Carl von Ossietzky
Universität
Oldenburg

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Engineeringprozess für Virtualisierte Dienste in Smart Grids

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation von

Herrn Carsten Krüger

geboren am 02. August 1990 in Norden (Deutschland)

Gutachter:

Prof. Dr. Sebastian Lehnhoff,
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften,
Carl von Ossietzky Universität Oldenburg.

Weitere Gutachter:

Prof. Dr. Christian Rehtanz,
Institut für Energiesysteme, Energieeffizienz und Energiewirtschaft (ie3),
Technische Universität Dortmund.

Tag der Disputation: 27. Februar 2025

Zusammenfassung

Die zunehmende Umstellung von zentraler auf dezentrale Energieerzeugung mittels erneuerbarer Energiequellen erfordert eine Anpassung der bisherigen Betriebsplanungs- und Betriebsführungsprozesse. Die Teilnahme der Verteilnetze an netzunterstützenden Diensten zur besseren Überwachung und Steuerung der Erzeugung in den unteren Spannungsebenen ist notwendig und bedarf der Integration von Informations- und Kommunikationstechnologie (IKT) in die Infrastruktur des Energiesystem. Das daraus resultierende Cyber-Physische Energiesystem (CPES) verknüpft physische Komponenten im Energiesystem mit computergestützten Technologien und ermöglicht die Bereitstellung netzdienlicher Dienste wie beispielsweise die Netzzustandsschätzung, Spannungshaltung oder Redispatch-Maßnahmen.

Verteilnetzstationen, wie Ortsnetzstationen, müssen digital aufgerüstet werden, um die Anforderungen komplexer Überwachungs- und Steuerungsmechanismen zu erfüllen. Stationsgeräte wie Intelligente Elektronische Geräte (IEDs) und Fernwirkeinheiten (RTUs) ermöglichen die Übertragung von Echtzeitdaten und die Fernüberwachung der Stationen. Mit der zunehmenden Anzahl digitaler Komponenten im Netz, steigt die Komplexität und der damit verbundene Verwaltungsaufwand. Konzepte und Paradigmen der Informatik wie Virtualisierung, Microservice-Architekturen und Edge-Computing können dabei helfen, diesen Herausforderungen gerecht zu werden.

Im ersten Schritt untersucht diese Arbeit Konzepte und Technologien zur flexiblen Ausführung von Diensten auf Stationsgeräten. Da die Integration mehrerer Stationsgeräte in eine Station nicht zielführend ist, wird durch den Ansatz der Virtualisierung eine herstellerunabhängige und skalierbare Bereitstellung beliebiger Dienste als isolierte Einheiten in Stationsgeräten untersucht. Ein entsprechendes Virtualisierungskonzept auf Basis existierender Container-Technologien wird entwickelt und in einem industriellen Stationsgerät implementiert. Notwendige Anpassungen der Hardware- und Softwareeigenschaften umfassen neben dem Austausch leistungsfähigerer Hardware auch die Konfiguration eines angepassten Betriebssystems.

Aufbauend auf diesem Konzept untersucht diese Arbeit darüber hinaus die stationsübergreifende Verwaltung aller Dienste in einem Netzausschnitt, um den Verwaltungsaufwand zu begrenzen. Eine entsprechende Erweiterung erfolgt durch die

Integration einer Container-Orchestrierungseinheit als Verwaltungsschicht, sodass ein Verwaltungskonzept entsteht. Dieses Verwaltungskonzept ermöglicht es, die Dienste einzelner Stationen zu überwachen und zu steuern, ohne direkten Zugriff auf das Stationsgerät zu haben.

In der letzten Phase erfolgt die Einbindung der Konzepte in einen standardisierten Engineeringprozess nach IEC 61850. Kontinuierliche und automatisierte Prozessschritte wurden ergänzt, um das dynamische und sich verändernde Verhalten in Smart Grids abzubilden. Dadurch werden notwendige Eigenschaften wie Wartbarkeit, Erweiterbarkeit und Skalierbarkeit gewährleistet.

Die Ergebnisse dieser Arbeit werden im Rahmen einer Labor- und Feldtesterprobung in einem Verteilnetz mit insgesamt elf digitalisierten Ortsnetzstationen validiert.

Abstract

The increasing shift from centralized to decentralized energy generation through renewable energy sources necessitates an adaptation of the existing operational planning and management processes. The participation of distribution networks in grid-supporting services for improved monitoring and control of generation at lower voltage levels is essential and requires the integration of information and communication technology (ICT) into the energy system infrastructure. The resulting Cyber-Physical Energy System (CPES) links physical components in the energy system with computer-based technologies, enabling the provision of grid-supportive services such as state estimation, voltage regulation, and redispatch measures.

Distribution substations, such as secondary substations, must be digitally upgraded to meet the requirements of complex monitoring and control mechanisms. Station devices such as Intelligent Electronic Devices (IEDs) and Remote Terminal Units (RTUs) facilitate the transmission of real-time data and remote monitoring of the stations. With the increasing number of digital components in the grid, the complexity and associated management effort rise. Concepts and paradigms from computer science, such as virtualization, microservice architectures, and edge computing, can help address these challenges.

The first step of this work examines concepts and technologies to flexible execute services on station devices. Since the integration of multiple station devices into one station is not feasible, the approach of virtualization is used to explore the

manufacturer-independent and scalable provision of various services as isolated units within station devices. A corresponding virtualization concept based on container technologies is developed and implemented in an industrial station device. Necessary adjustments to the hardware and software characteristics include the replacement of more powerful hardware and the configuration of a customized operating system.

Building on this concept, this work further investigates the cross-station management of all services in a grid to limit the administrative effort. A corresponding extension is achieved through the integration of a container orchestration unit as a management layer, resulting in a management concept. This management concept allows the monitoring and control of services of individual stations without direct access to the station device.

In the final phase, the concepts are integrated into a standardized engineering process according to IEC 61850. Continuous and automated process steps are added to capture the dynamic and changing behavior in Smart Grids. This ensures necessary properties such as maintainability, expandability, and scalability.

The results of this work are validated through laboratory and field testing in a distribution network with a total of eleven digitalized secondary substations.

Danksagung

Der Weg einer Promotion lässt sich mit vielen Wörtern, Bildern und Gefühlen beschreiben. Einige davon wurden sicherlich von Herausforderungen, Hürden, Rückschlägen und Triumphen begleitet. Ich bin deshalb sehr dankbar, während meiner Höhen und Tiefen Personen an meiner Seite gehabt zu haben, die mich aufgefangen, beraten und unterstützt haben.

Einen besonderen Dank möchte ich meinem Doktorvater Prof. Dr. Sebastian Lehnhoff aussprechen, der mich auf meiner Reise von Anfang bis Ende begleitet hat. In allen Phasen schaffte er es, mich zu leiten, mit seinem Wissen zu unterstützen und mir geeignete Herausforderungen zu stellen. Diesen Dank möchte ich auch meinem Zweitgutachter Prof. Dr.-Ing. Christian Rehtanz aussprechen, der mich bereits zuvor als Teil eines Forschungsprojektes stets mit positiver und begeisternder Stimmung begleitet hat. Die Zusammenarbeit mit beiden ermöglichte mir besondere Einblicke sowohl in die Fachgebiete der Informationssysteme als auch der Energiesysteme.

Dieses unglaubliche Ziel konnte ich nur dank der uneingeschränkten Unterstützung meiner Familie erreichen – allen voran meine Frau, die mir den notwendigen Rückhalt und die Kraft gab, um der Belastung standzuhalten. Mir wurden immer die benötigten Freiräume geschaffen, um der zusätzlichen Belastung bei der Erarbeitung dieser Promotion gerecht zu werden.

Der notwendige wissenschaftliche Austausch wurde maßgeblich durch meine Arbeit am OFFIS – Institut für Informatik in Oldenburg begleitet. Die Einarbeitung in die Problematiken und Herausforderungen zukünftiger Energiesysteme wurde durch den intensiven Forschungsaustausch in der Gruppe *Automation, Communication and Control* begleitet. Meinem früheren Gruppenleiter Dr. Davood Babazadeh möchte ich herzlich danken für die Unterstützung bei der Forschungsausrichtung meiner Arbeit. Durch einen Wechsel in die Gruppe *Smart Grid Testing* wurden mir neue Blickwinkel und Betrachtungsweisen eröffnet. Beide Gruppen waren Teil des Energiebereichs am OFFIS, und ich möchte mich herzlich bei allen Kolleginnen und Kollegen für die meist spontanen, aber interessanten Diskussionen bedanken.

Ein wesentlicher Bestandteil der Forschung wurde im Rahmen eines durch das BMWK gefördertes Forschungsprojekt i-Autonomous „Förderkennzeichen 03EI6001B“ be-

gleitet. Ich bedanke mich herzlich bei allen Teilnehmern dieses Projektes für den wissenschaftlichen Austausch.

Oldenburg, 23, März 2025

Carsten Krüger

Inhaltsverzeichnis

1 Einleitung

Die Herausforderungen weltweiter klimatischer Veränderungen verdeutlichen den Bedarf zur Reduzierung der Emissionen von Treibhausgasen und führen zu einer Dekarbonisierung des Energiesektors. Gesellschaftliche und politische Prozesse zur Umstellung des Energiesystems auf eine nachhaltige, klimafreundliche Energieversorgung als Teil der Energiewende [1] unterstützen diese Vorgehensweise. Erreicht werden kann die Energiewende durch das Ersetzen traditioneller Energieerzeuger, basierend auf fossilen Brennstoffen wie Kohle, Erdöl und Erdgas, durch erneuerbare Energieerzeuger wie Solar-, Wind-, Wasser- und Biomassekraftwerke [2], die auf nachhaltige Energiequellen basieren. Diese Art der Energieerzeugung basiert primär auf kleineren Distributed Energy Resources (DERs) und Windparks entgegen der konventionellen zentralen Energieerzeugungsanlagen. Die Dezentralität und Verteilung der DERs führt zu neuen Herausforderungen und Fragestellungen im Bereich der Betriebsplanung und Betriebsführung im Energiesystem. Unter Anderem die Verschiebung der Einspeisung von „Top-down“ in Richtung „Bottom-up“ bedarf neuer Steuerungs- und Überwachungsmechanismen, um die Netzstabilität zu erhalten [3]. Neben der reinen Stromerzeugung werden ergänzende Dienstleistungen [4] genutzt, um den sicheren und stabilen Betrieb im Stromnetz zu gewährleisten. Darunter fallen Frequenzregelung, Spannungsregelung, Reserveleistung, Blindleistungsunterstützung und Schwarzstartfähigkeit. Diese Dienstleistungen finden bisher primär Anwendung in großen zentralen Energieerzeugungsanlagen oder Umspannwerken auf der Hoch- und Höchstspannungsebene und sind nur geringfügig an die verteilten und dezentralen Eigenschaften der Distributed Energy Resource (DER) angepasst. Mit zunehmendem Anteil an erneuerbaren Energien im Strommix, sowie die Ergänzung durch neue Lasten wie Wärmepumpen und Speichern, ist eine Teilnahme an den ergänzenden Dienstleistungen der Stationen und DER in der Verteilnetzebene verpflichtend und benötigt geeignete Überwachungs- und Steuerungsmechanismen.

1.1 Dienste in Cyber-Physischen Energiesystemen

Die Teilnahme der Verteilnetzebene an den Herausforderungen der Betriebsplanung und Betriebsführung im Stromsystem durch geeignete Überwachungs- und Steuerungsmechanismen ist stark mit der Digitalisierung im Energiesystem verknüpft. Die Integration von Informations- und Kommunikationstechnologie (IKT) in das Energiesystem ermöglicht die Betrachtung eines Cyber-Physischem Energiesystems (CPES)[1]. Ein CPES[5, 6] verknüpft die physischen Komponenten im Energiesystem mit computergestützten Technologien und Netzwerken. Sensoren, Aktoren und Steuerungssysteme werden innerhalb eines CPES eingesetzt, um Daten über den Zustand und den Betrieb des Energiesystems zu sammeln, zu übertragen und zu analysieren. Die gesammelten Daten bilden die Grundlage zur Umsetzung von Diensten in der Verteilnetzebene und ermöglichen eine bessere Bereitstellung der ergänzenden Dienstleistungen [4]. Dienste [2], oft auch als Netz-Dienste oder Funktionen bekannt, sind beispielsweise State Estimation (SE) (deutsch: Netzzustandsschätzung [3] [7], Coordinated Voltage Control (CVC) (deutsch: koordinierter Spannungshaltung [4][8] oder Redispatch-Maßnahmen [3].

1.2 Stationsautomatisierung in der Verteilnetzebene

Stationen der Verteilnetzebene, wie Ortsnetzstationen, sind aktiv in die Veränderungen eines CPES eingebunden. Bisherige nicht digitale Ortsnetzstationen erfordern manuelle Eingriffe für Steuerungs- und Wartungsaufgaben. Daraus resultierte eine begrenzte Datenverfügbarkeit und geringe Flexibilität [7]. Optimierungen oder Anpassungen müssen durch physische Änderungen an der Hardware erfolgen, die zeitaufwändig und teuer sind. Der Automatisierungsgrad ist mit einigen festgelegten Schutzfunktionen und Datenloggern begrenzt. Moderne digitale Ortsnetzstationen ermöglichen die Übertragung von Echtzeit-Daten und die Fernüberwachung der Stationsgeräte wie Intelligent Electronic Devices (IEDs)(deutsch: Intelligentes Elektronische Gerät [5]) und Remote Terminal Units (RTUs) (deutsch: Fernwirkgerät [6]). Analysen und Entscheidungsfindungen in Echtzeit auf Basis der übertragenen Daten

[1] Im Rahmen dieser Dissertation werden ausschließlich die Synergien zwischen IKT und Stromsystem betrachtet. Andere Bereiche des Energiesystems wie Industrie, Gas oder Mobilität werden nicht berücksichtigt.
[2] Eine Definition eines Dienstes ist im Abschnitt 3 beschrieben.
[3] In dieser Arbeit wird der englische Fachbegriff State Estimation (SE) verwendet.
[4] In dieser Arbeit wird der englische Fachbegriff Coordinated Voltage Control (CVC) verwendet.
[5] In dieser Arbeit wird der englische Fachbegriff Intelligent Electronic Devices (IEDs) verwendet.
[6] In dieser Arbeit wird der englische Fachbegriff Remote Terminal Units (RTUs) verwendet.

ermöglichen eine präzisere Steuerung des Gesamtsystems. Gleichzeitig profitiert der Automatisierungsgrad digitaler Ortsnetzstationen stark von der hohen Flexibilität und Anpassungsfähigkeit, welche durch die digitalen Schnittstellen geschaffen werden. Komplexere Automatisierungsfunktionen innerhalb der Stationen oder stationsübergreifend ermöglichen die automatische Optimierung und Steuerung ganzer Bereiche (angelehnt an das Konzept des Edge-Computing [7]).

1.3 Engineeringprozess für Stationen

Aus der Perspektive eines Netzbetreibers beinhaltet der Engineeringprozess für Stationen viele Prozesse wie Planung, Design, Implementierung und Validierung der Anlagen und ihrer Komponenten. In Bezug auf die Stationsautomatisierung und Bereitstellung von Diensten in einem CPES bezieht sich das Engineering auf die technische Spezifikation und Konfiguration von Systemgeräten, die Integration der Systemgeräte, die Kommunikationskonfiguration, Inbetriebnahme und Validierung der ausgeführten Dienste. Ein bekannter internationaler Standard für Kommunikation und Systemautomatisierung ist in der Norm IEC 61850 definiert [9] [8] und fördert eine strukturierte, flexible und interoperable Umgebung für die Automatisierung auf Stationsebenen. Ein Vorteil dieses standardisierten Vorgehens ist die Interoperabilität zwischen verschiedenen Stationsgeräten unterschiedlicher Hersteller durch einheitliche Datenmodelle und Kommunikationsprotokolle.

1.4 Konzepte der Informatik

In der modernen Informatik spielen Virtualisierung, die Orchestrierung von Containern sowie Cloud- und Edge-Computing eine entscheidende Rolle bei der Entwicklung und dem Betrieb von IT-Systemen [10]. Diese Konzepte und Paradigmen ermöglichen eine flexible, skalierbare und effiziente Verwaltung von Ressourcen und Anwendungen, die eine Schlüsselrolle in der Gestaltung zukünftiger CPES einnehmen können.

Virtualisierung bezieht sich auf die Erstellung virtueller Versionen von physischen Ressourcen und ermöglicht beispielsweise die Bereitstellung von Diensten in Instanzen [11]. Die Orchestrierung von Containern ist ein weiteres Schlüsselelement

[7] Eine Erklärung erfolgt im Abschnitt 2.3.2
[8] Eine Übersicht über diesen Standard ist in Abschnitt 2.1 zu finden

moderner IT-Infrastrukturen. Container-Technologien ermöglichen es, Anwendungen in isolierten Umgebungen zu verpacken, die alle erforderlichen Abhängigkeiten enthalten. Orchestrierungstools verwalten diese Container automatisch und sorgen für Skalierbarkeit, Ausfallsicherheit und einfache Bereitstellung. Ergänzend können Paradigmen wie Edge Computing und Cloud Computing genutzt werden, um die Verarbeitung von Daten näher an die Quelle der Datenentstehung zu verlagern. Dies reduziert die Latenzzeiten, wodurch zum Beispiel in einem CPES eine schnellere Verarbeitung und Reaktion auf lokale Ereignisse erfolgen kann, was besonders für zeitkritische Anwendungen wie Netzstabilität und Ausfallsicherung wichtig ist. Geräte am Rand des Netzwerks, wie intelligente Sensoren und Aktoren, können sofortige Entscheidungen treffen und Maßnahmen ergreifen, bevor die Daten an zentrale Systeme weitergeleitet werden.

1.5 Herausforderungen und Forschungsbedarf

Die Notwendigkeit der Teilnahme an den ergänzenden Dienstleistungen durch Stationen und DERs der Verteilnetzebene ist der Auslöser für die steigenden Anforderungen an bereitzustellende Dienste sowie an Automatisierungs- und Engineeringsprozesse. Die nachfolgenden Unterkapitel beschreiben Herausforderungen und Forschungsbedarf in unterschiedlichen Bereichen des Verteilnetzes.

