">
875           </MemoryRequirement>
876           <ComponentReference ref="POSIX Process" />
877           <ComponentReference ref="PikeOS Kernel" />
878           <ComponentReference ref="System Software" />
879           <ComponentReference ref="sabrelite-config" />
880           <ComponentReference ref="imx6.psp" />
881           <ComponentReference ref="PCI Manager" />
882           <ComponentReference ref="Monitor Master" />
883           <ComponentReference ref="Monitor PSSW" />
884           <ComponentReference ref="Monitor APEX" />
885           <ComponentReference ref="HM Event Subscription" />
886           <ComponentReference ref="Standard Console" />
887           <ComponentReference ref="imx_uart-base" />
888           <ComponentReference ref="imx_uart-port0" />
889           <ComponentReference ref="imx_uart-port1" />
890           <ComponentReference ref="imx_uart-port2" />
891           <ComponentReference ref="imx_uart-port3" />
892           <ComponentReference ref="imx_uart-port4" />
893           <ComponentReference ref="imx_fec-base" />
894           <ComponentReference ref="imx_fec-device" />
895           <ComponentReference ref="imx_fec-vchan0" />
896           <ComponentReference ref="imx_fec-vchan1" />
897           <ComponentReference ref="imx_fec-vchan2" />
898           <ComponentReference ref="imx_fec-vchan3" />
899           <ComponentReference ref="iMX Clock Manager" />
900           <ComponentReference ref="POSIX Partition2" />
901           <ComponentReference ref="POSIX Process2" />
902         <ComponentReference ref="POSIX Partition3"/><ComponentReference
              ref="POSIX Process3"/><ComponentReference ref="imx_fec-vchan4
              "/><ComponentReference ref="POSIX Partition4"/><
              ComponentReference ref="POSIX Process4"/></SharedMemoryTable>
903         <SystemExtensionTable>
904           <FileProviderTable />
905           <GateProviderTable>
906             <ComponentReference ref="PCI Manager" />
907             <ComponentReference ref="Monitor Master" />
908             <ComponentReference ref="Monitor PSSW" />
909             <ComponentReference ref="Monitor APEX" />
910             <ComponentReference ref="HM Event Subscription" />
911             <ComponentReference ref="Standard Console" />
912             <ComponentReference ref="iMX Clock Manager" />
913           </GateProviderTable>
```

```
914           <PortProviderTable />
915         </SystemExtensionTable>
916         <ModuleHMTable>
917           <DefaultSwitch>
918             <Default Action="P4_HM_MAC_SHUTDOWN" Notify="0" />
919           </DefaultSwitch>
920         </ModuleHMTable>
921         <MultiPartitionHMTable>
922           <Table Identifier="0" Name="Default">
923             <DefaultSwitch>
924               <Default Action="P4_HM_MAC_SHUTDOWN"
925                 Level="P4_HM_LEVEL_PARTITION" Notify="0" />
926             </DefaultSwitch>
927           </Table>
928         </MultiPartitionHMTable>
929       </Configuration>
930     </Vmit>
931
932     <Romimage>
933       <properties>
934         <prop_dir name="app/Manager/Manager">
935           <prop_dir name="args">
936             <prop_string name="argv1" data="-c" />
937             <prop_string name="argv2"
938               data="/rfs/ov_server_manager.conf" />
939             <prop_uint32 name="numargs" data="2" />
940           </prop_dir>
941         </prop_dir>
942         <prop_dir name="app/Control/Control">
943
944           <prop_dir name="args">
945             <prop_string name="argv1" data="-c" />
946             <prop_string name="argv2"
947               data="/rfs/ov_server_manager2.conf" />
948             <prop_uint32 name="numargs" data="2" />
949           </prop_dir>
950         </prop_dir>
951
952         <prop_dir name="app/Interface/Interface">
953           <prop_dir name="args">
954             <prop_string name="argv1" data="-c" />
955             <prop_string name="argv2"
956               data="/rfs/ov_server_manager4.conf" />
957             <prop_uint32 name="numargs" data="2" />
958           </prop_dir>
959         </prop_dir>
960         <prop_dir name="app/Mand0/Mand0">
961           <prop_dir name="args">
962
963             <prop_string name="argv1" data="-c" />
964             <prop_string name="argv2" data="/rfs/ov_server_manager3.conf"
                  /><prop_uint32
965               name="numargs" data="2" /></prop_dir></prop_dir></
                  properties>
966       <files>
967
```

```
968       <file name="ov_server_manager.conf" resource="/cygdrive/f/SYSGO/
              POSIX4/POOL/ov_server_manager.conf" /><file
969         name="ov_server_manager2.conf" resource="/cygdrive/f/SYSGO/POSIX4
                /POOL/ov_server_manager2.conf" /><file
970           name="ov_server_manager2.conf" resource="/cygdrive/f/SYSGO/
                POSIX4/POOL/ov_server_manager2.conf" /><file
971           name="ov_server_manager3.conf" resource="/cygdrive/f/SYSGO/
                POSIX4/POOL/ov_server_manager.conf" /><file
972           name="ov_server_manager4.conf" resource="/cygdrive/f/SYSGO/
                POSIX4/POOL/ov_server_manager.conf" /></files></Romimage>
973     <DefinitionTable>
974       <Definition filename="build/fusion-integration-parameters.cmp"
975         name="Compilation Parameters" path_id="PIKEOS_POOL" />
976       <Definition filename="partition/service_partition.cmp"
977         name="service.partition" path_id="PIKEOS_POOL" />