1.5.1 Dezentrale Bereitstellung von Diensten

Die Verlagerung der Stromerzeugung und der größere Verbrauch durch e-Mobilität wirkt sich auf die Belastung der Verleitnetzstruktur aus. Das führt dazu dass erbrachten Dienstleistungen vom Übertragungsnetz in die Verteilnetzstruktur übertragen werden und die Zahl der bereitzustellenden Dienste signifikant aufgrund der vielen Anlagen und Netzabschnitte steigt. Eine Möglichkeit, mit dieser Art der Skalierung umzugehen, ist die dezentrale Bereitstellung von Diensten in Stationen des Verteilnetzes anstelle der zentralen Verarbeitung über eine leittechnische Anbindung. Werden die nicht digitalisierten Ortsnetzstationen betrachtet, müssen Stationsgeräte manuell geändert, erneuert oder hinzugefügt werden. Zusätzlich können nur Dienste bereitgestellt werden, die lokale Informationen aus Sensoren verarbeiten. Im Fall von digitalen Ortsnetzstationen können Stationsgeräte auch über eine Fernwartung verändert werden. Der Funktionsumfang der Stationsgeräte ist zumeist vordefiniert und kann über die Fernwartung neu parametriert und geändert werden. Die Bereitstellung von Diensten mit anderem Funktionsumfang bedarf eines Austauschs der

Stationsgeräte. Als Forschungsbedarf lässt sich die Notwendigkeit nach flexiblen, herstellerunabhängigen Systemarchitekturen feststellen, die es ermöglichen, beliebige Dienste fernwartbar und voneinander isoliert bereitzustellen. Einige Konzepte aus der Informatik zur Abstrahierung von Software und Hardware sollten im Kontext eines CPES untersucht werden.

1.5.2 Verwaltung und Verteilung von Diensten

Die Digitalisierung eines substanziellen Anteils von Stationen im Verteilnetz zur Bereitstellung von Diensten erhöht die Komplexität in Bezug auf deren Verwaltung. Das betrifft nicht nur Betriebsplanungsprozesse wie den Roll-Out (initiale Planung, Verteilung und Konfiguration der Dienste), sondern auch Betriebsführungsprozesse wie die Überwachung des Systemstatus, die Erkennung von Abweichungen und die Reaktion auf notwendige Veränderungen. Zusätzliche Komplexität durch herstellerspezifische Lösungen mit eigenständigen Betriebsführungs- und Wartungslösungen und fehlender Interoperabilität zwischen Herstellern gilt es abzubauen. Grundlegende Konzepte wie Containerisierung und Orchestrierung ermöglichen die Verwaltung, Verteilung, Überwachung und Steuerung kleinster Softwareelemente auf dezentralen Hardwarearchitekturen.

1.5.3 Standardisierung und Engineering von Diensten

Von Netzbetreibern angewandte Engineeringprozesse wie der IEC 61850 ermöglicht insbesondere die Abbildung von Stationsautomatisierungsfunktionen als Teil des Betriebsplanungsprozesses. Auch unterstützen einige Engineeringprozesse die Interoperabilität zwischen Herstellern in Bezug auf die verwendeten Stationsgeräte. Bisher nicht unterstützt wird die Betrachtung hardwareunabhängiger Dienste. Im Vergleich zur vollständigen Unterstützung der Betriebsplanungsprozesse unterstützen die Engineeringsprozesse keine der vorher aufgeführten Konzepte oder vergleichbaren Prozesse zur Verwaltung und Orchestrierung von Diensten, welche Teil eines Betriebsführungskonzeptes sind. Es ist notwendig die Weiterentwicklung standardisierterter Engineeringprozesse im Bezug auf der flexiblen Bereitstellung von Diensten zu untersuchen, um den Herausforderungen der Betriebsführungsprozesse verteilter Dienste und der dezentralen Bereitstellung gerecht zu werden.

1.5.4 Herausforderungen in Forschungsprojekten

In den vom BMWK geförderten Forschungsprojekten i-Automate [12] und i-Autonomous [13] wurde die Stationsautomatisierung in Bezug auf die Durchführung von Funktionen in Ortsnetzstationen untersucht. Abbildung 1.1 zeigt eine schematische Darstellung einer Automatisierungsarchitektur, die neben der standardisierten Bereitstellung von Funktionen, Datenmodellen und Kommunikationsschnittstellen auch die Verteilung von Funktionen abbildet. Beide Projekte beschäftigten sich mit den Herausforderungen zukünftiger Stationsautomatisierungsprozesse.

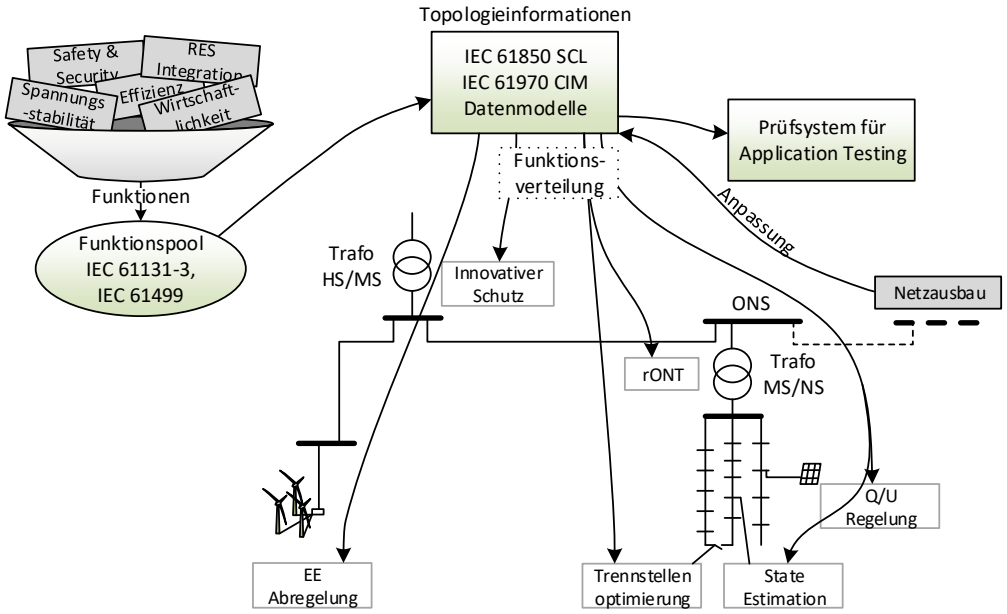

Abb. 1.1.: Automatisierungsarchitektur des Forschungsprojekts i-Automate [12]

1.6 Forschungsfrage und Zielsetzung

Angelehnt an die identifizierten Herausforderungen und den Forschungsbedarf aus Abschnitt 1.5 erfolgt in dieser Dissertation die Bearbeitung einer primären Forschungsfrage (FF) mit drei spezifischen Unterforschungsfragen. Der nachfolgende Abschnitt beschreibt jede der Forschungsfragen zusammen mit den funktionalen Anforderungen (FA) an die zu entwickelnden Artefakte. Die primäre Forschungsfrage lautet wie folgt:

Primäre FF: *Wie muss ein Engineeringprozess für virtualisierte Dienste in digitalisierten Verteilnetzen entworfen werden?*

Die primäre Forschungsfrage untersucht die Verwendbarkeit neuer digitalisierter Prozesse, Technologien und Konzepte aus der Informatik für die Anwendungsbereiche der Stationsautomatisierung im Verteilnetz. Das Ziel ist eine erhöhte Flexibilität durch die Bereitstellung und Verwaltung von Diensten zu erreichen, die zu einem erhöhten Autonomiegrad [14] führt. Grundlegende Anforderungen an die Beobachtbarkeit und Steuerbarkeit in der Nieder- und Mittelspannung sollen mit dieser Forschungsfrage adressiert werden. Parallele Aktivitäten im Ausbau konventioneller Ortsnetzstationen zu digitalisierten Ortsnetzstationen [15] basieren bisher auf herstellerspezifischen Lösungen, die nicht vollständig in die standardisierten Systemautomatisierungsprozesse eingebettet sind. Aus der primären Forschungsfrage wurden insgesamt drei spezifische Forschungsfragen abgeleitet, die im nachfolgenden detailliert beschrieben werden.

Spezifische FF1: *Wie lassen sich relevante Hardware- und Softwareanforderungen für die Virtualisierung von Diensten in Stationsgeräten abbilden?*

In der ersten spezifischen Forschungsfrage wird die Eignung eingesetzter Stationsgeräte und Dienste in Bezug auf die Verwendung von Virtualisierung untersucht. Virtualisierung ist ein bekanntes Konzept der Informatik und bedingt spezifische Eigenschaften der genutzten Hardware, des Betriebssystems und der ausgeführten Dienste. Ziel der Untersuchung ist die Identifizierung relevanter Hardware- und Softwareanforderungen, die es erlauben, Dienste als isolierte Einheiten auf einem Stationsgerät auszuführen. Auf Grundlage der Anforderungen wird ein **Virtualisierungskonzept** als erstes Artefakt unter Berücksichtigung der funktionalen Anforderungen 1-3 als Teil der Arbeit bereitgestellt.

FA1: Das aus dem Artefakt 1 entworfene Konzept muss die simultane Ausführung eines oder mehrerer Dienste auf einer Hardware ermöglichen. Eine unabhängige, konsistente und sichere Ausführung der Dienste muss durch eine Isolation der genutzten Ressourcen gewährleistet werden.

FA2: Trotz der in FA1 geforderten Isolation muss das Konzept die Ausführung auf Stationsgeräten unter Berücksichtigung der notwendigen Zugriffe auf Peripherie und Netzwerke gewährleisten. Genauer: Jedem Dienst muss der Zugriff zur Sensorik und Aktorik sowie die kommunikative Anbindung über eine Netzwerkschnittstelle ermöglicht werden.

FA3: Die für einen Dienst aufgewendeten Ressourcen müssen entsprechend der Verfügbarkeit im Stationsgerät konfigurierbar sein. Das Konzept muss in der

Lage sein, die Konfiguration sowohl der minimal erforderlichen als auch der maximal zulässigen Ressourcen für jeden Dienst abzubilden.

Die Nutzung der Virtualisierung bezieht sich nur auf die lokal isolierte Ausführung von Diensten auf einem Hardware-Gerät. Die nächste Forschungsfrage untersucht die lokale und verteilte Bereitstellung und Verwaltung von Diensten innerhalb eines Gerätes oder über mehrere Geräte und Stationen hinaus.

Spezifische FF2: *Welche Konzepte zur Verwaltung, Überwachung und Steuerung von virtualisierten Diensten lassen sich im Kontext von Edge-Computing für Stationsgeräte nutzen?*

Unter Verwendung des Artefakts 1 und Konzepten der Informatik im Bereich Edge-Computing, Dienste-Management und Orchestrierung untersucht die zweite spezifische Forschungsfrage, welche Konzepte sich für die Verteilung, Überwachung und Steuerung von Diensten nutzen lassen. Ziel dieser Forschungsfrage ist es, ein **Verwaltungskonzept** als Artefakt 2 bereitzustellen, welches es ermöglicht, Dienste als abstrakte virtualisierte Instanzen automatisiert über alle Stationen in einem Netzgebiet zu verteilen, zu überwachen und im Fall eines Fehlverhaltens darauf zu reagieren. Die funktionalen Anforderungen 4 und 5 müssen durch das Artefakt 2 berücksichtigt werden.

FA4: Das aus dem Artefakt 2 entworfene Konzept muss die Verteilung eines oder mehrerer Dienste auf einem oder mehreren Stationsgeräten ermöglichen. Die Verteilung beinhaltet die Auswahl des Dienstes, die Überprüfung der verfügbaren Ressourcen auf dem Zielgerät, die Übertragung der Software-Instanz und die Initialisierung des Dienstes.

FA5: Die Leistung der Dienste muss kontinuierlich überwacht werden. Fehlverhalten, wie der Ausfall eines Dienstes oder die Nichtverfügbarkeit von Stationsgeräten, müssen durch das Konzept erfasst und geeignete Maßnahmen ergriffen werden. Diese Maßnahmen müssen sowohl in automatisierten Routinen als auch in manuellen Eingriffen umsetzbar sein.

Die Ergebnisse der ersten beiden spezifischen Forschungsfragen und daraus resultierenden Artefakte müssen in standardisierten Systemautomatisierungsprozessen Anwendung finden, um einen vollständigen Engineeringprozess über alle Ebenen der Betriebsplanung und Betriebsführung zu schaffen. In diesem Zusammenhang wird die folgende Forschungsfrage betrachtet.

Spezifische FF3: *Wie lassen sich Virtualisierungs- und Verwaltungskonzepte in die standardisierten Systemautomatisierungsprozesse von Verteilnetzbetreibern integrieren?*

Teil der Virtualisierungs- und Verwaltungskonzepte sind Meta-Informationen, die eine Beschreibung der Dienste in Bezug auf Ausführung, Verteilung, Ressourcen und Weiteres umfassen. Im Rahmen der Forschungsfrage soll untersucht werden, ob diese Meta-Informationen in standardisierte Systemautomatisierungsprozesse abbildbar sind. Darüber hinaus muss untersucht werden, ob kontinuierliche Überwachungs- und Steuerungsprozesse in solchen Prozessen anwendbar sind oder ob zusätzliche Erweiterungen oder Prozesse geschaffen werden müssen. Als Ziel wird ein **Engineeringprozess** als Artefakt 3 erwartet, der eine vollständige Integration der Virtualisierungs- und Verwaltungskonzepte unterstützt unter Berücksichtigung der funktionalen Anforderungen 6-8.

FA6: Der Engineeringprozess im Artefakt 3 muss auf einem der standardisierten Systemautomatisierungsprozesse basieren.

FA7: Die erarbeiteten Konzepte aus Artefakt 1 und 2 müssen Berücksichtigung in dem erweiterten Engineeringprozess finden.

FA8: Standardisierte Kommunikation (z.B. IEC 61850 MMS) zwischen den Diensten muss innerhalb und außerhalb der Gerätegrenzen unterstützt werden.

Die primäre Fragestellung wird durch die Beantwortung aller spezifischen Forschungsfragen und ihrer Artefakte erfolgen. Jedes Artefakt gilt als vollständig, wenn die dazugehörigen funktionalen Anforderungen erfüllt sind. Zusätzlich zur Funktionalität der Artefakte beziehen sich nicht funktionale Anforderungen auf die Bewertung dieser im Kontext ihrer Anwendbarkeit in der gewünschten Domäne. Das bedeutet auch, dass Einschränkungen und Grenzen in Bezug auf gewünschte Anforderungen im Kontext der Domäne untersucht werden. Die folgenden nicht funktionalen Anforderungen werden berücksichtigt.

NFA1: *Skalierbarkeit* wird in Bezug auf zwei unterschiedliche Betrachtungsweisen untersucht. Im Kontext der spezifischen FF1 wird die Fähigkeit des entwickelten Konzepts zur Unterstützung mehrerer parallel ausgeführter Dienste in Stationsgeräten evaluiert. Die Skalierung der Dienste in Bezug auf die unterstützten Stationen, Geräte und Größe des Netzgebiets, bezieht sich auf das Verwaltungskonzept.

NFA2: *Flexibilität* evaluiert die Möglichkeit der Verwendung unterschiedlicher Dienste auf den gleichen Stationsgeräten im Kontext des Virtualisierungskonzepts. Eine weitere Betrachtungsweise untersucht die Anwendbarkeit des gesamten Engineeringprozesses mit allen Artefakten in Bezug auf die Stationsunabhängigkeit.

NFA3: *Interoperabilität* wird im Kontext der herstellerunabhängigen Nutzung von Diensten im gesamten Prozess evaluiert. Außerdem wird evaluiert, ob Einschränkungen in Bezug auf standardisierte Kommunikation und damit die Interoperabilität zu anderen Stationsgeräten bestehen.

NFA4: *Wartbarkeit* findet Anwendung in der Evaluation des Virtualisierungs- und Verwaltungskonzepts mit Bezug auf die Unterstützung bei der Umsetzung von Änderungen der Dienste.

1.7 Methodischer Ansatz

Die Bearbeitung dieser Dissertation wurde durch den in Abbildung 1.2 dargestellten Prozess begleitet. Die grundlegende Struktur basiert auf der Design Science Research Methodology (DSRM) nach [16]. Der methodische Ansatz beschreibt insgesamt sechs Phasen, von der Problemidentifizierung bis zur Dissemination der Ergebnisse. Die Einordnung der Phasen in die einzelnen Abschnitte dieser Arbeit wird nachfolgend beschrieben.

Phase I & II sind beide in den Kapiteln 1 und 2 abgedeckt. Die erste Phase bezieht sich dabei auf die grundlegende Untersuchung in einem Forschungsbereich sowie die Identifizierung von Herausforderungen und Forschungsbedarf. Darunter fällt auch ein großer Teil der Ermittlung zum Stand der Technik, dem Aufbau von Fachwissen und der Expertise in einer Domäne. In Bezug auf diese Arbeit wurden CPES, Stationsautomatisierung und Engineeringprozesse im Kontext der Digitalisierung und der Notwendigkeit zur Bereitstellung von Diensten beschrieben. Die daraus resultierenden Forschungsfragen und angestrebten Zielsetzungen in Abschnitt 1.6 entsprechen der zweiten Phase im Prozess der DSRM. Unabhängig von der allgemeinen Ermittlung zum Stand der Technik erfolgt eine detaillierte Analyse der thematischen Schwerpunkte entsprechend der spezifischen Forschungsfragen in den Kapiteln 3, 4 und 5. Diese Informationen werden durch ein separates Grundlagenkapitel unterstützt, in dem unterschiedliche Konzepte, Prozesse und Technologien im Bereich der Informationssysteme erläutert werden.

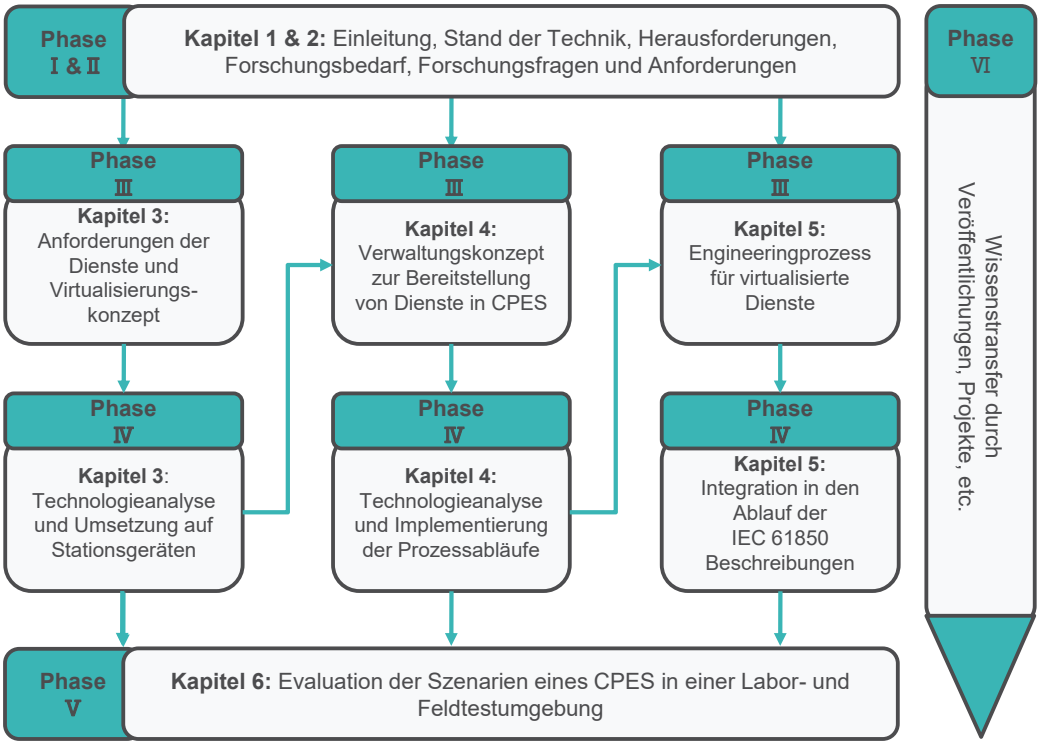

Abb. 1.2.: DSRM-Prozess [16], angewendet auf die Struktur dieser Dissertation.