978       <Definition filename="driver/misc/monitor.cmp" name="monitor"
979         path_id="PIKEOS_POOL" />
980       <Definition filename="driver/misc/traceserver.cmp"
981         name="traceserver" path_id="PIKEOS_POOL" />
982       <Definition filename="driver/misc/muxa.cmp" name="muxa"
983         path_id="PIKEOS_POOL" />
984       <Definition filename="posix/posix_partition_default.cmp"
985         name="POSIX Partition" path_id="PIKEOS_POOL" />
986       <Definition filename="posix/posix_process_default.cmp"
987         name="POSIX Process" path_id="PIKEOS_POOL" />
988       <Definition filename="kernel/kernel.cmp" name="PikeOS Kernel"
989         path_id="PIKEOS_POOL" />
990       <Definition filename="pssw/pssw.cmp" name="generic-pssw"
991         path_id="PIKEOS_POOL" />
992       <Definition filename="board/imx6/imx6q_sabrelite-config.cmp"
993         name="sabrelite-config" path_id="PIKEOS_POOL" />
994       <Definition filename="psp/imx6.psp.cmp" name="imx6.psp"
995         path_id="PIKEOS_POOL" />
996       <Definition filename="kerneldriver/pci_manager.cmp"
997         name="PCI Manager KDEV" path_id="PIKEOS_POOL" />
998       <Definition filename="kerneldriver/imon/imon-master.cmp"
999         name="imon-master" path_id="PIKEOS_POOL" />
1000      <Definition filename="kerneldriver/imon/imon-ssw.cmp"
1001        name="imon-ssw" path_id="PIKEOS_POOL" />
1002      <Definition filename="kerneldriver/imon/imon-apex.cmp"
1003        name="imon-apex" path_id="PIKEOS_POOL" />
1004      <Definition filename="kerneldriver/hmev.cmp" name="hmev"
1005        path_id="PIKEOS_POOL" />
1006      <Definition filename="kerneldriver/stdcon.cmp"
1007        name="Standard Console" path_id="PIKEOS_POOL" />
1008      <Definition
1009        filename="driver/serial/imx_uart/imx_uart-fp_ext.cmp"
1010        name="imx_uart-fp_ext" path_id="PIKEOS_POOL" />
1011      <Definition
1012        filename="driver/serial/imx_uart/imx_uart-device.cmp"
1013        name="imx_uart-device" path_id="PIKEOS_POOL" />
1014      <Definition
1015        filename="driver/ethernet/imx_fec/imx_fec-fp_ext.cmp"
1016        name="imx_fec-fp_ext" path_id="PIKEOS_POOL" />
1017      <Definition
1018        filename="driver/ethernet/imx_fec/imx_fec-device.cmp"
```

```
1019          name="imx_fec-device" path_id="PIKEOS_POOL" />
1020        <Definition filename="driver/config/hlnet/hlnet-vchan.cmp"
1021          name="hlnet-vchan" path_id="PIKEOS_POOL" />
1022        <Definition
1023          filename="driver/clock/imx_clk/imx_clk-prov_kdev.cmp"
1024          name="i.MX Clock Manager" path_id="PIKEOS_POOL" />
1025      </DefinitionTable>
1026    </Integration>
1027  </Project>
```

Nach der Konfiguration der Partitionen wird eine Binary erzeugt, die über Ethernet auf das Board übertragen wird.

```
=> dhcp 134.130.125.53:Control-Software
DHCP client bound to address 134.130.125.64 (1007 ms)
Using FEC device
TFTP from server 134.130.125.53; our IP address is 134.130.125.64
Filename 'test'.
Load address: 0x12000000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         ##################################################################
         #########
         46.9 KiB/s
done
Bytes transferred = 10615003 (a1f8db hex)
=> bootm
## Booting kernel from Legacy Image at 12000000 ...
   Image Name:   PikeOS Boot Image
   Image Type:   ARM Linux Kernel Image (gzip compressed)
   Data Size:    10614939 Bytes = 10.1 MiB
   Load Address: 10020000
   Entry Point:  10020000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK

Starting kernel ...

P4 kernel will start ...
PikeOS (C) Copyright SYSGO AG, Germany
ROM image build: devel-@Achernar-090821-09:24
Kernel build: 4.2-1784, type:  noassert tracesys smp v7
ASP: "arm_v7hf" ARM v7, endian: little, VFP: d0-31
PSP build: 4.2-246
PSP: "imx6x" Freescale i.MX 6 (SMP - L2C)
Features: RETAIL TRACER-SYSCALL OPT SMP(4/32)
Configuration limits:
  respart:    63
  task:       256
```

```
  thread:     511
  timepart:   63
  priority:   256
  kprio:      32
  interrupts: 1024
  TP windows: 256
  thr sstack: 4096 B
Resource partition 0 kernel memory refill strategy: dynamic (on demand)
Time stamp counter clock: 1000000 kHz, via system call
System ticker: periodic mode, resolution 10000000 ns
Time partition switch: 10000000 ns, watchdog timeout: 10000000 ns
Time partition synchronization: default
Free memory: 1011256 KiB
PikeOS PCI Manager KDEV, Build: 4.2-186
PCIMGR: message: PSP returned empty PCI device list
PSSW +Ext. FPs +Messages (Production), Build: 4.2-3668
PIKEOS_MON: Started, version: 4.2-325
Trace Server: version: 4.2-25
imx_uart: Provider "ser0" started, Build: 4.2-87 Production
MUXA: Version: 4.2-289
<DRV INFO> eth0: fec_mii_info: FEC: PHY identify @ 0x7 = 0x00221611
<DRV INFO> eth0: Registered MAC address(02:70:34:03:a9:88) for channel '4'
<DRV INFO> eth0: Registered MAC address(06:70:34:03:a9:88) for channel '3'
<DRV INFO> eth0: Registered MAC address(0a:70:34:03:a9:88) for channel '2'
<DRV INFO> eth0: Registered MAC address(0e:70:34:03:a9:88) for channel '1'
<DRV INFO> eth0: Registered MAC address(12:70:34:03:a9:88) for channel '0'
imx_fec: Provider "eth0" started, Build: 4.2-82 Production
```

Literatur

1. ALBRECHT, Harald. *On meta-modeling for communication in operational process control engineering: Zugl.: Aachen, Techn. Univ., Diss., 2002*. Als Ms. gedr. Düsseldorf: VDI-Verl., 2003. Fortschritt-Berichte VDI Reihe 8, Meß-, Steuerungs- und Regelungstechnik. ISBN 3183975084.
2. ANCA APOSTU; FLORINA PUICAN; GEANINA ULARU; GEORGE SUCIU; GYORGY. *Study on advantages and disadvantages of Cloud Computing – the advantages of Telemetry Applications in the Cloud*. [o.D.]. Nr. 978-1-61804-179-1. Auch verfügbar unter: `https://pdfs.semanticscholar.org/ada5/876e216130cdd7ad6e44539849049dd2de39.pdf`.
3. ARMAND, Francois; GIEN, Michel. *A Practical Look at Micro-Kernels and Virtual Machine Monitors*. 2009. Abger. unter DOI: `10.1109/CCNC.2009.4784874`.
4. ARMOUSH, A.; FRANKE, D.; KALKOV, I.; KOWALEWSKI, S. An Approach for Using Mobile Devices in Industrial Safety-Critical Embedded Systems. *In: Memmi G., Blanke U. (eds) Mobile Computing, Applications, and Services. MobiCASE 2013. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. 2014. Abger. unter DOI: `10.1109/ETFA.2016.7733669`.
5. ASHTARI TALKHESTANI, Behrang; JUNG, Tobias; LINDEMANN, Benjamin; SAHLAB, Nada; JAZDI, Nasser; SCHLOEGL, Wolfgang; WEYRICH, Michael. *An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System*. 2019. Abger. unter DOI: `10.1515/auto-2019-0039`.
6. ATTELE, Kapila Rohan; KUMAR, Amit; SANKAR, V.; RAO, N. V.; SARMA, T. Hitendra (Hrsg.). *Emerging Trends in Electrical, Communications and Information Technologies: Proceedings of ICECIT-2015*. Singapore und s.l.: Springer Singapore, 2017. Lecture Notes in Electrical Engineering. ISBN 978-981-10-1538-0. Abger. unter DOI: `10.1007/978-981-10-1540-3`.
7. *AUTOSAR, Specifications (Release 4.2)*. [o.D.].
8. AZARMIPOUR, Mahyar; ELFAHAM, Haitham; GRIES, Caspar; EPPLE, Ulrich. PLC 4.0: A Control System for Industry 4.0. In: *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*. 2019, Bd. 1, S. 5513–5518. Abger. unter DOI: `10.1109/IECON.2019.8927026`.
9. AZARMIPOUR, Mahyar; ELFAHAM, Haitham; GRIES, Caspar; KLEINERT, Tobias; EPPLE, Ulrich. A Service-based Architecture for the Interaction of Control and MES Systems in Industry 4.0 Environment. In: *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*. 2020, Bd. 1, S. 217–222. Abger. unter DOI: `10.1109/INDIN45582.2020.9442083`.

10. AZARMIPOUR, Mahyar; ELFAHAM, Haitham; GROTHOFF, Julian; TROTHA, Christian von; GRIES, Caspar; EPPLE, Ulrich. Dynamic Resource Management for Virtualization in Industrial Automation. In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*. 2018, S. 2878–2883. Abger. unter DOI: 10.1109/IECON.2018.8591622.

11. AZARMIPOUR, Mahyar; TROTHA, Christian von; GRIES, Caspar; KLEINERT, Tobias; EPPLE, Ulrich. A Secure Gateway for the Cooperation of Information Technologies and Industrial Automation Systems. In: *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*. 2020, S. 53–58. Abger. unter DOI: 10.1109/IECON43393.2020.9254634.

12. BACIC, M. On hardware-in-the-loop simulation. 2005, S. 3194–3198. Abger. unter DOI: 10.1109/CDC.2005.1582653.

13. BARHAM, Paul; DRAGOVIC, Boris; FRASER, Keir; HAND, Steven; HARRIS, Tim; HO, Alex; NEUGEBAUER, Rolf; PRATT, Ian; WARFIELD, Andrew. *Xen and the Art of Virtualization*. 2003.

14. BARRETT, Diane; KIPPER, Gregory. *Virtualization and Forensics*. 1 - How Virtualization Happens. Hrsg. von BARRETT, Diane; KIPPER, Gregory. Boston: Syngress, 2010. ISBN 978-1-59749-557-8. Abger. unter DOI: https://doi.org/10.1016/B978-1-59749-557-8.00001-1.

15. BARTODZIEJ, Christoph Jan. *The concept Industry 4.0*. BestMasters. Springer Gabler, 2017. ISBN 978-3-658-16501-7.

16. BASHARI RAD, Babak; BHATTI, Harrison; AHMADI, Mohammad. An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*. 2017, Jg. 173, S. 8.

17. BOSCHERT, Stefan; ROSEN, Roland. Digital Twin—The Simulation Aspect. 2016. Abger. unter DOI: https://doi.org/10.1007/978-3-319-32156-1_5.