Phase III entspricht der Entwurfs- und Entwicklungsphase im verwendeten methodischen Ansatz. Die Kapitel 3, 4 und 5 durchlaufen jeweils diese Phase. Das bedeutet, dass jedes Kapitel der Entwurfs- und Entwicklungsphase eines Artefakts entspricht. Im ersten Schritt erfolgt die Beschreibung von Diensten aus der Stationsautomatisierung von Verteilnetzen. Darauf aufbauend wird ein Virtualisierungskonzept für Stationsgeräte entworfen. Im nächsten Kapitel erfolgt die Abstraktion von einzelnen Stationen und deren Hardware auf die allgemeine Verwaltung von Diensten im Verteilnetz. In Abhängigkeit vom Virtualisierungskonzept wird im Kapitel 4 ein Verwaltungskonzept zur Beschreibung, Konfiguration und Bereitstellung von Diensten über mehrere Stationen bereitgestellt. Erneut findet die Entwurfs- und Entwicklungsphase in dem Kapitel 5 statt. Dieses Kapitel ergänzt einen bestehenden standardisierten Engineeringprozess um Eigenschaften zur Verwaltung von Diensten im Verteilnetz. Neben der Adaption einmaliger Prozesse werden kontinuierliche Überwachungs- und Steuerungsmaßnahmen ergänzt.

Phase IV weist die gleiche Struktur wie Phase III auf und beschreibt den Kontext und die Umsetzung der Artefakte. Für die Artefakte Virtualisierungs- und Verwaltungskonzept wurde jeweils eine technologische Analyse durchgeführt. Neben der Analyse und Identifizierung geeigneter Technologien wurden Technologien aus ver-

schiedenen Bereichen auf ihre Eignung im Kontext eines CPES untersucht. Das Virtualisierungskonzept wurde auf die Hardware eines Herstellers für Prüf- und Messtechnik umgesetzt. Das Verwaltungskonzept wurde als zentrale Einheit auf die Serverinfrastruktur einer Laborumgebung im OFFIS - Institut für Informatik und in der ISO 27001 zertifizierten Umgebung eines Netzbetreibers umgesetzt. Das dritte Artefakt, der Engineeringprozess, wurde aufbauend auf dem IEC 61850 Engineeringprozess entwickelt und um fehlende Aspekte erweitert. Die Umsetzung ist vielschichtig, da Teile des beschriebenen Prozesses durch Technologien der beiden Konzepte umgesetzt werden, während andere Bereiche, wie die Konfigurationsbeschreibungen, in den dazugehörigen Beschreibungssprachen erweitert wurden. Inkompatibilitäten zwischen Technologien und IEC 61850 Beschreibungen mussten durch eigene Implementierungen ergänzt werden. Unter Beachtung der in Abbildung 1.2 beschriebenen Phasen III & IV erfolgt die Erarbeitung und Umsetzung eines Artefakts auf Basis von Informationen aus der ersten Phase. Die Bearbeitung der Artefakte und der damit verbundenen Forschungsfragen erfolgt horizontal von links nach rechts in den Kapiteln 2-4. Zusätzlich fließen die Artefakte und deren Umsetzungen in die Bearbeitung der fünften Phase ein.

Phase V entspricht der Evaluationsphase, die in Kapitel 6 dieser Arbeit erfolgt. In diesem Kapitel werden zwei unterschiedliche Testumgebungen betrachtet. Die Laborumgebung untersucht Szenarien, die unabhängig von den spezifischen Eigenschaften der Stationsautomatisierung und Netzbetreiberbedingungen durchgeführt werden können. Szenarien, die eine realistischere Umgebung erfordern, wurden in einem Feldtestgebiet eines Netzbetreibers evaluiert und beinhalten einen kleinen Mittelspannungsabschnitt mit mehreren Ortsnetzstationen. Jede Station wurde digital aufgerüstet und die Konzepte auf die Stationsgeräte umgesetzt. Die Anbindung der digitalisierten Ortsnetzstationen wurde durch eine separate gesicherte Netzwerkstruktur bereitgestellt. Neben den funktionalen Anforderungen, die bereits in der Umsetzungsphase evaluiert werden, erfolgt im Kapitel 6 die Bewertung der funktionalen und nicht funktionalen Anforderungen.

Phase VI beschreibt den disziplinären Wissenstransfer, der im Rahmen dieser Arbeit in unterschiedlichen Formen erfolgt ist. Primär erfolgte der Wissenstransfer durch Veröffentlichungen auf Konferenzen, die sich entsprechend des CPES auf Energiesysteme und Informationssysteme beziehen. Eine Auflistung der *eigenen Veröffentlichungen* ist am Ende zu finden. Während sich die Arbeiten [17], [18] und [7] mit den Herausforderungen zukünftiger Energiesysteme und dem Bedarf nach Automatisierungssystemen beschäftigten, beziehen sich die Arbeiten [8], [19] und [4] auf den Entwurf und die Umsetzung eines Virtualisierungs- und Verwaltungskonzepts. Der Engineeringprozess wurde zusammen mit den Ergebnissen der

vorherigen Konzepte in den Arbeiten [11], [20] und [21] veröffentlicht. Ergänzt wurde diese Art des Wissenstransfers durch die Begleitung der Forschungsprojekte i-Automate [12] und i-Autonomous [13], die sowohl thematisch viele Überschneidungen und Synergien bereitgestellt haben als auch die Evaluation im Feldtestgebiet des Netzbetreibers beinhalteten.

2 Grundlagen

Dieses Kapitel beschreibt einige Grundlagen zu wesentlichen Konzepten, Technologien und Schnittstellen. Diese Informationen stellen ein einheitliches Verständnis für die nachfolgenden Kapitel bereit. Der Abschnitt 2.1 beschreibt die grundlegenden Konzepte eines Engineeringprozesses und die genutzten Schnittstellen auf Basis des IEC 61850 Standards. Diese Konzepte werden in dieser Arbeit um weitere Prozessschritte, basierend auf den Konzepten der Virtualisierung (Abschnitt 2.2) und des Edge-Computing (Abschnitt 2.3.1), erweitert.

2.1 IEC 61850 - Konzepte und Schnittstellen

Der Standard IEC 61850 gilt als internationaler Standard für die Beschreibung von Stationsautomatisierungssystemen und die standardisierte Kommunikation innerhalb von Automatisierungs- und Schutzsystemen der Energiewirtschaft. Ziel ist es, die Interoperabilität zwischen Geräten und Systemen verschiedener Hersteller zu ermöglichen. Im Folgenden wird auf die Kommunikation über IEC 61850 und das Engineering von Stationen eingegangen.

2.1.1 IEC 61850 Kommunikation

Der Standard lässt sich in drei unterschiedliche Kommunikationsdienste aufteilen, die je nach Anwendungsfall für verschiedene Kommunikationsanforderungen ausgelegt sind.

Manufacturing Message Specification (MMS):
Dieser Dienst basiert auf einer Client-Server-Struktur und ermöglicht den Zugriff auf Datenobjekte innerhalb eines IEDs. Ein Anwendungsgebiet dieser Kommunikation ist der Austausch von Daten zwischen dem Leitsystem und den Stationsgeräten.

Generic Object-Oriented Substation Event (GOOSE):
Im Gegensatz zu MMS basiert dieser Kommunikationsdienst auf einer Peer-to-Peer-Kommunikation, wodurch Daten mit hoher Frequenz und Geschwindigkeit gesendet

werden können. Diese Informationen werden insbesondere für zeitkritische Anwendungen genutzt, wie es oft bei Schutzfunktionen der Fall ist. Diese Art der Kommunikation ist auf die Kommunikation innerhalb einer Station begrenzt.

Sampled Measured Values (SMV):
Dieser Kommunikationsdienst überträgt zeit-gestempelte Messwerte, z.B. Strom- und Spannungsdaten, innerhalb einer Station. Dieser Dienst zeichnet sich durch hohe Genauigkeit und zeitliche Präzision aus und wird normalerweise für Echtzeitüberwachung und -steuerung genutzt.

2.1.2 IEC 61850 Engineering

IEC 61850 ist mehr als nur ein Kommunikationsdienst. Dieser Standard spezifiziert auch Beschreibungen und Prozesse zur Konfiguration und Parametrierung von IEDs in Stationen. Die Systembeschreibung erfolgt in einer XML-basierten Sprache, der System Configuration Language (SCL). Diese Sprache ermöglicht die einheitliche und standardisierte Beschreibung von Geräten, sodass eine Interoperabilität zwischen Geräten verschiedener Hersteller erreicht werden kann.

Dieser Prozess wird durch einen strukturierten Engineeringprozess [9] unterstützt und mithilfe unterschiedlicher Engineeringwerkzeuge [22] durchgeführt. Das Engineering durchläuft dabei verschiedene Phasen, die im Kapitel 5 dieser Arbeit detaillierter erläutert werden.

2.2 Virtualisierung

Virtualisierung ist eine Methode in der Informationstechnik, bei der Ressourcen so abstrahiert werden, dass ihre Auslastung optimiert werden kann [23]. Unter Anderem wird dazu die Entkopplung von Software und Hardware genutzt, die entsprechende Unabhängigkeiten und Flexibilitäten bei der Verwendung von Ressourcen ermöglicht. Virtualisierung umfasst mehrere grundsätzlich verschiedene Konzepte und Technologien. Im Rahmen dieser Arbeit wird die Betriebssystem-Virtualisierung (auch als Containerisierung bekannt) im Detail beschrieben. Weitere Konzepte im Bezug zur Betriebssystem-Virtualisierung umfassen Partitionierung, Hardware-Emulation, Anwendungsvirtualisierung, vollständige Virtualisierung, Paravirtualisierung und Hardware-Virtualisierung. Die meisten Virtualisierungstechniken erlauben es, eine Virtuelle Maschine (VM) zu erzeugen. Jede VM kann eine Anwendung mit Betriebssystem in einer isolierten Umgebung auf einer physischen Hardware darstellen. Aus

Sicht der Anwendungen existiert ein vollwertige Ausführungsumgebung mit eigenen Komponenten und Ressourcen, ohne die darunterliegenden Virtualisierungstechniken wahrzunehmen.

2.2.1 Betriebssystem-Virtualisierung

Die Betriebssystem (BS)-Virtualisierung startet im Gegensatz zu anderen Virtualisierungstechniken kein zusätzliches Betriebssystem, sondern erzeugt isolierte Laufzeitumgebungen. Anbieter von Technologien zur Bereitstellung dieser Laufzeitumgebungen nutzen in der Regel den Begriff Container. Alle Container verwenden denselben Betriebssystemkern als Grundlage. Anwendungen lassen sich direkt auf dem BS ausführen oder in der isolierten Laufzeitumgebung. Die Abbildung 2.1 zeigt die unterschiedlichen Schichten einer BS-Virtualisierung und die Verortung einer Anwendung. Die Sicht einer Anwendung ist auf einen Container begrenzt, sodass andere Anwendungen nur innerhalb desselben Containers gesehen werden.

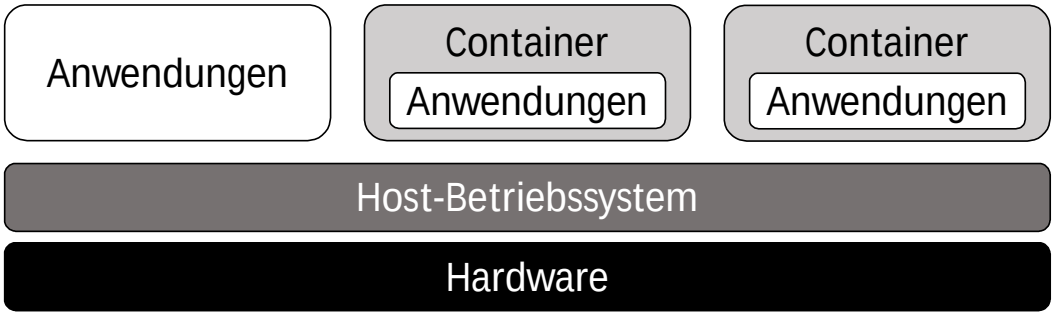

Abb. 2.1.: Darstellung einer Betriebssystem (BS)-Virtualisierung

Das Konzept der BS-Virtualisierung ist in der Literatur auch unter der Bezeichnung leichtgewichtige Virtualisierung bekannt. Durch die einheitliche Nutzung der Verwaltungsschicht (dem Betriebssystemkern) besteht ein geringer Verwaltungsaufwand für die einzelnen Container. Im Vergleich zur Hypervisor-Virtualisierung 2.2.2 besteht eine Beschränkung auf nur einen Betriebssystemkern. Dadurch ist es nicht möglich, verschiedene Betriebssysteme gleichzeitig auf einer physischen Hardware zu verwenden. Stattdessen werden mehrere unabhängige Instanzen eines Betriebssystems gestartet.

Einige interessante Szenarien für dieses Virtualisierungskonzept sind im Buch [23] beschrieben. Darunter der Betrieb von Anwendungen mit einem Bedarf an hoher Sicherheit und isolierter Umgebung sowie die Möglichkeit, den Zustand eines Betriebssystems nicht durch Paketabhängigkeiten zu beeinflussen.

2.2.2 Hypervisor-Virtualisierung

Der Hypervisor ist eine Softwareschicht, die das Ausführen mehrerer unabhängiger Laufzeitumgebungen auf einer einzelnen Ausführungseinheit ermöglicht [24]. Diese Softwareschicht erlaubt den simultanen Betrieb verschiedener Gastsysteme auf einem Hostsystem und verwaltet die Zuteilung der Ressourcen. Es wird zwischen zwei Typen der Hypervisor-Architektur unterschieden. Der Hypervisor des Typs 1 wird direkt auf der Hardware betrieben und ist auch als nativer Hypervisor bekannt. Der Hypervisor des Typs 2 wird als Applikation direkt auf dem Host-BS ausgeführt. Die Abbildung 2.2 zeigt die unterschiedlichen Hypervisor-Architekturen.

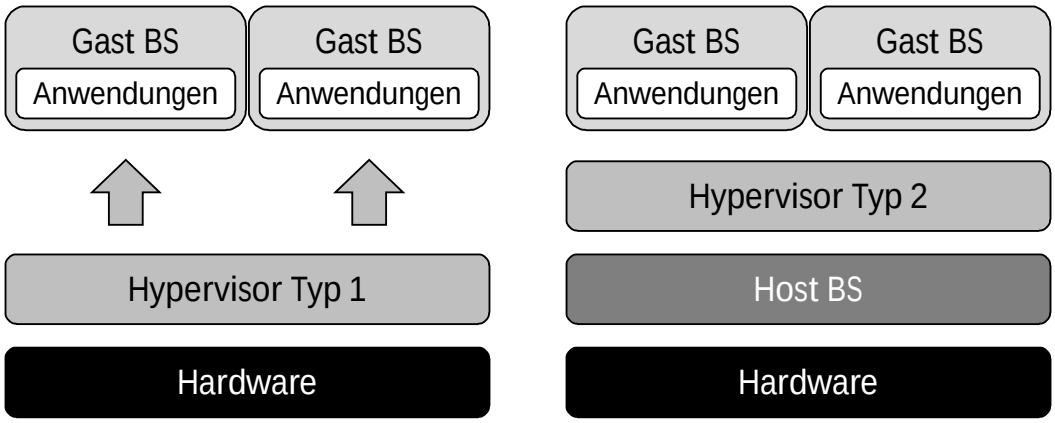

Abb. 2.2.: Darstellung der Virtualisierung mittels Hypervisor Typ 1 und Hypervisor Typ 2.

Im Vergleich zur BS-Virtualisierung ist die Hypervisor-Architektur unabhängig vom Betriebssystem des Hosts und bietet die Möglichkeit, verschiedene Gast-Betriebssysteme gleichzeitig auszuführen.

2.3 Konzepte und Paradigmen der Informatik

In der Informatik finden unterschiedliche Konzepte und Paradigmen Anwendung. Im Folgenden wird auf die Themen Cloud- und Edge-Computing eingegangen.

2.3.1 Cloud Computing

Die Digitalisierung in allen Bereichen der Gesellschaft führt zu erhöhten Anforderungen an Informationssysteme, insbesondere einem wachsenden Bedarf an Rechenleistung und Speicherkapazität. Klassische Modelle der Datenverarbeitung, wie

das Client/Server-Modell, können diesen wachsenden Herausforderungen teilweise nicht mehr gerecht werden [25]. Das National Institute of Standards and Technology (NIST) definiert Cloud Computing wie folgt:

Definition 1 (Cloud Computing)**.** *„Cloud Computing ist ein Modell für den allgegenwärtigen, bequemen und bedarfsgerechten Zugang zu einer gemeinsam genutzten Menge von konfigurierbaren Computerressourcen (z.B. Netzwerke, Server, Speicher, Anwendungen und Dienste), die mit minimalem Verwaltungsaufwand und ohne Interaktion mit dem Dienstanbieter schnell bereitgestellt und freigegeben werden können.“*[1] *[26].*

Das NIST beschreibt zusätzlich wesentliche Merkmale, Dienstmodelle und Einsatzmodelle, die das Cloud Computing näher charakterisieren. Als Merkmale werden die folgenden fünf aufgeführt:

Selbstbedienung auf Abruf: Ein Nutzer kann Computerressourcen automatisch nach Bedarf anfordern, ohne dass eine menschliche Interaktion mit dem Dienstanbieter erforderlich ist.

Umfangreicher Netzwerkzugang: Es besteht die Möglichkeit eines Zugriffs über Netzwerke und standardisierte Mechanismen.

Ressourcenbündelung: Abhängig vom Nutzerbedarf können Computerressourcen gebündelt werden, um mehrere Anfragen unterschiedlicher Nutzer parallel zu bearbeiten. Es können sowohl virtuelle als auch physikalische Ressourcen dynamisch zugewiesen werden. Zudem besteht aus der Sicht eines Nutzers eine räumliche Unabhängigkeit, sodass dieser weder Kontrolle noch exaktes Wissen über die räumliche Verortung der Ressourcen hat.