18. BOSS, B.; BADER, S.; A., Orzelski; HOFFMEISTER, M.; HOMPEL, M. ten; VOGEL-HEUSER, Birgit; BAUERNHAUSL, T. Verwaltungsschale. *in Handbuch Industrie 4.0: Produktion, Automatisierung und Logistik*. 2019.

19. BREIVOLD, Hongyu Pei; JANSEN, Anton; SANDSTRÖM, Kristian; CRNKOVIC, Ivica. Virtualize for Architecture Sustainability in Industrial Automation. In: *2013 IEEE 16th International Conference on Computational Science and Engineering*. 2013, S. 409–415. Abger. unter DOI: 10.1109/CSE.2013.69.

20. BREIVOLD, Hongyu Pei; SANDSTRÖM, Kristian. Virtualize for test environment in industrial automation. In: *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. 2014, S. 1–8. Abger. unter DOI: 10.1109/ETFA.2014.7005089.

21. BRENDAN, Burns; JOE, Beda; KALSEY, Hightower; THOMAS, Demmig. *Kubernetes: Eine kompakte Einführung*. 2020. ISBN 978-3864908033.

22. BUYYA, Rajkumar; VECCHIOLA, Christian; SELVI, S. Thamarai. *Mastering Cloud Computing*. Chapter 3 - Virtualization. Hrsg. von BUYYA, Rajkumar; VECCHIOLA, Christian; SELVI, S. Thamarai. Boston: Morgan Kaufmann, 2013. ISBN 978-0-12-411454-8. Abger. unter DOI: https://doi.org/10.1016/B978-0-12-411454-8.00003-6.

23. CHEN, Yinong; DU, Zhihui; GARCIA-ACOSTA, Marcos. Robot as a Service in Cloud Computing. In: 2010, S. 151–158. Abger. unter DOI: 10.1109/SOSE.2010.44.

24. CHRIS PAUL IATROU; MARKUS GRAUBE; LEON URBAS; TIM-PETER HENRICHS, Stefan Erben. *NOA Verification of Request: Reintegrating insights of cloud based added value service*. Atp - Magazine. 2018.

25. CRESPO, A.; RIPOLL, I.; MASMANO, M. Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach. In: *2010 European Dependable Computing Conference*. 2010, S. 67–72. Abger. unter DOI: 10.1109/EDCC.2010.18.

26. CRUZ, Tiago; SIMÕES, Paulo; MONTEIRO, Edmundo. Virtualizing Programmable Logic Controllers: Toward a Convergent Approach. *IEEE Embedded Systems Letters*. 2016, Jg. 8, Nr. 4, S. 69–72. Abger. unter DOI: 10.1109/LES.2016.2608418.

27. D., Juergens; D., Reinhardt; R., Schneider; G., Hofstteter. *Implementing mixed criticality software integration on multicore - a cost model and the lessons learned*. 2015.

28. *Details of the Asset Administration Shell: Part 1 - The exchange of information between partners in the value chain of Industrie 4.0*. [o.D.]. Auch verfügbar unter: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2018-verwaltungsschale-im-detail.html.

29. DILLE, Nicholas; GROTE, Marc; KACZENSKI, Nils; KAPPEN, Jan. *Microsoft Hyper-V. Das Handbuch für Administratoren*. Rheinwerk-Verlag, 2017. ISBN 978-3-8362-4327-8.

30. DIN SPEC 91345:2016-04 - Referenzarchitekturmodell Industrie 4.0 (RAMI4.0). [o.D.].

31. DOCKER. [o.D.]. Auch verfügbar unter: https://www.docker.com/.

32. DOCKER. *Documentation of Docker*. 2018.

33. DOLUI, Koustabh; KIRALY, Csaba. Towards Multi-Container Deployment on IoT Gateways. 2018, S. 1–7. Abger. unter DOI: 10.1109/GLOCOM.2018.8647688.

34. ELFAHAAM, Haitham. *A Runtime Adaptation Concept to reinforce Versatility in Industrial Automation*. 2019. ISBN 978-3-18-526708-6.

35. ELFAHAM, Haitham; EPPLE, Ulrich. Meta models for intralogistics. *at - Automatisierungstechnik*. 2020, Jg. 68, Nr. 3, S. 208–221. Abger. unter DOI: doi:10.1515/auto-2019-0083.

36. FERRER, Borja Ramis; MOHAMMED, Wael M.; CHEN, Enbo; LASTRA, Jose L. Martinez. Connecting web-based IoT devices to a cloud-based manufacturing platform. In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. 2017, S. 8628–8633. Abger. unter DOI: 10.1109/IECON.2017.8217516.

37. FÜRST, Simon; BECHTER, Markus. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform. [o.D.]. Abger. unter DOI: 10.1109/DSN-W.2016.24.

38. GILANI, Syed Shiraz; JUNGBLUTH, Florian; FLATT, Holger; WENDT, Verena; JASPERNEITE, Jürgen. Alternative controls for soft real-time industrial control services in case of broken cloud links. 2016. Abger. unter DOI: 10.1109/ETFA.2016.7733669.

39. GIVEHCHI, Omid; IMTIAZ, Jahanzaib; TRSEK, H.; JASPERNEITE, J. Control-as-a-service from the cloud: A case study for using virtualized PLCs. *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. 2014, S. 1–4. Abger. unter DOI: 10.1109/WFCS.2014.6837587.