Schnelle Elastizität: Elastizität beschreibt die Fähigkeit, abhängig vom Bedarf flexibel und teilweise automatisiert Ressourcen zuzuweisen und freizugeben. Für einen Nutzer kann dies den Eindruck einer unbegrenzten Hochskalierung zu jeder Zeit erwecken.

Messbare Dienstleistung: Cloud-Systeme steuern und optimieren automatisch die Nutzung der Ressourcen durch eine Messfunktion auf einer für die Art des Dienstes geeigneten Abstraktionsebene (z.B. Speicher, Verarbeitung, Bandbreite und aktive Benutzerkonten). Die Ressourcennutzung kann überwacht, kontrolliert und gemeldet werden, sowohl für den Anbieter als auch für den Nutzer des Dienstes.

[1] Teilweise wörtlich aus dem Englischen übersetzt.

Die Abbildung 2.3 zeigt die unterschiedlichen Varianten der Dienst- und Einsatzmodelle. Software as a Service (saas) (deutsch: Software als Dienstleistung [2]), Platform as a Service (PaaS) (deutsch: Plattform als Dienstleistung [3]) und Infrastructure as a Service (IaaS) (deutsch: Infrastruktur als Dienstleistung [4]) beschreiben verschiedene Dienstleistungen, die einem Nutzer bereitgestellt werden.

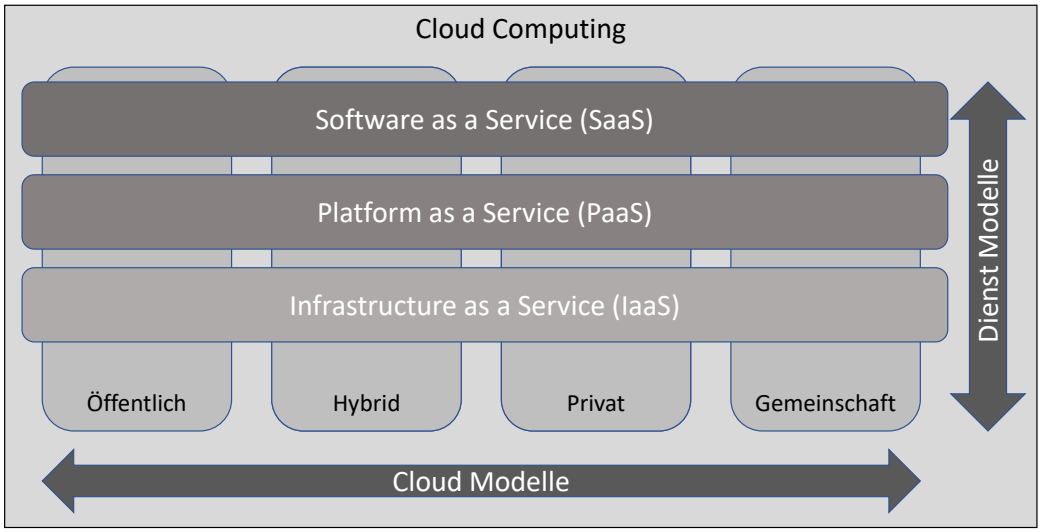

Abb. 2.3.: Dienst- und Einsatzmodelle des Cloud Computings

2.3.2 Edge Computing

Edge Computing unterscheidet sich vom traditionellen Cloud Computing, da es die Berechnungen nicht zentral in großen Rechenzentren ausführt, sondern näher an den Ursprung der Daten bringt, also an den „Rand“ (Edge) des Netzwerks [27]. Im Anwendungsfall der Stationsautomatisierung bedeutet dies, dass die Dienste zur Berechnung in die Stationen verlegt werden. Eine Definition von Edge Computing ist in der Arbeit [28] dargestellt und beschreibt Edge Computing wie folgt:

Definition 2 (Edge Computing)**.** *Edge Computing ist ein neues Computermodell, das Ressourcen in geografischer oder netzwerktechnischer Nähe zum Benutzer vereint, um Anwendungen mit Rechen-, Speicher- und Netzwerkkapazitäten zu versorgen.*

[2] In dieser Arbeit wird der englische Fachbegriff Software as a Service (saas) verwendet.
[3] In dieser Arbeit wird der englische Fachbegriff Platform as a Service (PaaS) verwendet.
[4] In dieser Arbeit wird der englische Fachbegriff Infrastructure as a Service (IaaS) verwendet.

Im Rahmen dieser Arbeit wird Edge Computing unter dem Verständnis der Nutzung von Systemressourcen einzelner Stationsgeräte zur Berechnung netzdienlicher Dienste betrachtet.

3 Virtualisierte Dienste in Stationsgeräten

Die Analyse, Konzeption und Umsetzung virtualisierter Dienste in Stationsgeräten unter Berücksichtigung bestehender Infrastrukturen, insbesondere die Verwendung bestehender Hardware-Geräte, wird thematisch in fünf Abschnitten erläutert. Im ersten Schritt erfolgt die Analyse und Beschreibung von Diensten, die in einer Station Anwendung finden oder im Rahmen der Digitalisierung notwendig werden. Der Abschnitt 3.2 beschreibt Konzepte zur Virtualisierung von Diensten und setzt diese in Bezug auf Anforderungen an die Hardware-Infrastruktur einer Station um. Die technische Umsetzung der Konzeption erfolgt in drei Phasen: die Analyse verfügbarer Technologien, die Implementierung und die Parametrierung. Abschließend erfolgt eine Zusammenfassung.

3.1 Dienste in Stationsgeräten

Die Beschreibung von Diensten ist vielfältig und beinhaltet neben der gewünschten Funktionalität auch Prozesse des kommunikativen Austauschs. Die folgende Definition und Beschreibung eines Dienstes wird in dieser Arbeit verwendet.

Definition 3 (Dienst)**.** *„Ein Dienst umfasst die Ausführung einer oder mehrerer Funktionen (z.B. Funktionen der Stationsautomatisierung) sowie die damit verbundenen Aufgaben wie Vorbereitungen, Kommunikation, Konfiguration und Parametrierung.*

Der Bezug zur Stationsautomatisierung wird in den nächsten Abschnitten durch verwandte Arbeiten dargestellt und relevante Dienste werden identifiziert. Anforderungen an die technische Umsetzbarkeit werden abgeleitet und als Basis zur Virtualisierung von Diensten genutzt.

3.1.1 Verwandte Arbeiten

In der Arbeit [17] werden unter dem Begriff *Higher Virtual Function* virtualisierte Smart Grid-Funktionen vorgestellt, darunter Netzzustandsschätzung, Spannungsregelung und Betriebsführung. Eingebettet wurden die Smart-Grid-Funktionen in ein Konzept, das als Higher Function Virtualization (HFV) [1] bezeichnet wird. Der Kerninhalt der Arbeit bezieht sich auf die Darstellung einer zukünftigen Supervisory Control and Data Acquisitions (SCADA) [2]-Architektur für IKT-basierte Energiesysteme. Eine wichtige Hervorhebung der Arbeit ist die Verwendung von Parametern wie Quality of Services (QoS) [3]-Informationen (Verzögerungen, Durchsatz etc.) und Zustandsinformationen von Geräten (Rechenleistung, Arbeitsspeicher und Speicherplatz) aus den Bereichen der IKT und Operational Technology (OT)(deutsch: Operative Technologie [4]). Diese Parameter bieten ein hohes Potenzial zur Erhöhung der Flexibilität im Gesamtsystem durch die Verwendung unterschiedlicher Konzepte wie Software Defined Networking (SDN) [5], Network Function Virtualization (NFV)[6] und HFV. Es gilt zu beachten, dass HFV (auch Grid Function Virtualization (GFV)) ein weniger bekanntes Konzept ist und in den folgenden Arbeiten Berücksichtigung findet [17, 19, 8]. Im Rahmen dieser Arbeit werden die genannten Zustandsinformationen berücksichtigt und im Zusammenhang mit Diensten angewendet.

Die Bereitstellung von Diensten in CPES unter Verwendung von Virtualisierung wird in [8] untersucht. Zusätzlich zur Funktionalität der Zustandsschätzung und Spannungsregelung werden in Form von Teilfunktionalitäten einer Remote Terminal Unit (RTU) (deutsch: Fernwirkgerät [7]) virtualisierte Dienste bereitgestellt. In dieser Arbeit werden Dienste als eine Kette von Funktionen dargestellt. Genauer gesagt, umfasst eine Zustandsschätzung die Funktionen Datenbeschaffung, Verarbeitung und Steuerung.

Unter den Untersuchungsaspekten einer Smart-Grid-Automatisierungsarchitektur werden in der Dissertation von Bauernschmitt et al. [29] verschiedene Funktionen zur Überwachung, Steuerung, Automatisierung und zum Schutz eines Smart Grids vorgestellt. Die Abbildung 3.1 listet die von Bauernschmitt et al. untersuchten Funktionen auf und ordnet sie in Bezug auf relevante Anforderungen ein. Zusätzlich präsentiert Abbildung 3.2 eine Kategorisierung von Funktionen basierend auf

[1] In dieser Arbeit wird der englische Fachbegriff Higher Function Virtualization (HFV) verwendet.
[2] In dieser Arbeit wird der englische Fachbegriff Supervisory Control and Data Acquisitions (SCADA) verwendet.
[3] In dieser Arbeit wird der englische Fachbegriff Quality of Services (QoS) verwendet.
[4] In dieser Arbeit wird der englische Fachbegriff Operational Technology (OT) verwendet.
[5] In dieser Arbeit wird der englische Fachbegriff Software Defined Networking (SDN) verwendet.
[6] In dieser Arbeit wird der englische Fachbegriff Network Function Virtualization (NFV) verwendet.
[7] In dieser Arbeit wird der englische Fachbegriff Remote Terminal Unit (RTU) verwendet.

	Funktion	Messwerte	Geschwindigkeit / Qualität Messwerte	Zeitverhalten	Kommunikation	Weiteres
Monitoring	Visualisierung	U, I, P, Q	1-60 s		Leitstellen-anbindung	
	Power Quality	U, I	25./50. Harmonische			Zeitsynchronisation
	Online-SE	$\underline{U}$, $\underline{I}$	100 ms-60 s			
	Offline-SE	U, I, P, Q	1-15 min			
	Topologie	Schalterstellung	1 s- 15 min			
Schutz	Überstromzeit-schutz	I oder $\underline{I}$ und $\underline{U}$ (Fehlerrichtung)	10-20 ms	Deterministisch, Auslösezeit 300 ms	Leitstellen-anbindung	Leistungsschalter
	Distanzschutz	$\underline{I}$ und $\underline{U}$	10-20 ms	Deterministisch, Auslösezeit 100 ms		Leistungsschalter
	Differenzial-schutz	$\underline{I}$ oder i	10-20 ms	Deterministisch, Auslösezeit 100 ms	Deterministisch	Leistungsschalter, Zeitsynchronisation
	Frequenzschutz/ Lastabwurf	f	Genauigkeit 30 mHz	Deterministisch, Auslösezeit 150 ms	ggf. Fernauslesung	Leistungsschalter
Fehleranzeiger	Konventionell	I oder $\underline{I}$ und $\underline{U}$	20 ms, ggf. 5./7. Harmonische		ggf. Fernauslesung	
	Ortung, multiple Messorte	$\underline{I}$ und $\underline{U}$ ($\underline{Z}$-Bestimmung)	10-20 ms			Zeitsynchronisation
	Erdschluss-wischer	i und u	24 kHz			
Regelung	Traforegelung	U	1-10 s			
	Q-/P-Regelung Flexibilitäten	U, I, P, Q	100 ms-1 s		Anbindung an Flexibilität	

Abb. 3.1.: Anforderungen ausgewählter Funktionen [29]

ihrem Ausführungsort und den dazugehörigen Kommunikationsbeziehungen. Ergänzt wird auch die Betrachtung eines hierarchischer Ansatz, in der Arbeit [30], zur Verteilung von Schutzfunktionen über Ortsnetzstationen hinweg. Diese Funktionen mit einer zeitlichen Auflösung von wenigen Sekunden und niedriger, wie die Schutzfunktionen, sind im lokalen System einzuordnen. Andere Funktionen, wie die Zustandsschätzung (angegeben als online-SE/offline-SE), welche klassisch zentral genutzt werden, können auch für verteilte oder dezentrale Systeme Anwendung finden. Die Kategorie Messwerte zeigt, dass nahezu alle Funktionen eine Abhängigkeit zur Sensorik oder anderen Funktionen als Datenbeschaffung beinhalten. Im

Rahmen dieser Arbeit können mögliche Dienste der Stationsgeräte und deren Anforderungen abgeleitet sowie die Kategorisierung des Ausführungsortes und verknüpfte Kommunikationsverbindungen genutzt werden.

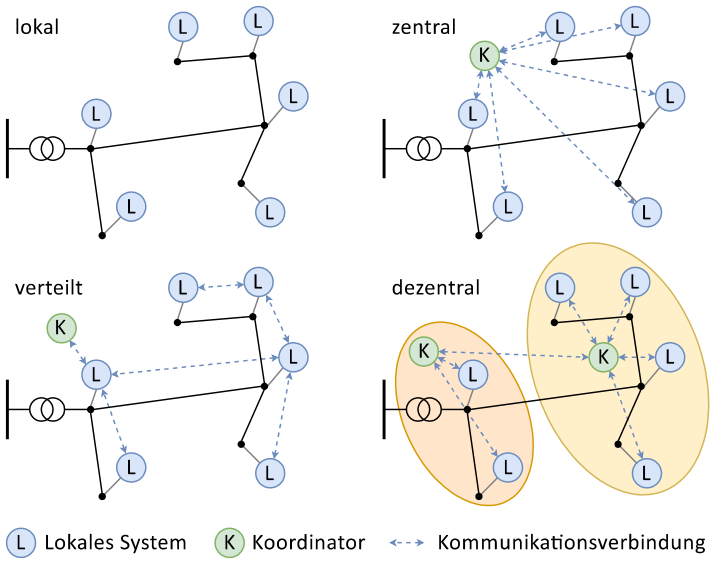

Abb. 3.2.: Kategorisierung anhand des Ausführungsortes [29]

Im Rahmen des Projekts i-Autonomous [13] wurden Expertengespräche mit Fachleuten von Verteilnetzbetreibern, Industrie und Forschungsinstituten geführt, in denen Schutz- und Automatisierungsfunktionen für Stationsgeräte untersucht wurden. Ziel der Untersuchung war die Identifizierung möglicher Schutz- und Automatisierungsfunktionen sowie die Einschätzung des Aufwand/Nutzen-Verhältnisses. Abbildung 3.3 enthält eine Liste aller untersuchten Funktionen und eine grafische Darstellung des Aufwand/Nutzen-Verhältnisses. Die in Blau eingezeichnete Diagonallinie wurde als Prioritätsgrenze innerhalb des Projekts festgelegt, d. h. alle Funktionen oberhalb dieser Grenze wurden genauer untersucht und ein Teil davon im Rahmen des Projekts in einer Feldteststudie umgesetzt.

Die Erkenntnisse aus den verwandten Arbeiten bezüglich des Bedarfs, Nutzens und der Umsetzbarkeit von Schutz- und Automatisierungsfunktionen in Stationsgeräten unterstützen die Motivation und Problemstellung dieser Arbeit. Zur Bereitstellung der Funktionen als virtualisierte Dienste werden im Folgenden die Anforderungen an die technische Umsetzung der Dienste überprüft.

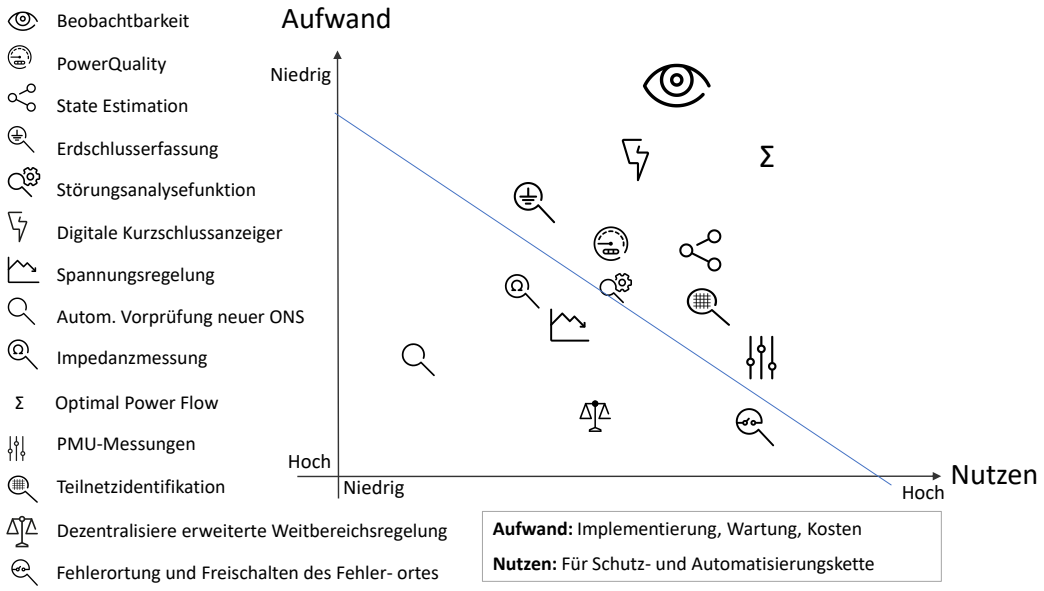

Abb. 3.3.: Schutz- und Automatisierungsfunktionen in Stationsgeräten [13]

3.1.2 Anforderungen an Dienste

Die Bereitstellung von Diensten auf Stationsebene erfordert neben den Anforderungen der darin enthaltenen Prozesse, wie beispielsweise Datenerfassung, algorithmische Funktionalität und Datenübertragung, auch Vorgaben für die Implementierung eines Dienstes innerhalb einer Station auf einer Hardware-Infrastruktur. Diese Anforderungen wurden im Kontext der Forschungsprojekte i-Automate [12] und i-Autonomous [13] sowie durch die Analyse einschlägiger Literatur identifiziert. Zu jeder Anforderung erfolgt eine präzise Beschreibung. Die Erfüllung dieser Anforderungen wird in den Kapiteln 3.2 und 5 im Hinblick auf das Konzept der Virtualisierung und in Bezug auf die Verwendung in einem Engineeringprozess behandelt.

Hardwareunabhängigkeit

Diese Anforderung bezieht sich auf die Gestaltung der Software, sodass diese auf unterschiedlichen Hardwareplattformen ohne wesentliche Änderungen oder Anpassungen ausgeführt werden kann. Genauer gesagt sollte eine Software-Anwendung auf verschiedenen Computertypen, Prozessorarchitekturen oder Betriebssystemen reibungslos funktionieren, ohne spezifische Hardware-Abhängigkeiten berücksichtigen zu müssen. Zusammenfassend lässt sich die Hardwareunabhängigkeit als Konzept beschreiben, das die Übertragbarkeit von Software gewährleistet. Es ermöglicht Unternehmen und Entwicklern, ihre Software-Anwendungen auf eine Vielzahl von

Geräten und Infrastrukturen einzusetzen. Technologien wie Virtualisierung, plattformunabhängige Programmiersprachen und standardisierte Schnittstellen ermöglichen die Umsetzung der Hardwareunabhängigkeit. Die konzeptionellen Grundlagen zur Virtualisierung sind im Kapitel 2.2 aufgezeigt. Die Betrachtung der Hardwareunabhängigkeit innerhalb dieser Arbeit und entsprechend der Anforderungen aus dem Forschungsprojekt i-Autonomous bezieht sich auf die Übertragbarkeit von Diensten zwischen vorausgewählten Herstellern von Stationsgeräten. Technologien der plattformunabhängigen Programmiersprachen finden keine Anwendung, sodass keine vollständige Unabhängigkeit unterschiedlicher Prozessorarchitekturen gewährleistet werden kann.

Beobachtbarkeit
Generell bezeichnet „Beobachtbarkeit“ die Fähigkeit eines Systems, klare und umfassende Informationen über Zustand, Leistung, Aktivitäten oder Fehler des Systems bereitzustellen. Die erfassten Informationen ermöglichen die Fehlersuche, Diagnose und Wartung innerhalb des Systems. Die Granularität der Beobachtbarkeit lässt sich von einzelnen Teilsystemen wie Komponenten, Funktionen, Diensten usw. bis hin zu ganzen Systemen anwenden. Technologien zur Überwachung, Protokollierung, Verfolgung, Fehlererkennung und Telemetrie können genutzt werden, um diese Informationen zu erfassen. Im Kontext der Virtualisierung von Diensten werden alle Informationen betrachtet, die den Zustand der einzelnen Dienste oder der zugrunde liegenden Hardware-Infrastruktur darstellen. Zusätzlich werden Informationen zum Prozessablauf der Dienste und die Verfügbarkeit von Systeminformationen für den Engineeringprozess genutzt. Eine umfassende Liste aller relevanten Informationen wird im Abschnitt 3.5 zur technischen Umsetzung virtualisierter Dienste präsentiert.

Automatische Updateprozesse & Plug-and-Play
Die Anforderung automatischer Updateprozesse bezieht sich auf Prozesse, die darauf abzielen, Softwareanwendungen und Systemkomponenten in regelmäßigen Intervallen, bei Verfügbarkeit oder durch manuelle Interaktion automatisch zu aktualisieren. Das bedeutet, dass alle Aufgaben der Identifizierung, Implementierung, Überwachung und Überprüfung werden mithilfe von Technologien und den dazugehörigen automatisierten Prozessen durchgeführt. Die Identifizierung umfasst die Überwachung von Sicherheitsmeldungen, Ankündigungen von Softwareherstellern und anderen Quellen, um die Verfügbarkeit neuer Updates zu erkennen. Die Implementierung beinhaltet Prozesse zur Bereinigung nicht notwendiger Software. Die Installation erfolgt durch Skripte und die Ausführung vordefinierter Konfigurationen. Anschließend wird durch Überwachungs- und Überprüfungsprozesse festgestellt, ob das Update vollständig und ordnungsgemäß installiert wurde. Diese

Prozesse können durch Validierungstests ergänzt werden. Die Anforderung an eine Plug-and-Play-Lösung bezieht sich im Rahmen dieser Arbeit auf die Fähigkeit, Softwareanwendungen auf die Hardware-Infrastruktur von Stationsgeräten möglichst automatisiert installieren zu können. Im Vergleich zum automatischen Updateprozess unterscheidet sich die Plug-and-Play-Lösung hauptsächlich dadurch, dass eine neue Softwareanwendung bereitgestellt wird und nicht, wie beim Updateprozess, bestehende Softwareanwendungen verändert werden. Es ist wichtig zu beachten, dass nur Teile eines vollständigen Patchmanagements im Rahmen dieser Arbeit Anwendung finden, um den Anforderungen der Stationsautomatisierung zu entsprechen. Die technologische Ausgestaltung eines automatischen Updateprozesses mit einer Plug-and-Play-Lösung ist im Kapitel 5 beschrieben.

Verteilte Dienste

Die Ausführung von Diensten in einer verteilten Struktur, wie sie in Abbildung 3.2 dargestellt ist, entspricht den aktuellen Fragestellungen im Bereich der Stationsautomatisierung. In den Forschungsprojekten i-Automate [12] und i-Autonomous [13] wurde die Ausführung von Diensten in dieser Struktur untersucht. Eine verteilte Ausführung von Diensten bedeutet, dass Ressourcen und Komponenten über mehrere Standorte und Knotenpunkte hinweg angeordnet sind. Diese Verteilung kann sich auf geografische Standorte, verschiedene Ressourcen oder mehrere Instanzen innerhalb eines Netzwerks beziehen. Im Rahmen dieser Arbeit bezieht sich die verteilte Ausführung von Diensten auf die geografische Trennung von Stationen. Die Ziele einer solchen Verteilung sind vielfältig. Sie ermöglicht Lastausgleich, Redundanz, Skalierbarkeit und verbesserte Ausfallsicherheit in verteilten Systemen. Ein weiteres relevantes Kriterium bei der Stationsautomatisierung ist die Notwendigkeit, einige Dienste lokal aufgrund ihrer benötigten Reaktionsgeschwindigkeiten auszuführen, wie es beispielsweise bei Diensten mit Schutzfunktionen der Fall ist. Zusammenfassend betrachtet diese Arbeit insbesondere die Anforderung, Dienste auf unterschiedlichen Stationen zu verteilen und zu verwalten.

Fernwartbarkeit

Im Kontext der Stationsautomatisierung bezieht sich die Fernwartbarkeit auf die Fähigkeit, Systeme, Geräte und Ausrüstung in entfernten Stationen oder Anlagen aus der Ferne zu überwachen, zu konfigurieren und zu warten. Die Umsetzung der Fernwartbarkeit erfordert zuverlässige Netzwerkinfrastrukturen, geeignete Kommunikationsprotokolle, Sicherheitsmaßnahmen und Fernwartungssysteme. Im Zusammenhang mit Fernwartbarkeit können die Aspekte Überwachung und Diagnose, Fernkonfiguration, Fernwartung und Sicherheit betrachtet werden. In Bezug auf die Sicherheit beschreibt das Bundesamt für Sicherheit in der Informationstechnik in der aktuellen Fassung (Edition 2023) der IT-Grundschutz-Bausteine den Teil IND.3.2

Fernwartung im industriellen Umfeld [31]. Zielsetzung ist es, die Informationssicherheit bei der Fernwartung im industriellen Umfeld zu gewährleisten.

Kontinuierlicher Engineeringprozess
Die Anforderung an einen Engineeringprozess bezieht sich auf das Verständnis bestehender Engineeringprozesse in der Stationsautomatisierung und den darunterliegenden systematischen Ansatz, um automatisierte Steuerungs- und Überwachungssysteme zu implementieren und zu warten. Es wird Bezug auf die Beschreibung eines Engineeringprozesses der DKE AK 952.0.1 [9] und der Anforderungen an IEC 61850 Engineeringwerkzeuge [22] genommen. Im Rahmen dieser Arbeit müssen entwickelte Konzepte und verwendete Technologien auf den Engineeringprozess abgestimmt werden. Das kontinuierliche Verhalten, welches durch die Anforderungen der „Beobachtbarkeit“ und „Automatische Updateprozesse“ erfolgt, bedarf einer Erweiterung des Bestandsprozesses um kontinuierliche Teilprozesse. Eine genaue Betrachtung erfolgt im Kapitel 5.

3.2 Virtualisierung von Diensten

Die Verwendung von Virtualisierung bietet Vorteile und Eigenschaften, die zur Bereitstellung von Diensten auf Stationsebene genutzt werden können. Neben einer Beschreibung dieser Eigenschaften werden verwandte Arbeiten in diesem Bereich analysiert und ein Virtualisierungskonzept unter Berücksichtigung der Anforderungen für Dienste in Stationsgeräten vorgestellt. Im Anschluss erfolgt eine Analyse möglicher Technologien für die weitere Betrachtung im Rahmen dieser Arbeit. Die Grundlagen der Virtualisierung sind im Kapitel 2.2 beschrieben.

3.2.1 Eigenschaften von Virtualisierung

Einige hervorstechende Eigenschaften und Vorteile, die durch Virtualisierung bereitgestellt werden, sind in der Arbeit von Betzler et al. [32] beschrieben. Zusammen mit den Informationen aus dem Kapitel 2.2 wurden die folgenden Eigenschaften ermittelt:

Abstraktion von Ressourcen
Virtualisierung ermöglicht die Abstraktion von physischen Ressourcen wie Rechenleistung, Speicher und Netzwerk, sodass virtuelle Instanzen unabhängig von der darunterliegenden Hardware agieren können. Die Fähigkeit, physische Ressourcen zu separieren, ermöglicht die gemeinsame Nutzung durch die Ausführung mehrerer

virtueller Instanzen auf einer Hardware. Dadurch wird eine effizientere Verwaltung von Hardware-Ressourcen zwischen den virtuellen Instanzen ermöglicht.

Flexibilität
Virtuelle Umgebungen sind flexibel und können Ressourcen dynamisch bereitstellen oder neu konfigurieren, um sich den ändernden Anforderungen anzupassen. Diese Flexibilität unterstützt auch die Verwaltung von Ressourcen einer zugeordneten virtuellen Instanz.

Isolation und Sicherheit
Die Isolation von virtuellen Instanzen bedeutet, dass diese unabhängig voneinander arbeiten können und sich gegenseitig nicht beeinflussen. Zudem lässt sich ein sicherer Zugriff auf Ressourcen, Daten und Dienste umsetzen. Die Isolation ermöglicht auch eine einfachere Entfernung, Aktualisierung oder Änderung einzelner Instanzen.

Skalierbarkeit und Replizierbarkeit
Durch Aggregation und Ressourcenpartitionierung können virtuelle Ressourcen viel kleiner oder größer als die einzelnen physischen Ressourcen ausgelegt werden. Zudem können einzelne virtuelle Ressourcen sehr einfach repliziert und einzelnen virtuellen Instanzen zugeordnet werden.

Sicherung und Testen
Eine wichtige Funktionalität von virtuellen Instanzen ist die Möglichkeit, eine Momentaufnahme eines Zustands zu einem beliebigen Zeitpunkt festzuhalten. Dies bietet die Möglichkeit einer Sicherung sowie die einfache Zurücksetzung einer virtuellen Instanz auf einen festgelegten Zeitpunkt. Dies bietet ideale Voraussetzungen, um Tests durchzuführen.

Geringere Verwaltungskosten
Durch die Abstraktion der physischen Ressourcen in einen Pool von virtuellen Instanzen wird ein Teil der Komplexität maskiert, wodurch allgemeine Verwaltungsaufgaben einfacher oder automatisiert ausgeführt werden können.

3.2.2 Verwandte Arbeiten

Die zuverlässige Bereitstellung von Diensten im Kontext von Smart Grids wird in den Arbeiten [19, 4] untersucht. Neben Technologien wie Software Defined Networking (SDN) und Network Functions Virtualization (NFV) zur Verwaltung des Netzwerks wird Virtualisierung zur Abstraktion der Hardware- und Software-Schichten beschrieben. Die durch die Separierung gewonnene Flexibilität soll es ermöglichen, besser auf fehlerhafte oder störende Ereignisse zu reagieren.

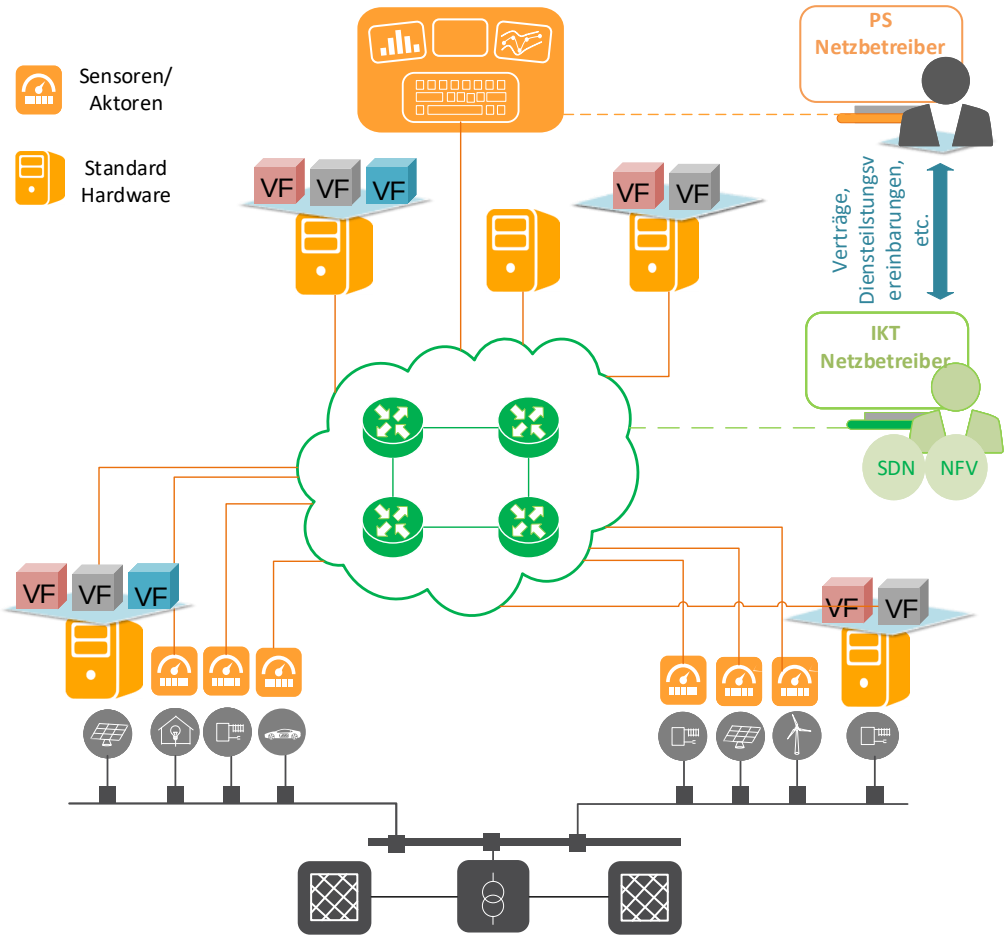

Abb. 3.4.: Grid Function Virtualization (GFV) in Smart Grids [4]

Die Abbildung 3.4 zeigt einen strukturellen Aufbau, wobei im Bereich unterhalb der Aktoren und Sensoren (in Schwarz) die Netzstruktur (z. B. eines Verteilnetzes), im Zentrum (in Grün) ein Kommunikationsnetzwerk und dazwischen (in Orange) Komponenten wie Sensoren, Aktuatoren und Standardhardware abgebildet sind. Die Standardhardware repräsentiert die Bereitstellung klassischer Hardware in der Stationsautomatisierung wie Phasor Measurement Units (pmus) , IEDs oder Server-Infrastrukturen. Auf dieser Hardware wird die Software von netzdienlichen Funktionen durch Virtual Grid Functions (VGFs) repräsentiert. Eine Menge von VGFs bildet die Funktionalitäten eines Dienstes ab, ähnlich zur Definition eines Dienstes in Abschnitt 3.

Das Konzept der GFV ist als Architektur in den Arbeiten [19, 4] beschrieben. An dieser Stelle wird einmal die Beziehung der Virtual Grid Function (VGF) und GFV dargestellt. Die Virtual Grid Function (VGF) beschreibt eine Funktion, die auf eine

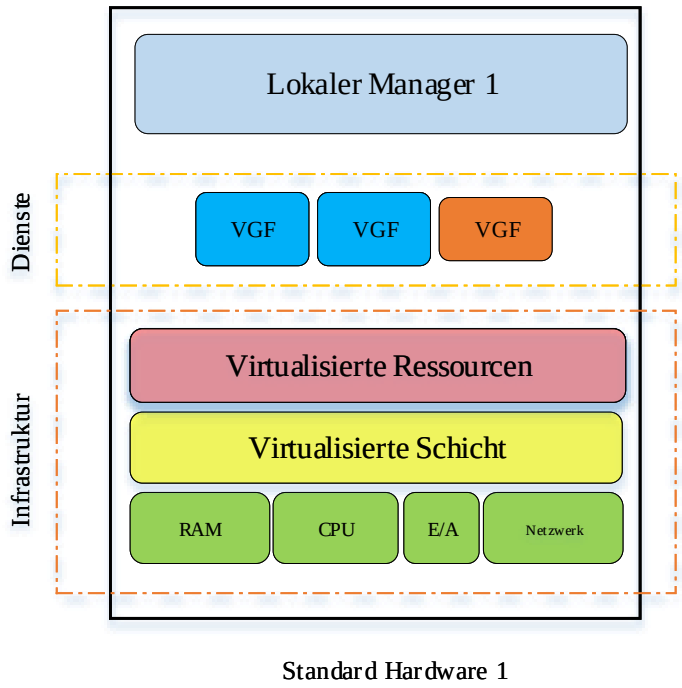

Abb. 3.5.: Ausschnitt der Grid Function Virtualization (GFV)-Architektur [19]

Hardware unter Verwendung von Virtualisierungskonzepten ausführbar ist. Der strukturelle Aufbau in einem System und die Verteilung der Funktionen wird in einem Konzept dargestellt welches als Grid Function Virtualization (GFV) bezeichnet wird. Die Abbildung 3.5 zeigt den für die Virtualisierung von Diensten relevanten Teil der Architektur. Eine Standardhardware ist in die Schichten der Infrastruktur, der darauf laufenden Dienste und der Verwaltungsschicht (dargestellt als „Lokaler Manager“) aufgeteilt. Die Infrastruktur beschreibt die Abstraktion von Ressourcen wie Rechenleistung, Speicher, Netzwerk und Ein- und Ausgabe (E/A)-Schnittstellen zur Bereitstellung der virtualisierten Ressourcen. Zwischen diesen Bereichen liegt die Virtualisierung, die den Transfer zwischen Hardware-Ressourcen und virtualisierten Ressourcen ermöglicht. Die exakten Abläufe sind abhängig von den Hardware-Ressourcen, dem Betriebssystem und der genutzten Technologie und werden im Kapitel 3.5 beschrieben. In der Arbeit werden die Dienste in VGFs aufgeteilt, die durch einen Hypervisor (s. 2.2.2) oder eine Container Laufzeitumgebung (s. 2.2.1) einen isolierten Bereich mit zugewiesenen virtualisierten Ressourcen erhalten. Die Zuweisung und Bereitstellung der virtualisierten Ressourcen ist Teil der Verwaltungsschicht und ermöglicht das Ausführen ganzer Dienste oder einzelner VGF. Basierend auf der verwandten Arbeit werden dem „Local Manager“ Aufgaben zur

Instanziierung, Beendigung und Aktualisierung von VGFs zugewiesen, sowie die Überwachung der Zustandsinformationen.