40. GOLDBERG, Robert P. Survey of virtual machine research. *Computer*. 1974, Jg. 7, Nr. 6, S. 34–45. Abger. unter DOI: 10.1109/MC.1974.6323581.

41. GOLDSCHMIDT, Thomas; HAUCK-STATTELMANN, Stefan; MALAKUTI, Somayeh; GRÜNER, Sten. Container-based architecture for flexible industrial control applications. *Journal of Systems Architecture*. 2018, Jg. 84, S. 28–36. ISSN 1383-7621. Auch verfügbar unter: https://www.sciencedirect.com/science/article/pii/S1383762117304988.

42. GOLDSCHMIDT, Thomas; MURUGAIAH, Mahesh Kumar; SONNTAG, Christian; SCHLICH, Bastian; BIALLAS, Sebastian; WEBER, Peter. Cloud-Based Control: A Multi-tenant, Horizontally Scalable Soft-PLC. In: *2015 IEEE 8th International Conference on Cloud Computing*. 2015, S. 909–916. Abger. unter DOI: 10.1109/CLOUD.2015.124.

43. GÖLZER, Philipp. *Big Data in Industrie 4.0 - Eine strukturierte Aufarbeitung von Anforderungen, Anwendungsfällen und deren Umsetzung*. 2017.

44. GRIES CASPAR AND WENGER MONIKA AND AZARMIPOUR MAHYAR. *PC2.5 Concept for the Software Abstraction Layer between PikeOS and the Generic Application Container*. Hrsg. von BASYS4.0-PROJECT, Foerderkennzeichen 01IS16022. [o.D.].

45. GRIEVES M., Vickers J. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems, In: Kahlen FJ., Flumerfelt S., Alves A. (eds) Transdisciplinary Perspectives on Complex Systems. Springer*. 2017. ISBN 978-3-319-38754-3.

46. GROTHOFF, Julian Alexander; WAGNER, Constantin August; EPPLE, Ulrich. *BaSys 4.0: Metamodell der Komponenten und Ihres Aufbaus; 1st ed.* RWTH Aachen University, 2018. Abger. unter DOI: 10.18154/RWTH-2018-225880.

47. GRÜNER, Sten; EPPLE, Ulrich. Paradigms for unified runtime systems in industrial automation. In: *2013 European Control Conference (ECC)*. 2013, S. 3925–3930. Abger. unter DOI: 10.23919/ECC.2013.6669313.

48. GUO, SONG AND ZENG, DEZE. *Cyber-Physical Systems: Architecture, Security and Application*. Cham: Springer International Publishing, 2019. EAI/Springer Innovations in Communication and Computing. ISBN 978-3-319-92564-6.

49. HEGAZY, Tamir; HEFEEDA, Mohamed. Industrial Automation as a Cloud Service. 2015, Nr. 10. Abger. unter DOI: 10.1109/TPDS.2014.2359894.

50. HEISER, Gernot. The role of virtualization in embedded systems. 2008, S. 11–16. ISBN 978-1-60558-126-2. Abger. unter DOI: 10.1145/1435458.1435461.

51. HEISER, Gernot; LESLIE, Ben. The OKL4 Microvisor: Convergence point of microkernels and hypervisors. 2010, S. 19–24.

52. IEC 62264, Integration von Unternehmens-EDV und Leitsystemen. [o.D.].

53. IEC 62541. *OPC Unified Architecture: Part 1 - 10*. 2010.

54. IGOR, Kalkov. *A Real-time Capable, Open-Source-based Platform for Off-the-Shelf Embedded Devices*. 2018. ISSN 0935-3232.

55. *Industrial Internet Consortium*. [o.D.]. Auch verfügbar unter: https://www.iiconsortium.org/.

56. *Industrie 4.0*. [o.D.]. Auch verfügbar unter: https://www.plattform-i40.de/PI40/Navigation/DE/Home/home.html.

57. ISA-95, Enterprise-Control System Integration Part 1: Models and Terminology. [o.D.].

58. J. POPEK, Gerald; P. GOLBERG, Robert. Formal Requirements for Virtualizable Third Generation Architectures. 1974.

59. JACQUES BRYGIER; MEMET OEZER. *Safety and Security for the Internet of Things*. 2016. Auch verfügbar unter: https://hal.archives-ouvertes.fr/hal-01292301/.

60. KOBRYN, Pamela A; TUEGEL, Eric J; BRANCH, Structural Mechanics. Condition-based Maintenance Plus Structural Integrity (CBM+ SI) & the Airframe Digital Twin. *USAF Air Force Research Laboratory, 88ABW-201101428*. 2011.

61. LANGMANN, Reinhard; ROJAS-PEÑA, Leandro F. A PLC as an Industry 4.0 component. In: *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. 2016, S. 10–15. Abger. unter DOI: 10.1109/REV.2016.7444433.

62. LANGMANN, Reinhard; STILLER, Michael. The PLC as a Smart Service in Industry 4.0 Production Systems. *Applied Sciences*. 2019, Jg. 9, Nr. 18. ISSN 2076-3417. Abger. unter DOI: 10.3390/app9183815.

63. LEMERRE, Matthieu; OHAYON, Emmanuel; CHABROL, Damien; JAN, Mathieu; JACQUES, Marie-Bénédicte. Method and Tools for Mixed-Criticality Real-Time Applications within PharOS. *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 2011, Jg. 0, S. 41–48. ISBN 978-0-7695-4377-2. Abger. unter DOI: 10.1109/ISORCW.2011.15.