3.2.3 Virtualisierungskonzept

Die Informationen aus den Eigenschaften der Virtualisierung 3.2.2 und den Anforderungen an Dienste 3.1.2 bilden die Grundlage für das Virtualisierungskonzept. Unter Hinzunahme der Konzepte aus den verwandten Arbeiten beschreibt das in Abbildung 3.6 dargestellte Konzept die Bereitstellung von Diensten auf Standardhardware. Zur Beschreibung des Konzepts werden alle Teilbereiche der drei Schichten (Infrastruktur, Dienste und Verwaltung) im Bezug auf die Designentscheidungen und Anforderungen im Folgenden dargestellt.

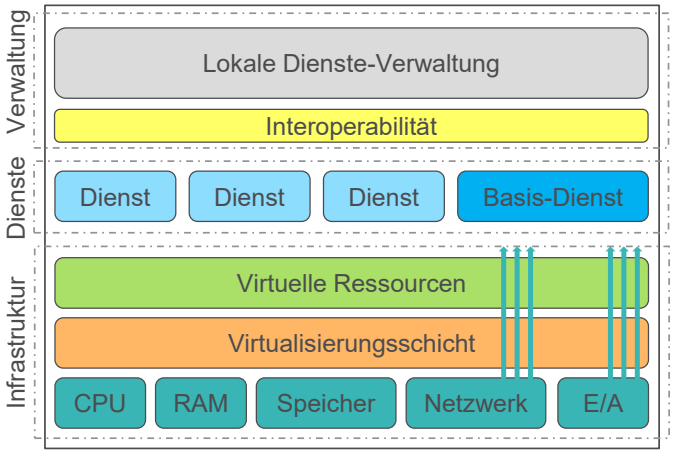

Abb. 3.6.: Virtualisierungskonzept zur Bereitstellung von Diensten auf Standardhardware

Infrastrukturschicht

Die Infrastrukturschicht beinhaltet die Abstraktion von Hardwareressourcen unter Verwendung einer Virtualisierungsschicht. Unter Hardwareressourcen werden ausschließlich Central Processing Unit (CPU) (deutsch: Hauptprozessor [8]), Random Access Memory (RAM) (deutsch: Arbeitsspeicher [9]), Speicher, Netzwerk und E/A-Schnittstellen betrachtet. Unter CPU wird die verfügbare Rechenleistung verstanden, wobei nur die Eigenschaften der Anzahl der Kerne und die Taktgeschwindigkeit pro Kern in GHz untersucht wird. Andere Eigenschaften zur Parallelverarbeitung, Leistung pro Watt, FLOPS (Floating Point Operations Per Second) usw. werden nicht explizit aufgeführt. Beim RAM wird die Kapazität in GB als verwendbare Größe betrachtet. Andere Eigenschaften wie die Taktrate, Speichertyp, CAS (Column Address

[8] In dieser Arbeit wird der englische Fachbegriff Central Processing Unit (CPU) verwendet.
[9] In dieser Arbeit wird der englische Fachbegriff Random Access Memory (RAM) verwendet.

Strobe)-Latenzen usw. werden nicht aufgeführt. Ähnlich verhält es sich beim Speicher, bei dem nur die Kapazität betrachtet wird und andere Eigenschaften zur Taktrate, Datensicherheit, Wiederbeschreibbarkeit usw. nicht weiter aufgeführt sind. Unter Virtualisierung des Netzwerks wird die Abstraktion der Netzwerkschnittstelle verstanden, wobei je nach technologischer Implementierung unterschiedliche Verfahren genutzt werden. Bekannte Verfahren der Technologie Docker sind Bridge-Netzwerke, Host-Netzwerke, Overlay-Netzwerke usw. [33] und bilden eigene Netzwerkschichten zwischen isolierten Diensten oder im Fall von Host-Netzwerken keine Isolierung und damit keine Abstraktion zur Netzwerkschnittstelle der Hardware. Der letztere Fall ist in der Abbildung durch Pfeile zwischen Netzwerk und Diensten dargestellt. Ein ähnlicher Aufbau existiert bei den E/A-Schnittstellen wie den Eingabe- und Ausgabeströmen (stdin, stdout, stderr) oder dem Dateisystem. Durch Pfeile dargestellt kann Diensten auch der direkte Zugriff auf die Hardwareressourcen gewährt werden, in diesem Fall müssen spezielle Befugnisse (engl. Cababilities) [34, 35] des Betriebssystems erlaubt werden. Innerhalb der Infrastrukturschicht beschreibt die Virtualisierungsschicht die Umsetzung zwischen Hardwareressourcen und virtuellen Ressourcen. Die Umsetzung erfolgt, wie in Kapitel 2.2 beschrieben, typischerweise durch Konzepte der Hypervisor-Virtualisierung oder Betriebssystem-Virtualisierung. Die genaue Implementierung ist abhängig von den gewählten Betriebssystemen, der Hypervisorvariante oder der Containertechnologie. Im Unterkapitel 3.3.3 erfolgt eine Analyse und Auswahl der für diese Arbeit verfolgten Technologie. Die virtuellen Ressourcen stehen ebenfalls dieser Technologie zur Verfügung, wie in der Abbildung zu virtuellen Ressourcen dargestellt.

Diensteschicht

Die Diensteschicht beschreibt die isolierte Darstellung von Diensten und deren Trennung untereinander. Ein Dienst ist wie unter der Definition 3 beschrieben zu verstehen. Im Vergleich zur GFV-Architektur, in der Dienste in logisch getrennte VGF aufgeteilt werden, ist in diesem Virtualisierungskonzept die vollständige Sicht eines Dienstes abgebildet. Diese Darstellung wurde aus Gründen der Interoperabilität und Spezifikation gewählt. In Bezug auf die Darstellung ist die Zusammengehörigkeit zwischen einzelnen VGF zu einem Dienst farblich erkennbar, jedoch nicht, inwieweit sich diese Ressourcen aus einem isolierten Bereich der Virtualisierung (z. B. als Container) teilen. Eine Isolierung jeder einzelnen VGF mit eigenen Ressourcen würde höhere Ressourcenkosten durch die Umsetzung vieler isolierter Instanzen verursachen. Zudem wird die Interoperabilität zwischen isolierten Bereichen im Regelfall durch Netzwerkschnittstellen geregelt, sodass jede VGF einzelne Kommunikationsschnittstellen implementieren muss. Mit Spezifikation ist die Beschreibung eines Dienstes im Engineeringprozess gemeint, was sich auf Kapitel 5 bezieht. Hierbei

skaliert die Komplexität der Spezifikation mit den Einzelbeschreibungen pro isoliertem Bereich. Besteht der Bedarf der Trennung eines Dienstes in seine Teilaufgaben, kann das Konzept eines System of System (SoS) (deutsch: System von Systemen [10])[36] betrachtet werden, solange die Merkmale Autonomie und Interoperabilität erhalten bleiben. Autonomie bedeutet, dass jedes einzelne System autonom agiert und klar definierte Ziele verfolgt. Unter Interoperabilität wird der Austausch von Informationen und Daten zwischen einzelnen Systemen unter der Verwendung spezifizierter Schnittstellen verstanden. Der im Virtualisierungskonzept dargestellte Basis-Dienst übernimmt spezifische Aufgaben, die durch exklusiv zugreifbare Ressourcen begrenzt sind. Ein Beispiel dafür sind Zugriffe auf einzelne analoge E/A-Schnittstellen zum Auslesen von Sensoren oder zur Steuerung von Aktoren. Es gibt unterschiedliche Mechanismen zur Synchronisation paralleler Zugriffe wie exklusiver Zugriff, Semaphoren etc. [37]. Damit diese Mechanismen nicht hardwarespezifisch umgesetzt werden müssen, stellt der Basis-Dienst diese über eine standardisierte Kommunikation zwischen den Diensten bereit.

Verwaltungsschicht

Innerhalb der Verwaltungsschicht definiert das Virtualisierungskonzept eine lokale Dienste-Verwaltung zur Planung, Organisation, Koordination und Kontrolle von Ressourcen und Aktivitäten in Bezug auf Dienste innerhalb eines Hardware-Geräts. Das bedeutet, verfügbare virtuelle Ressourcen werden einzelnen Diensten zugewiesen und deren Aktivität überwacht. Die Planung und Organisation von Diensten wird durch lokale Steuerung auf diesem Hardware-Gerät übernommen, wobei die genaue Funktionsweise technologieabhängig ist. In dem Unterkapitel 3.5 ist eine Implementierung dargestellt. Der Austausch zwischen Diensten und Verwaltung wird durch eine Interoperabilitätsschicht realisiert, die im Regelfall über interne Netzwerkschichten implementiert ist. Die Interoperabilität wurde im Vergleich zur GFV-Architektur ergänzt, da die Funktionsweise der Kommunikation zwischen Diensten und Verwaltung stark variieren kann und Einfluss auf die Kommunikation außerhalb des Hardware-Geräts hat. Neben der lokalen Diente Verwaltung erfolgt in Kapitel 4 die Beschreibung der zentralen Orchestrierung und in diesem Zusammenhang der Austausch von Planungsinformationen zwischen lokaler und zentraler Steuerung.

Bezug auf die Anforderungen an Dienste

Die Anforderung an eine hardwareunabhängige Bereitstellung von Diensten lässt sich durch Eigenschaften in der Infrastrukturschicht und Diensteschicht erfüllen. Die in der Infrastrukturschicht beschriebene Abstraktion der Hardwareressourcen ermöglicht die gemeinsame und gleichzeitige Nutzung und Ausführung mehrerer virtueller Ressourcen. Darauf aufbauend stellt die Diensteschicht isolierte virtuelle

[10] In dieser Arbeit wird der englische Fachbegriff System of System (SoS) verwendet.

Instanzen bereit, in denen Dienste implementiert und ausgeführt werden können. In Bezug auf die Beobachtbarkeit ermöglicht die Verwaltungsschicht die Ermittlung von Telemetriedaten, Ressourceninformationen und Zustandsinformationen der ausgeführten Dienste auf dem Hardware-Gerät. Eine Beobachtbarkeit des Gesamtsystems über mehrere Hardware-Geräte wird nicht durch das Virtualisierungskonzept abgedeckt. Die Beobachtbarkeit ist ebenfalls Teil der Anforderung an automatische Updateprozesse und erfüllt die Grundvoraussetzungen für die Überwachung und Überprüfung von Diensten in einem System. Die Identifizierung von neuen Updates muss im Rahmen des Virtualisierungskonzeptes manuell durchgeführt werden. Das im Kapitel 4 beschriebene Verwaltungskonzept ermöglicht weitere automatisierte Prozesse. Die Struktur einzelner isolierter Instanzen ermöglicht den Plug-and-Play-Austausch einzelner Dienste. Dabei erfolgt eine Bereinigung des zu ändernden Dienstes und die Erstellung einer neuen isolierten Instanz, in der die neue Version implementiert und konfiguriert wird. Dieser Vorgang erfolgt über alle Schichten des Virtualisierungskonzeptes. Die Anforderungen der Fernwartbarkeit und Verteilung von Diensten sowie eines kontinuierlichen Engineeringprozesses werden im Kapitel 4 zum Thema zentrales Management behandelt.

3.3 Technologieanalyse

Dieser Abschnitt beschreibt eine Analyse des bisher technologieunabhängigen Konzepts im Bezug auf verfügbarer Technologien und deren Umsetzbarkeit zum Zeitpunkt der Analyse. Die Tabelle 3.1 setzt sechs unterschiedliche Technologien (3.3.2) in Relation zu acht verschiedenen Kriterien (3.3.1). Unabhängig von den Kriterien wurden nur Technologien ausgewählt, die ihren Schwerpunkt auf geringen Ressourcenverbrauch, Effizienz und Flexibilität legen. Diese werden auch Container-Technologien genannt und fallen unter die Kategorie der BS-Virtualisierung, wie im Abschnitt 2.2.1 erläutert. Diese Einschränkung bezieht sich auf die geringen verfügbaren Ressourcen der Hardware-Infrastruktur 3.4, wie sie in der Stationsautomatisierung Anwendung findet.

3.3.1 Vergleichskriterien

Die einzelnen Kriterien und ihre Ausprägungen werden in diesem Abschnitt kurz beschrieben:

Tab. 3.1.: Auswertung der Technologieanalyse

Kriterien	Technologien		
	Docker	**Rocket**	**LXC**
Virtualisierungstyp	Container	Container	Container
Nutzungsbedingungen	Open Source	Open Source	Open Source
BS-Unterstützung	Win, Linux, Mac	Linux	Linux
Veröffentlichungsstatus	Aktiv	Beendet (2020)	Aktiv
Dokumentation	Hervorragend	Gut	Akzeptabel
Community-Unterstützung	Hervorragend	Gut	Gut
Repository	Öffentlich & Privat	Privat	Öffentlich & Privat
Kubernetes-Unterstützung	✓	✓	Drittanbieter

Kriterien	Technologien		
	runC	**Containerd**	**CRI-O**
Virtualisierungstyp	Container	Container	Container
Nutzungsbedingungen	Open Source	Open Source	Open Source
BS-Unterstützung	Linux	Win, Linux	Linux
Veröffentlichungsstatus	Aktiv	Aktiv	Aktiv
Dokumentation	Akzeptabel	Akzeptabel	Gut
Community-Unterstützung	Gut	Gut	Gut
Repository	Privat	Öffentlich & Privat	Öffentlich & Privat
Kubernetes-Unterstützung	Drittanbieter	✓	✓

Virtualisierungstyp

Unter Virtualisierungstyp wird eine Einteilung in die unterschiedlichen allgemeinen Virtualisierungskonzepte 2.2 wie Hypervisor Typ 1, Typ 2 oder BS-Virtualisierung verstanden. Technologien, die Container als virtualisierte Instanz bereitstellen, nutzen die BS-Virtualisierung und zeichnen sich durch eine leichtgewichtige, kontrollierbare und einfach zu verwaltende Umgebung aus.

Nutzungsbedingungen

Die Nutzungsbedingungen beschreiben mögliche Einschränkungen durch lizenzierte oder vollständig kommerzielle Software. Dieses Kriterium ist insbesondere in Anbetracht einer hohen Skalierung in der Verteilnetzebene relevant, weshalb ein hohes Interesse an Open-Source-Lösungen besteht.

BS-Unterstützung

Auch wenn die Verwendung eines allgemeinen Virtualisierungskonzepts die flexible Nutzung des gewünschten BS ermöglicht, gilt dies nicht für die Technologie selbst. Das bedeutet, es gibt Anforderungen an das Betriebssystem der Hardware-Infrastruktur. Die möglichen Ausprägungen wurden auf die drei bekanntesten Betriebssystem (BS) (Windows, Linux und Mac) begrenzt.

Veröffentlichungsstatus

Besonders wichtig sind Informationen zum Erscheinungsdatum von Softwarever-

sionen und zur geplanten Fortführung von Entwicklungen. Unter „Aktiv“ wird die aktive Weiterentwicklung und Bereitstellung aktueller Softwareversionen verstanden, während „Beendet“ den Zeitpunkt der letzten Veröffentlichung und die Einstellung weiterer Softwareentwicklungen darstellt.

Dokumentation
Das Kriterium „Dokumentation“ beschreibt den Umfang und die Verfügbarkeit von Informationen zur Technologie. Die Einschätzung ist in die Kategorien „Hervorragend“, „Gut“ und „Akzeptabel“ unterteilt. Als Grundlage wurden die offiziellen Webseiten, Dokumentationen und GitHub herangezogen.

Community-Unterstützung
Im Vergleich zur Dokumentation bezieht sich die Community-Unterstützung auf zusätzliche Informationsquellen wie Beispiele, Tutorials, Foren, Videos und allgemeine Nutzerzahlen. Die Einschätzung ist in die Kategorien „Hervorragend“, „Gut“ und „Akzeptabel“ unterteilt.

Repository
Unter einem Repository (auch Registry) versteht man einen zentralen Speicherort zur Ablage von Daten, Dateien oder Informationen. Ein Dienst kann als vordefiniertes, eigenständiges Paket mit allen notwendigen Informationen und Abhängigkeiten dargestellt werden. Dieses Paket, als Image bezeichnet, wird in einem Repository gespeichert. Grundsätzlich wird zwischen privaten und öffentlichen Repositories unterschieden. Ein privates Repository ist vorzugsweise im eigenen Netzwerk vorhanden und enthält nur die eigenen Images. Ein öffentliches Repository hingegen beinhaltet viele vordefinierte Images und bietet die Möglichkeit, eigene Images der Allgemeinheit zur Verfügung zu stellen.

Kubernetes-Unterstützung
Dieses Kriterium beschreibt, ob eine Technologie kompatibel mit der Orchestrierungssoftware von Kubernetes ist. Neben der vollständigen Unterstützung (durch Haken gekennzeichnet) gibt es einige Technologien, die nur durch zusätzliche Software eines Drittanbieters die Unterstützung gewährleisten können.

3.3.2 Technologieüberblick

Nachfolgend erfolgt eine kurze Beschreibung der betrachteten Technologien.

Docker
Docker stellt seit März 2013 eine Container-Plattform zur Verfügung. Diese er-

möglicht es Anwendungen und deren Abhängigkeiten in leichtgewichtigen, portablen Containern zu verpacken und zu verteilen. Dabei unterstützt Docker eine Microservice-Architektur, bei der Anwendungen in kleinere, unabhängige Dienste aufgeteilt werden und in eigenen Docker-Containern ausführbar sind. Über Docker Hub stellt Docker ein öffentliches Repository für Docker-Images bereit, wobei jedes Image eine eigenständige, ausführbare Softwarepaketierung ist, die alle notwendigen Ressourcen, Abhängigkeiten und den Anwendungscode enthält.

Rocket
Im Dezember 2014 wurde Rocket erstmals von CoreOS als alternative Container-Runtime mit einem Fokus auf Sicherheit, Isolation und Standardisierung veröffentlicht. Im Gegensatz zu Docker-Images unterstützte Rocket das App Container (appc)-Format, das eine standardisierte Spezifikation für Container-Images und Laufzeitumgebungen bereitstellt. Die kleinste Einheit wurde ähnlich dem Konzept von Kubernetes nicht als Container, sondern als Pod dargestellt. Im April 2017 wurde die aktive Entwicklung eingestellt. Der letzte Entwicklungsstand ist weiterhin verfügbar und kann über GitHub (die offizielle Webseite) abgerufen werden [38].

Linux Containers (LXC)
Linux Containers (LXC) [11] ermöglicht Anwendungen in leichtgewichtigen, isolierten Containern auszuführen und stellt eine Container-Technologie exklusiv für Linux-Systeme bereit. LXC wurde erstmals 2008 veröffentlicht und nutzt explizite Funktionen im Linux-Kernel (z. B. cgroups und namespaces), um Ressourcen zu isolieren und Prozesse zu trennen. Im Vergleich zu Docker liegt der Schwerpunkt von LXC in der Systemvirtualisierung, während Docker auf die Bereitstellung und Isolierung von Anwendungen abzielt. Daher wird LXC oft für die Bereitstellung von Systemcontainern verwendet, bei denen der Fokus auf der Virtualisierung des gesamten Betriebssystems liegt [39].

runC
Die Open Container Initiative (OCI)[40] veröffentlichte die erste standardisierte Container-Laufzeitumgebung im Jahr 2015. runC stellt eine klare, modulare Architektur zur Verfügung, die sich auf Effizienz und Flexibilität konzentriert. Als Teil der Open Container Initiative ermöglicht runC die Interoperabilität mit anderen OCI-konformen Laufzeitumgebungen und Werkzeuge. Dadurch können Container von mehreren unterschiedlichen Laufzeitumgebungen ausgeführt und verarbeitet werden. Zusätzlich ist runC im Regelfall ein integraler Bestandteil der Docker-Plattform. Das bedeutet, dass Docker-Container unter Verwendung von runC-Prozessen ausgeführt werden [41].

[11] In dieser Arbeit wird der englische Fachbegriff Linux Containers (LXC) verwendet.

Containerd
Containerd ist eine von Docker initiierte standardisierte Container-Laufzeitumgebung, die auf dem OCI-Standard basiert. Diese Umgebung wurde erstmals im Dezember 2016 veröffentlicht und später der Cloud Native Computing Foundation [42] übergeben. Containerd besitzt ähnliche Eigenschaften wie die runC-Container-Laufzeitumgebung und kann als alternativer Bestandteil für die Docker-Plattform genutzt werden. Gleichzeitig wird sie auch als Standard-Container-Laufzeitumgebung für Kubernetes eingesetzt und bietet die Basis für die Verwaltung von Containern in Kubernetes-Cluster [43].

CRI-O
CRI-O ist eine Container-Laufzeitumgebung, die speziell für Kubernetes entwickelt wurde und ebenfalls ein Open Container Initiative (OCI)-kompatibles Format unterstützt. Diese Umgebung kann ausschließlich im Zusammenhang mit Kubernetes genutzt werden. Prozesse im Hintergrund werden über Kubernetes aufgerufen und verwaltet. Der Vorteil liegt in der einfachen und effizienten Nutzung von Containern im Kubernetes-System. Dadurch ist keine integrierte Orchestrierungsschicht seitens CRI-O notwendig, im Gegensatz zur Docker-Plattform [44].

3.3.3 Technologieauswahl

Unter Berücksichtigung der Kriterien aus der Tabelle 3.1 und dem Überblick der einzelnen Technologien wurde folgende Auswahl getroffen:

1. Docker als Container-Plattform bildet die Grundlage zur Verwaltung von Diensten und deren Lebenszyklus in einem Container. Ein besonders wichtiger Aspekt ist die Interoperabilität mit Kubernetes, da Kubernetes als zentrale Orchestrierungsplattform eingesetzt wird. In Bezug auf diese Arbeit sind die Eigenschaften der guten Dokumentation und Community-Unterstützung besonders hilfreich, um die technologischen Eigenschaften zu evaluieren und prototypische Implementierungen zu realisieren. Die domänenspezifische Umsetzung in ein Verteilnetzgebiet, wie in Abschnitt 6.2.4 behandelt, legt einen Fokus auf die Eigenschaften der Open-Source-Lösungen, die Unterstützung aller gängigen Betriebssysteme und die aktive Weiterentwicklung. Docker erfüllt all diese Kriterien.

2. Containerd ist eine der benötigten Laufzeitumgebungen (auch Docker Daemon genannt), welche die Aufgabe übernimmt, Docker-Container zu erstellen, auszuführen und zu verteilen. Damit erfüllt Containerd die Aufgaben der

Infrastrukturschicht und Teile der Diensteschicht aus dem in Kapitel 3.2.3 dargestellten Konzept, während Docker die lokale Verwaltung und die Schnittstelle zur zentralen Orchestrierung übernimmt.

3. Rocket wurde aufgrund der fehlenden Weiterentwicklung nicht berücksichtigt.
4. LXC wurde aufgrund der eingeschränkten Kompatibilität mit Kubernetes nicht weiter betrachtet.
5. runC wird als Teilprozess in der Containerd-Architektur abgebildet.
6. CRI-O bietet aufgrund der eingeschränkten BS-Unterstützung nicht genügend Flexibilität für die Evaluation in einem domänenspezifischen Umfeld.

3.3.4 Exploration der Technologie

Dieses Kapitel beschreibt die Funktionsweise der ausgewählten Technologien Docker und Containerd. Die Funktionsweise wird unter Berücksichtigung der Architektur der einzelnen Technologien dargestellt und die Interoperabilität zwischen den Technologien geklärt.

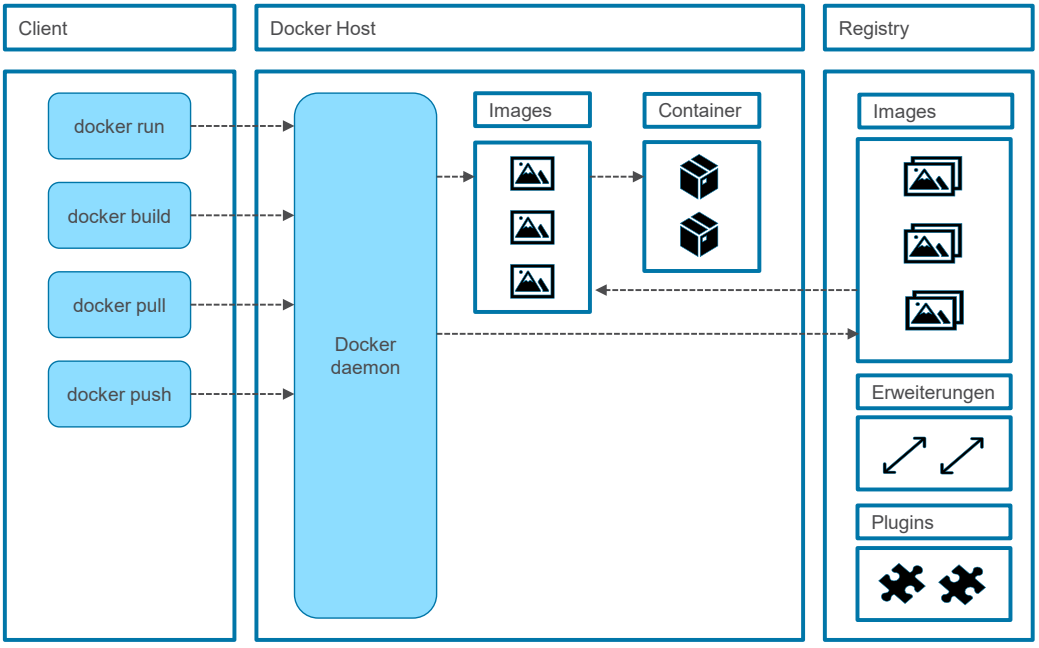

Abb. 3.7.: Docker-Architektur nach [45]

Docker-Architektur

Die Abbildung 3.7 beschreibt die Client-Server-Architektur von Docker und kann

wie folgt interpretiert werden. Docker Client beschreibt die Schnittstelle zwischen dem Nutzer und den von Docker bereitgestellten Befehlen. Die vier dargestellten Befehle ermöglichen das Ausführen von Images als Container (docker run), das Erstellen eines Images (docker build) und das Verschieben von Images zwischen dem lokalen System und dem Repository (docker pull und docker push). Die Client-Server-Architektur erlaubt es, dass die Client-Schnittstelle unabhängig vom Hostsystem läuft. Innerhalb des Hostsystems (Docker Host) werden API-Anfragen des Clients an den Docker-Daemon gesendet. Der Docker-Daemon repräsentiert dabei eine Container-Laufzeitumgebung wie Containerd oder runC, um Anfragen auf die verwalteten Docker-Objekte umzusetzen. Ein Docker-Objekt entspricht den erzeugten und verwalteten Objekten innerhalb eines Hostsystems wie Images, Container, Netzwerke, Speichervolumen und Plugins. Jedes Image repräsentiert eine schreibgeschützte Vorlage mit Instruktionen zum Erzeugen eines Containers. Docker Images ermöglichen es, dass ein Image auf einem anderen Image aufbaut, wobei jede Schicht eine bestimmte Änderung oder Erweiterung gegenüber der vorherigen Schicht darstellt. Dieses Verfahren wird Schichtung genannt und basiert auf dem Unix File System (UFS) [12] [46]. In Abbildung 3.8 ist die Funktionsweise der einzelnen Schichten dargestellt. Aufbauend auf dem „Bootfs“, welches das minimale Dateisystem für den Kernel-Bootvorgang beinhaltet, folgen die Schichten des UFS. Die erste Schicht entspricht einem schreibgeschützten Basis-Image, welches das grundlegende Dateisystem für die darauf aufbauenden Schichten beschreibt. Weitere Schichten sind ebenfalls schreibgeschützt und können Änderungen am Dateisystem beinhalten. Jede Schicht bildet ein neues Image. Die schreibbare Schicht beschreibt die Container-Laufzeitumgebung, d. h. wenn ein Container zur Laufzeit Änderungen vornimmt, werden diese in einen nicht schreibgeschützten Bereich geschrieben. Der Einsatz des UFS ermöglicht eine effiziente Nutzung von Ressourcen, da mehrere Container dasselbe Basis-Image nutzen und sich den schreibgeschützten Bereich teilen können. Änderungen werden nur in den eigenen schreibbaren Bereich durchgeführt. Ebenfalls muss kein aufwändiger Kopiervorgang erfolgen, da ein Container nur Abweichungen vom Image auf die Festplatte schreiben muss.

Befehle zu Docker-Objekten, die der Container-Laufzeitumgebung nicht lokal innerhalb des Hostsystems zur Verfügung stehen, werden über ein Repository bereitgestellt. Neben öffentlich verfügbaren wie Docker Hub können auch private Repository genutzt werden. Diese können weitere Docker-Objekte wie Images, Erweiterungen oder Plugins zur Verfügung stellen, sodass die Laufzeitumgebung diese lokal weiterverarbeiten kann.

[12] In dieser Arbeit wird der englische Fachbegriff Unix File System (UFS) verwendet.

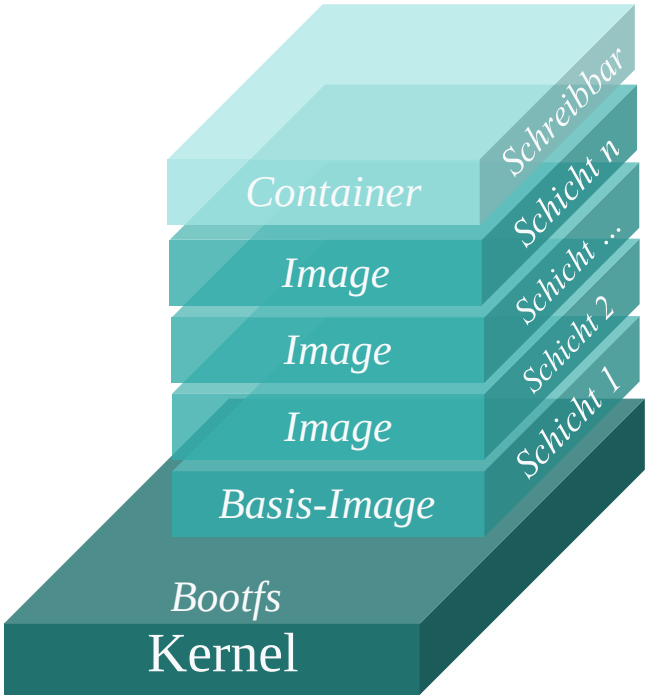

Abb. 3.8.: Aufbau eines Docker Unix File System (UFS)

Containerd-Architektur

In Abbildung 3.9 ist die allgemeine Architektur von Containerd sowie deren Schnittstellen zum darunterliegenden System und dem darüberliegenden Ökosystem dargestellt [43]. Das darunterliegende System bezieht sich auf die unterstützte Systemarchitektur (ARM und Intel) sowie das unterstützte Host-BS (Windows oder Linux). Ausgehend von diesem darunterliegenden System stellt Containerd die Schnittstelle „containerd-shim“ bereit, die zwischen Containerd und den eigentlichen Containerprozessen vermittelt. Diese Containerprozesse können beispielsweise durch runC bereitgestellt werden. Im Backend verwendet Containerd mehrere zentrale Komponenten, um eine effiziente Verwaltung der Container zu gewährleisten. Der „Content Store“ dient dazu, die Images, die die Grundlage für die Container bilden, sicher zu speichern. Ein weiterer wichtiger Bestandteil ist der „Snapshotter“, welcher dafür verantwortlich ist, Momentaufnahmen des aktuellen Zustands eines Containers zu erstellen. Diese Momentaufnahmen ermöglichen es, den Zustand eines Containers zu einem bestimmten Zeitpunkt festzuhalten und bei Bedarf darauf zurückzugreifen, was besonders für Backups und Wiederherstellungen nützlich ist. Zudem umfasst das Backend eine Laufzeitumgebung, die essenzielle Funktionen wie das Starten, Stoppen und Pausieren von Containern verwaltet. Diese Laufzeitumgebung stellt sicher, dass die Container effizient und zuverlässig betrieben werden können, indem sie die notwendigen Ressourcen bereitstellt und überwacht. Im „Core“ werden verschiedene

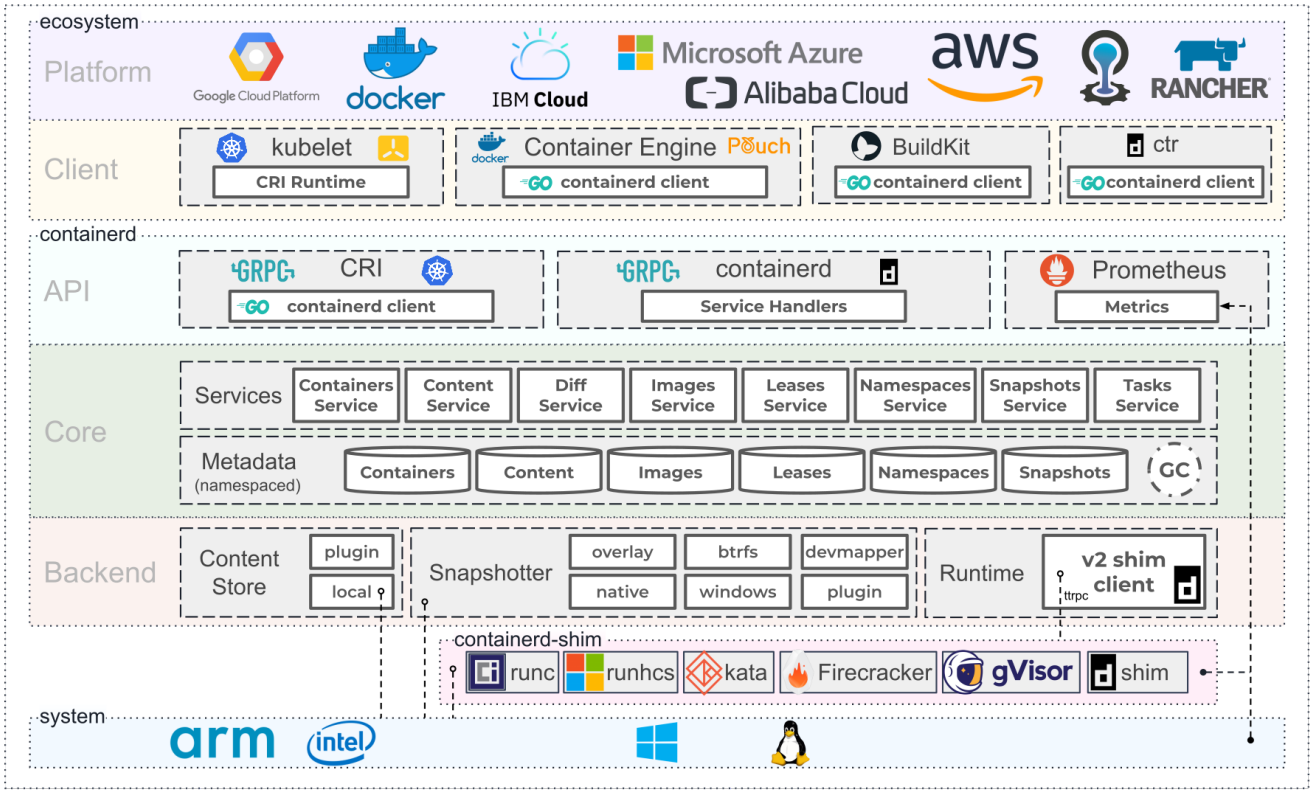

Abb. 3.9.: Containerd-Architektur [43]

Verwaltungsaufgaben und Metadaten als Dienste bereitgestellt. Jeder Dienst übernimmt eine spezifische Rolle zur Verwaltung der Container und implementiert das gRPC-Protokoll, um über „Service Handler“ eine API-Schnittstelle bereitzustellen. Diese API-Schnittstelle wird zur Client-Anbindung darüberliegender Technologien im Ökosystem verwendet. Ein Beispiel für eine solche Schnittstelle ist die Anbindung als Laufzeitumgebung in der Client-Server-Architektur von Docker.

3.4 Anforderungen an die Hardware- und Software-Infrastruktur

Dieses Kapitel beschreibt die notwendigen Anforderungen der Hardware-Infrastruktur zur Umsetzung der Technologien in einem domänenspezifischen Kontext. Die Anforderungen beziehen sich auf die notwendigen Systemressourcen, das unterstützte BS und die konfigurierte Software zur Umsetzung der ausgewählten Technologien. Anforderungen aus dem domänenspezifischen Kontext beziehen sich auf die verfügbaren Hardware-Geräte der Stationsautomatisierung und eventuelle Einschränkungen.

3.4.1 Anforderungen an das Betriebssystem

Die Anforderungen an das Betriebssystem lassen sich in zwei unterschiedliche Bereiche unterteilen. Zunächst stellen die Technologien Anforderungen an die unterstützte Variante des Betriebssystems. Allgemein werden Windows- und Linux-Systeme seitens Containerd und Docker unterstützt. Docker stellt einige Informationen zu den unterstützten Plattformen bereit (z.B. Ubuntu, Debian, CentOS, etc.) und zur unterstützten Architektur (z.B. amd64, arm, etc.).