64. LI, Zheng; KIHL, Maria; LU, Qinghua; ANDERSSON, Jens A. Performance overhead comparison between hypervisor and container based virtualization. In: *Proceedings - 31st IEEE International Conference on Advanced Information Networking and Applications, AINA 2017*. Institute of Electrical und Electronics Engineers Inc., 2017, S. 955–962. ISBN 9781509060283. Abger. unter DOI: 10.1109/AINA.2017.79.

65. LIN, Chung-Wei; KIM, BaekGyu; SHIRAISHI, Shinichi. Hardware Virtualization and Task Allocation for Plug-and-Play Automotive Systems. *IEEE Design Test*. 2019, S. 1–1. Abger. unter DOI: 10.1109/MDAT.2019.2932936.

66. LUCAS, Pierre; CHAPPUIS, Kevin; BOUTIN, Benjamin; VETTER, Julian; RAHO, Daniel. VOSYSmonitor, a TrustZone-based Hypervisor for ISO 26262 Mixed-critical System. In: *2018 23rd Conference of Open Innovations Association (FRUCT)*. 2018, S. 231–238. Abger. unter DOI: 10.23919/FRUCT.2018.8588018.

67. M, Grieves. *Digital twin: manufacturing excellence through virtual factory replication*. 2014. Auch verfügbar unter: www.apriso.com/library/Whitepaper_%20Dr_Grieves_DigitalTwin_ManufacturingExcellence.php.

68. M. POLKE. *Prozeßleittechnik: Mit 8 Tabellen*. 2., völlig überarb. und stark erw. Aufl. München: Oldenbourg, 1994. ISBN 3-486-22549-9.

69. MAHYAR, Azarmipour; TROTHA CHRISTIAN, von; ULRICH, Epple; ZEESHAN, Ansar; CASPAR, Gries. Realisierung der NAMUR-Diode mittels Virtualisierung. In: *atp magazin 5/2020*. 2020, S. 2878–2883. Abger. unter DOI: https://doi.org/10.17560/atp.v62i5.2472.

70. MEYER, Dirk. *Objektverwaltungskonzept für die operative Prozeßleittechnik: Zugl.: Aachen, Techn. Hochsch., Diss*. Als Ms. gedr. Düsseldorf: VDI-Verl., 2002. Fortschritt-Berichte VDI Reihe 8 Meß-, Steuerungs- und Regelungstechnik. ISBN 3183940086.

71. MOHAN RAJ, V.K.; SHRIRAM, R. A study on server Sleep state transition to reduce power consumption in a virtualized server cluster environment. In: *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012)*. 2012, S. 1–6. Abger. unter DOI: 10.1109/COMSNETS.2012.6151371.

72. MORABITO, Roberto. Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation. 2017. Abger. unter DOI: 10.1109/ACCESS.2017.2704444.

73. MURRAY, Glenn; JOHNSTONE, Michael N.; VALLI, Craig. The convergence of IT and OT in critical infrastructure. 2017. Abger. unter DOI: 10.4225/75/5a84f7b595b4e.

74. NE 175. *Namur Open Architechture - NOA Concept*. [o.D.]. Auch verfügbar unter: https://www.namur.net/de/fokusthemen/namur-open-architecture.html.

75. NE177. *NAMUR Open Architecture - NOA Security Architecture and Security Gateway*. [o.D.].

76. ORACLE. *Virtualbox*. [o.D.]. Auch verfügbar unter: www.oracle.com/de/virtualization/virtualbox.

77. PALM, Florian; EPPLE, Ulrich. openAAS - Die offene Entwicklung der Verwaltungsschale. *Tagungsband Automation*. 2017.

78. PETRUCCI, Vinicius; CARRERA, Enrique V.; LOQUES, Orlando; LEITE, Julius C.B.; MOSSÉ, Daniel. Optimized Management of Power and Performance for Virtualized Heterogeneous Server Clusters. In: *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. 2011, S. 23–32. Abger. unter DOI: 10.1109/CCGrid.2011.15.

79. PHILLIP LIPSON, GEERT VAN DER ZALM. *PC vs. PLC: Comparing Control Options*. 2011. Auch verfügbar unter: http://www.msalah.com/A/PCvsPLC.pdf.

80. *PikeOS User Manual*. 2009.

81. POGGI, Tomaso; ONAINDIA, Peio; AZKARATE-ASKATSUA, Mikel; GRÜTTNER, Kim; FAKIH, Maher; PEIRÓ, Salvador; BALBASTRE, Patricia. A Hypervisor Architecture for Low-Power Real-Time Embedded Systems. In: *2018 21st Euromicro Conference on Digital System Design (DSD)*. 2018, S. 252–259. Abger. unter DOI: 10.1109/DSD.2018.00054.

82. POPOVIC, I. T.; RAKIC, A. Z. The Fog-Based Framework for Design of Real-Time Control Systems in Internet of Things Environment. 2018, S. 1–6. Abger. unter DOI: 10.1109/INDEL.2018.8637639.

83. REINHARDT, Dominik; MORGAN, Gary. An embedded hypervisor for safety-relevant automotive E/E-systems. In: *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*. 2014, S. 189–198. Abger. unter DOI: 10.1109/SIES.2014.6871203.

84. REPSCHLAEGER, Jonas; PANNICKE, Danny; ZARNEKOW, Rüdiger. Cloud Computing: Definitionen, Geschäftsmodelle und Entwicklungspotenziale. 2010, Nr. 5. Abger. unter DOI: 10.1007/BF03340507.

85. RUSSELL, B. *Passive Benchmarking with docker LXC, KVM and OpenStack*. 2015.

86. SANDSTROM, Kristian; VULGARAKIS, Aneta; LINDGREN, Markus; NOLTE, Thomas. Virtualization technologies in embedded real-time systems. [o.D.]. Abger. unter DOI: 10.1109/ETFA.2013.6648012.

87. SCHÄUFFELE, Jörg; ZURAWKA, Thomas. *Automotive Software Engineering*. 2010. Abger. unter DOI: 10.1007/978-3-8348-9368-0.

88. SCHENK, Michael; WIRTH, Siegfried; MÜLLER, Egon (Hrsg.). *Fabrikplanung und Fabrikbetrieb: Methoden für die wandlungsfähige, vernetzte und ressourceneffiziente Fabrik*. 2., vollständig überarbeitete und erweiterte Auflage 2014. Berlin: Springer Vieweg, 2014. ISBN 978-3-642-05458-7.

89. SCHLAGER, Martin. *Hardware-in-the-loop simulation*. 2008.

90. SCHÜTZE, Andreas; HELWIG, Nikolai; SCHNEIDER, Tizian. Sensors 4.0: smart sensors and measurement technology enable Industry 4.0. [o.D.]. Auch verfügbar unter: https://doi.org/10.5194/jsss-7-359-2018.

91. SEGOVIA, Vanessa Romero; THEORIN, Alfred. History of Control: History of PLC and DCS. 2013. Auch verfügbar unter: http://archive.control.lth.se/media/Education/DoctorateProgram/2012/HistoryOfControl/Vanessa_Alfred_report.pdf.

92. SEHR, Martin A.; LOHSTROH, Marten; WEBER, Matthew; UGALDE, Ines; WITTE, Martin; NEIDIG, Joerg; HOEME, Stephan; NIKNAMI, Mehrdad; LEE, Edward A. Programmable Logic Controllers in the Context of Industry 4.0. *IEEE Transactions on Industrial Informatics*. 2021, Jg. 17, Nr. 5, S. 3523–3533. Abger. unter DOI: 10.1109/TII.2020.3007764.

93. SHAFTO, Mike; CONROY, Mike; DOYLE, Rich; GLAESSGEN, Ed; KEMP, Chris; LEMOIGNE, Jacqueline; WANG, Lui. Modeling, simulation, information technology & processing roadmap. *National Aeronautics and Space Administration*. 2012, Jg. 32, Nr. 2012, S. 1–38.

94. SPEZIFIKATION DIN SPEC 40912. *Kernmodelle - Beschreibung und Beispiele*. 2014.

95. STRASSER, Thomas; ZOITL, ALOIS : EBENHOFER, GERHARD. 4DIAC - Ein Open Source Framework für verteilte industrielle Automatisierungs- und Steuerungssysteme. In: *Informatik 2010*. Bonn: Ges. für Informatik, 2010, S. 435–440. GI-Edition lecture notes in informatics P, Proceedings. ISBN 978-3-88579-269-7.

96. SYSGO EMBEDDING INNOVATIONS. [o.D.]. Auch verfügbar unter: `https://www.sysgo.com/`.

97. TAUCHNITZ, Thomas; UWE, Maier. *Speicherprogrammierbare Steuerungen*. Handbuch der Prozessautimatisierung, Oldenbourg Industrieverlag, 2009. ISBN 383563142X 2.2.3, ISBN 2.2.3.

98. TERZIMEHIC, Tarik; WENGER, Monika; ZOITL, Alois; BAYHA, Andreas; BECKER, Klaus; MÜLLER, Thorsten; SCHAUERTE, Hubertus. Towards an industry 4.0 compliant control software architecture using IEC 61499 amp; OPC UA. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2017, S. 1–4. Abger. unter DOI: `10.1109/ETFA.2017.8247718`.

99. THÖNNESSEN, David. *Hardware-in-the-Loop testing of industrial automation systems using PLC languages*. Aachen: RWTH Aachen University, 2021. Abger. unter DOI: `10.18154/RWTH-2021-08705`. Diss. RWTH Aachen University.

100. TURNBULL, J. The Docker Book: Containerization is the new virtualization. 2014. ISBN 9780988820203.

101. U. GURAV, R. Shaikh. *Virtualization: a key feature of cloud computing*. [o.D.]. Auch verfügbar unter: `https://doi.org/10.1145/1741906.1741957`.

102. USTUNDAG, Alp; CEVIKCAN, Emre (Hrsg.). *Industry 4.0: Managing the digital transformation*. Cham: Springer, 2018. Springer series in advanced manufacturing. ISBN 978-3-319-57869-9.

103. VAIDYA, Saurabh; AMBAD, Prashant; BHOSLE, Santosh. Industry 4.0 – A Glimpse. 2018. Abger. unter DOI: `10.1016/j.promfg.2018.02.034`.

104. VANDERLEEST, Steven H.; WHITE, Dagan. MPSoC hypervisor: The safe and secure future of avionics. [o.D.]. Abger. unter DOI: `10.1109/DASC.2015.7311448`.

105. *VDI Richtlinien 5201, Wandlungsfaehigkeit Beschreibung und Messung der Wandlungsfaehigkeit produzierender Unternehmen Beispiel Medizintechnik*. VDI-Gesellschaft Technologies of Life Sciences, [o.D.].

106. Virtualisierung in der Automatisierungstechnik am Beispiel des SIMATIC S Software, https://silo.tips/download/virtualisierung-in-der-automatisierungstechnik-am-beispiel-des-simatic-s-softwar. [o.D.].

107. VMWARE. *Understanding Full Virtualization, Paravirtualization, and Hardware Assist*. 2008. Auch verfügbar unter: `https://www.vmware.com/de/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html`.

108. VOGEL-HEUSER, Birgit. Die Auflösung der Automatisierungspyramide: Die Maschinenkommunikation in der Smarten Fabrik. 2016.

109. W., Dorst. Umsetzungsstrategie Industrie 4.0: Ergebnisbericht der Plattform Industrie 4.0. 2015.

110. WAGNER, Constantin; GROTHOFF, Julian; EPPLE, Ulrich; DRATH, Rainer; MALAKUTI, Somayeh; GRÜNER, Sten; HOFFMEISTER, Michael; ZIMERMANN, Patrick. The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant. In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2017, S. 1–8. Abger. unter DOI: 10.1109/ETFA.2017.8247583.

111. WEBB, John W.; REIS, Ronald A. (Hrsg.). *Programmable logic controllers: Principles and applications*. 5. ed. Upper Saddle River, NJ: Prentice Hall, 2003. ISBN 978-0-13-041672-8.

112. WIND RIVER. *Wind River Hypervisor*. [o.D.]. Auch verfügbar unter: https://www.windriver.com/products/.

113. WIND RIVER. *Wind River Hypervisor*. [o.D.]. Auch verfügbar unter: https://www.windriver.com/products/product-notes/wind-river-hypervisor-product-note.pdf.

114. WU, Dazhong; GREER, Matthew John; ROSEN, David W.; SCHAEFER, Dirk. Cloud manufacturing: Strategic vision and state-of-the-art. 2013, Nr. 4. Abger. unter DOI: 10.1016/j.jmsy.2013.04.008.

115. XF, Yao; JJ, Zhou; CJ, Zhang; M, Liu. Proactive manufacturing: a big-data based emerging manufacturing paradigm. *Comput Integr Manuf Syst 23(1):172–185*. 2017.

116. XU, Li Da; HE, Wu; LI, Shancang (Hrsg.). *Internet of Things in Industries: A Survey*. 2014. Abger. unter DOI: 10.1109/TII.2014.2300753.

117. YONGWANG, Zhao; ZHIBIN, Yangi; MA, Dianfu. A survey on formal specification and verification of separation kernels. In: 2017, S. 585–607. Abger. unter DOI: DOI10.1007/s11704-016-4226-2.

118. YU, Liyong; GRÜNER, Sten; EPPLE, Ulrich. An engineerable procedure description method for industrial automation. In: *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*. 2013, S. 1–8. Abger. unter DOI: 10.1109/ETFA.2013.6648002.

119. ZHUANG, C.; LIU, J.; XIONG, H. Digital twin-based smart production management and control framework for the complex product assembly shop-floor. *The International Journal of Advanced Manufacturing Technology*. 2018. Abger. unter DOI: doi.org/10.1007/s00170-018-1617-6.

VDI verlag

Alle 23 Reihen der „Fortschritt-Berichte VDI“
in der Übersicht – bequem recherchieren unter:
elibrary.vdi-verlag.de

Und direkt bestellen unter:
www.vdi-nachrichten.com/shop

Reihe 01 Konstruktionstechnik / Maschinenelemente
Reihe 02 Fertigungstechnik
Reihe 03 Verfahrenstechnik
Reihe 04 Bauingenieurwesen
Reihe 05 Grund- und Werkstoffe/Kunststoffe
Reihe 06 Energietechnik
Reihe 07 Strömungstechnik
Reihe 08 Mess-, Steuerungs- und Regelungstechnik
Reihe 09 Elektronik/Mikro- und Nanotechnik
Reihe 10 Informatik/Kommunikation
Reihe 11 Schwingungstechnik
Reihe 12 Verkehrstechnik/Fahrzeugtechnik
Reihe 13 Fördertechnik/Logistik
Reihe 14 Landtechnik/Lebensmitteltechnik
Reihe 15 Umwelttechnik
Reihe 16 Technik und Wirtschaft
Reihe 17 Biotechnik/Medizintechnik
Reihe 18 Mechanik/Bruchmechanik
Reihe 19 Wärmetechnik/Kältetechnik
Reihe 20 Rechnergestützte Verfahren
Reihe 21 Elektrotechnik
Reihe 22 Mensch-Maschine-Systeme
Reihe 23 Technische Gebäudeausrüstung

INGENIEUR.de
BEST MATCH

powered by ASCOTTY

So findet Sie Ihr Traumjob!

Ingenieure aller Fachrichtungen, Absolventen und wechselwillige Professionals aufgepasst: Sagen Sie uns, was Sie können, wollen und lieben – dann bieten Ihnen die besten Unternehmen den passenden Job für Ihr Talent. Schnell, unkompliziert, ohne Aufwand.

DAS SIND IHRE VORTEILE:

Einfache Profilerstellung | Persönliche Beratung | Passgenaue Job-Angebote | Keine aufwändige Job-Suche | Unternehmen bewerben sich bei Ihnen | Kostenfreie Nutzung | **Transparenz:** alle wichtigen Informationen zum Traumjob | **Sicher:** Ihr Arbeitgeber hat keine Einsicht in Ihr Profil

JETZT ALS TALENT REGISTRIEREN:
BESTMATCH.INGENIEUR.DE

VDI

REIHE 08
MESS-,
STEUERUNGS-
UND REGELUNGS-
TECHNIK

NR. 1275

ISBN 978-3-18-527508-1
E-ISBN 978-3-18-627508-0

BAND
1|1

VOLUME
1|1