Zur Überprüfung der Anforderungen an den Linux-Kernel einer Plattform gibt es eine Liste an notwendigen Anforderungen (Tabelle 3.2) und optionalen Anforderungen (Tabelle 3.3). Die optionalen Anforderungen können in speziellen Konfigurationen von Docker und im Verhalten von Containern Anwendung finden. Ein Beispiel ist die Verwendung von Netzwerkprioritäten beim Zugriff von Containern auf das Netzwerk. Im Fall von Windows werden generell Versionen ab Windows 10 64 Bit und einige WSL (Windows Subsystem for Linux) Systeme unterstützt [45].

Tab. 3.2.: Notwendige Anforderungen zur Container-Kompatibilität im Linux-Kernel

Eigenschaft	Status
cgroup hierarchy	Richtig eingebunden [/sys/fs/cgroup]
apparmor	Aktiviert und Werkzeuge installiert
CONFIG_NAMESPACES	Aktiviert
CONFIG_NET_NS	Aktiviert
CONFIG_PID_NS	Aktiviert
CONFIG_IPC_NS	Aktiviert
CONFIG_UTS_NS	Aktiviert
CONFIG_CGROUPS	Aktiviert
CONFIG_CGROUP_CPUACCT	Aktiviert
CONFIG_CGROUP_DEVICE	Aktiviert
CONFIG_CGROUP_FREEZER	Aktiviert
CONFIG_CGROUP_SCHED	Aktiviert
CONFIG_CPUSETS	Aktiviert
CONFIG_MEMCG	Aktiviert
CONFIG_KEYS	Aktiviert
CONFIG_VETH	Aktiviert (als Modul)
CONFIG_BRIDGE	Aktiviert (als Modul)
CONFIG_BRIDGE_NETFILTER	Aktiviert (als Modul)
CONFIG_IP_NF_FILTER	Aktiviert (als Modul)
CONFIG_IP_NF_MANGLE	Aktiviert (als Modul)
CONFIG_IP_NF_TARGET_MASQUERADE	Aktiviert (als Modul)
CONFIG_NETFILTER_XT_MATCH_ADDRTYPE	Aktiviert (als Modul)
CONFIG_NETFILTER_XT_MATCH_CONNTRACK	Aktiviert (als Modul)
CONFIG_NETFILTER_XT_MATCH_IPVS	Aktiviert (als Modul)
CONFIG_NETFILTER_XT_MARK	Aktiviert (als Modul)
CONFIG_IP_NF_NAT	Aktiviert (als Modul)
CONFIG_NF_NAT	Aktiviert (als Modul)
CONFIG_POSIX_MQUEUE	Aktiviert
CONFIG_CGROUP_BPF	Aktiviert

3.4.2 Anforderungen an die Systemressourcen

Informationen zu den benötigten Systemressourcen wie CPU und RAM sind in den Dokumentationen der Technologien oft unzureichend beschrieben. Diese Angaben liefern Richtwerte und berücksichtigen meist nicht die Details der verbauten Hardware und deren exakte Leistungsfähigkeit. Beispielsweise wird bei der Angabe von benötigten 2 GB RAM nicht spezifiziert, welche Taktfrequenz oder Generation des RAM erforderlich ist. Daher kann eine genaue Einschätzung der benötigten Systemressourcen nur durch eine Evaluation auf der Ziel-Hardware erfolgen. Im Kapitel 6 wird die Leistung einzelner Dienste in einem Stationsgerät genauer untersucht.

Tab. 3.3.: Optionale Anforderungen zur Container-Kompatibilität im Linux-Kernel

Eigenschaft	Status
CONFIG_USER_NS	Aktiviert
CONFIG_SECCOMP	Aktiviert
CONFIG_SECCOMP_FILTER	Aktiviert
CONFIG_CGROUP_PIDS	Aktiviert
CONFIG_MEMCG_SWAP	Aktiviert
CONFIG_MEMCG_SWAP_ENABLED	Fehlend
CONFIG_BLK_CGROUP	Aktiviert
CONFIG_BLK_DEV_THROTTLING	Aktiviert
CONFIG_CGROUP_PERF	Aktiviert
CONFIG_CGROUP_HUGETLB	Aktiviert
CONFIG_NET_CLS_CGROUP	Aktiviert (als Modul)
CONFIG_CGROUP_NET_PRIO	Aktiviert
CONFIG_CFS_BANDWIDTH	Aktiviert
CONFIG_FAIR_GROUP_SCHED	Aktiviert
CONFIG_IP_NF_TARGET_REDIRECT	Aktiviert (als Modul)
CONFIG_IP_VS	Aktiviert (als Modul)
CONFIG_IP_VS_NFCT	Aktiviert
CONFIG_IP_VS_PROTO_TCP	Aktiviert
CONFIG_IP_VS_PROTO_UDP	Aktiviert
CONFIG_IP_VS_RR	Aktiviert (als Modul)
CONFIG_SECURITY_SELINUX	Aktiviert
CONFIG_SECURITY_APPARMOR	Aktiviert
CONFIG_EXT4_FS	Aktiviert
CONFIG_EXT4_FS_POSIX_ACL	Aktiviert
CONFIG_EXT4_FS_SECURITY	Aktiviert

3.5 Umsetzung virtualisierter Dienste

In diesem Kapitel wird die Umsetzung virtualisierter Dienste unter Berücksichtigung des beschriebenen Virtualisierungskonzepts, der dargestellten Technologien und einer ausgewählten Hardware-Infrastruktur erläutert. Zudem wird der Prozess zur Erstellung von Images und Containern als Vorbereitung für die Evaluation in Kapitel 6 beschrieben.

3.5.1 Hardware-Infrastruktur

Im Rahmen dieser Arbeit werden drei unterschiedliche Hardware-Infrastrukturen verwendet, die im Hinblick auf die Evaluation in zwei unterschiedlichen Stufen Anwendung finden. Als erste Hardware-Infrastruktur wurden Raspberry Pis verwendet. Diese eignen sich aufgrund ihrer Vielseitigkeit, Erschwinglichkeit und der großen Unterstützung durch die Community besonders für prototypische Implementierungen im wissenschaftlichen Kontext. In den Forschungsprojekte i-Automate [12] und i-Autonomous [13] erfolgten Felderprobungen, in denen zwei industrielle

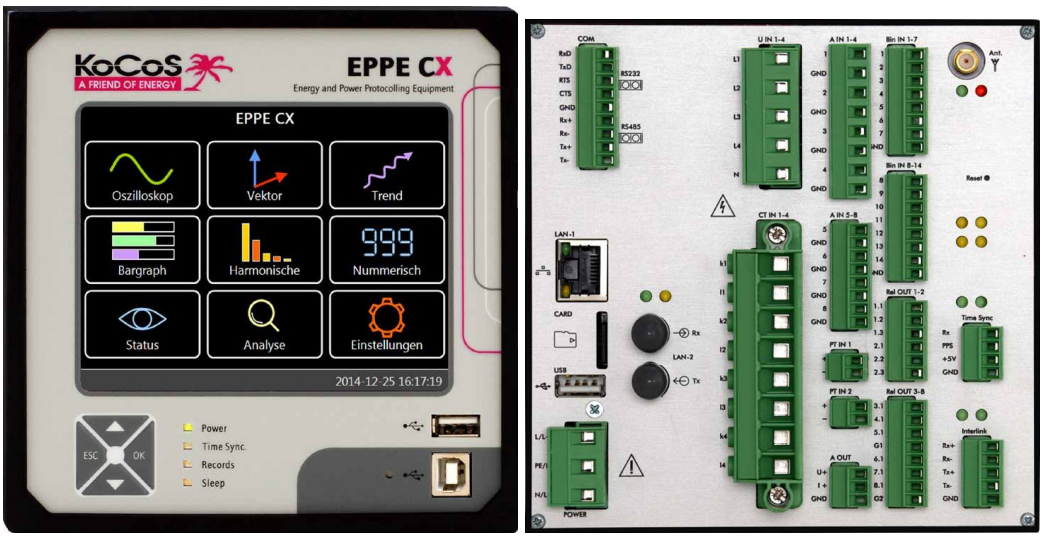

Abb. 3.10.: Stationäres Überwachungssystem EPPE CX [47]

Hardwaregeräte der Firma KoCoS [47] eingesetzt wurden. Das EPPE CX 3.10 dient als stationäres Überwachungssystem für den Schalttafeleinbau, während das Sherlog CRX 3.11 als konfigurierbarer Störschreiber und Energiequalitätsmonitor verwendet wird. Die industriellen Hardwaregeräte stellen im Vergleich zum Raspberry Pi die notwendigen Schnittstellen zur Umwandlung von Kleinsignalen für die verbaute Sensorik in den Stationen bereit. Außerdem sind industrielle Komponenten für den Dauerbetrieb geeignet und erfüllen eine Reihe von notwendigen Zertifizierungen in der Stationsautomatisierung. Die Spezifikationen der drei Geräte in Bezug auf CPU, RAM, Speicher und Netzwerk sind in Tabelle 3.4 aufgeführt. Der direkte Leistungsvergleich zeigt eine ähnliche Prozessorarchitektur und Schnittstellenspezifikation für das Netzwerk. Die Speicherkapazität und der RAM unterscheiden sich nur minimal, während die CPU-Leistung beim Raspberry Pi deutlich erhöht ist.

Tab. 3.4.: Spezifikation der Hardware-Module

Typ	RPi 3b+	EPPE CX + Sherlog CX
CPU	4 x 1,4 GHz Broadcom BCM2837B0	2 x 1.4 GHz ARM Cortex-A53
Echtzeit-Prozessor	-	ARM Cortex-M4, 200 MHz
RAM	1 GB DDR3 L-1066	1 GB LPDDR2-SDRAM
Speicher	16 GB microSD	16 GB eMMC
Netzwerkschnittstelle	1 x 10/100/1000 Mbps	1-2 x 10/100/1000 Mbps
Prozessorarchitektur	Arm (32-Bit)	Arm (32-Bit)

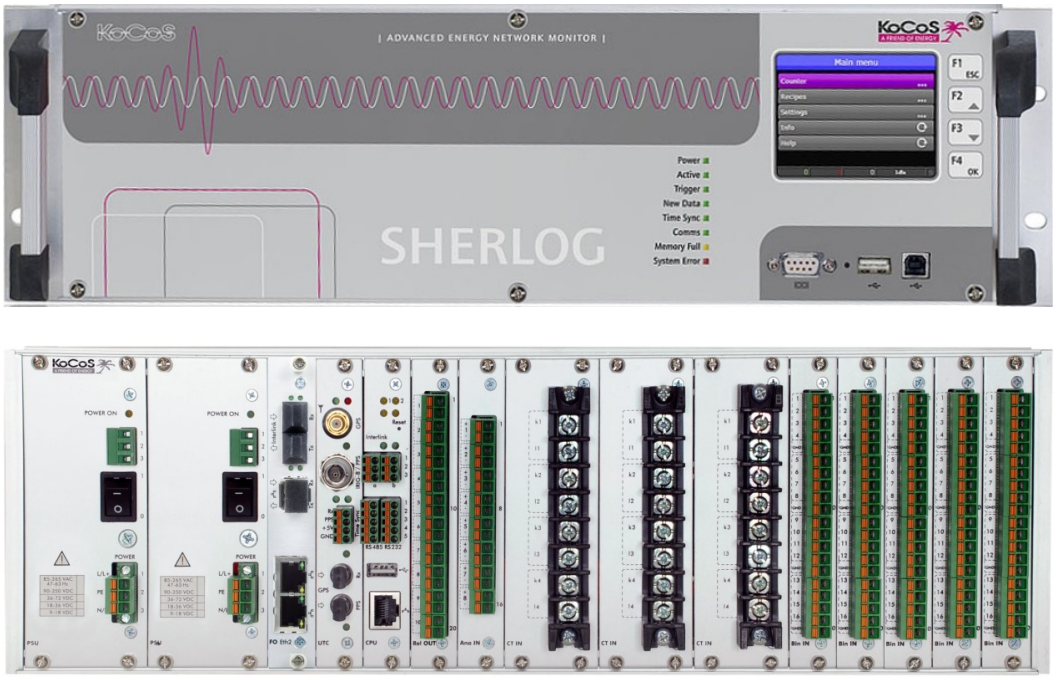

Abb. 3.11.: Konfigurierbarer Störschreiber und Energiequalitätsmonitor [47]

3.5.2 Implementierung der Technologien

Unter Berücksichtigung der Anforderungen an das Betriebssystem wurden für die Raspberry Pis (RPIs) eine bestehende Linux-Variante „Ubuntu 18.04.5 LTS“ verwendet. Die notwendigen und optionalen Anforderungen zur Container-Kompatibilität aus 3.4.1 werden vollständig erfüllt. Die Berechnungsmodule der Industriekomponenten unterstützen in ihrer Standardauslieferung dieses Betriebssystem nicht, sondern eine Debian-Version ohne die Unterstützung der notwendigen Anforderungen zur Container-Kompatibilität. Als Konsequenz wurde aufbauend auf einer „Debian Buster 10“-Version ein neues Betriebssystem konfiguriert, notwendige Module implementiert und für die Prozessorarchitektur der Hardware kompiliert.

Die Abbildung 3.12 zeigt das die ausgewählten Technologien aus 3.3 in das Virtualisierungskonzept überführbar ist. Die Systemressourcen werden über das Host-BS bereitgestellt. RunC und containerd-shim bilden die Schnittstelle zu diesen Systemressourcen und stellen diese der darüberliegenden containerd-Architektur zur Verfügung. Diese übernimmt die Verwaltung und Virtualisierung der Systemressourcen. Über eine API-Schnittstelle zur Docker-Plattform erfolgt der Austausch zwischen Systemressourcen, Diensten und der Verwaltung durch Docker. In dem Virtualisierungskonzept ist diese Schnittstelle als Interoperabilität dargestellt.

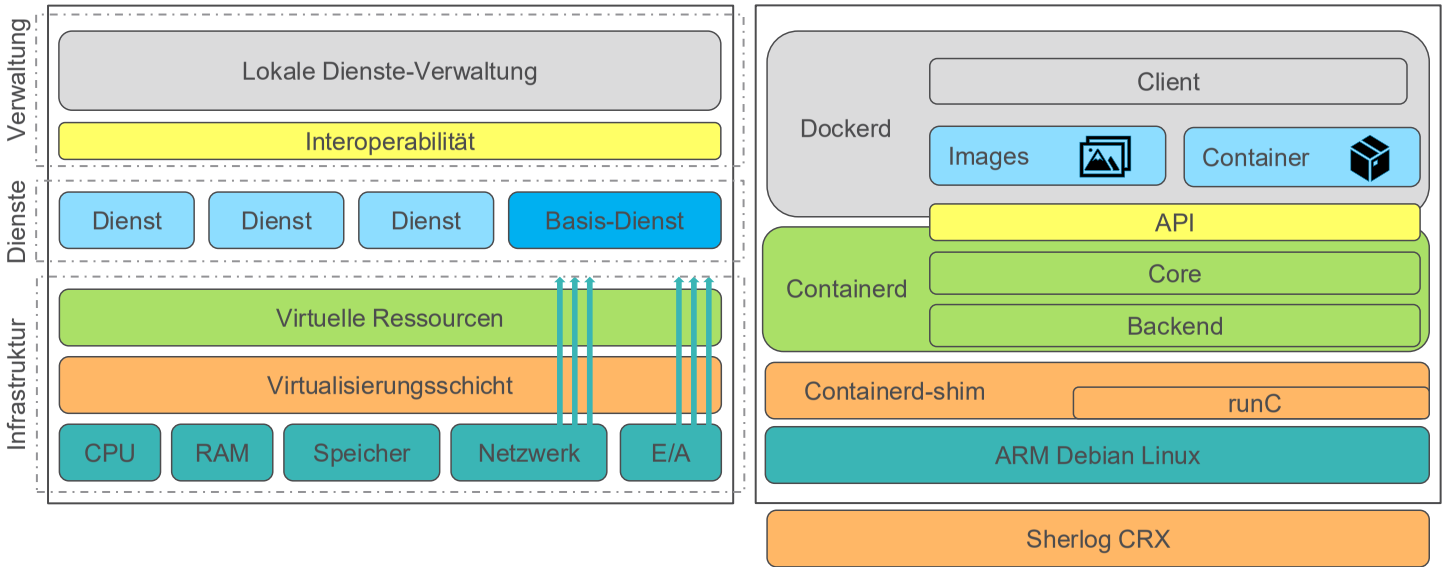

Abb. 3.12.: Technologische Umsetzung

Die Installation des Betriebssystems auf dem Raspberry Pi ist sehr einfach und erfolgt durch die Übertragung eines Installationsabbilds (ISO) auf die SD-Karte. Bei den Industriekomponenten musste zunächst ein Live-Betriebssystem von einem USB-Stick geladen werden, um im Anschluss per RS232-Schnittstelle die Übertragung eines Installationsabbilds in den eMMC-Speicher vorzunehmen. Aufbauend auf dem Betriebssystem konnte per Paketmanager die Installation der einzelnen Technologien erfolgen. Nach vollständiger Konfiguration und Überprüfung der Funktionsfähigkeit aller Technologien wurde ein Installationsabbild erstellt. Da es sich im Rahmen der Feldtestüberprüfung um ein geschlossenes Netzwerk mit starken Restriktionen und keinem Zugriff auf das „World Wide Web“ handelt, wäre eine nachträgliche Installation im verbauten Zustand deutlich aufwändiger und langsamer.

3.5.3 Erstellung von Images und Containern

Der Aufbau eines Images entspricht der Struktur, die in Abbildung 3.8 dargestellt ist. Die Erstellung eines Images kann auf zwei Arten erfolgen: durch die Verwendung eines Dockerfiles oder durch das Extrahieren eines Images aus einem exportierten Dateisystem (z.B. einem Container). Ein Dockerfile, wie im Codeausschnitt 3.1 dargestellt, ermöglicht die Spezifikation des Images unter Verwendung von zehn unterschiedlichen Anweisungen. Eine kurze Beschreibung der Anweisungen ist wie folgt:

- **FROM:** Beschreibung des Basis-Images und der Version.
- **WORKDIR:** Festlegung des Arbeitsverzeichnisses für die folgenden Anweisungen.
- **COPY:** Kopieren von Dateien und Verzeichnissen vom Host in das Image.
- **ENV:** Setzen der Umgebungsvariablen für den Container.
- **RUN:** Ausführen von Befehlen während des Build-Prozesses.
- **EXPOSE:** Angabe des Ports, auf dem der Container lauschen soll.
- **CMD:** Angabe des auszuführenden Befehls beim Start des Containers (einmalig).
- **LABEL:** Hinzufügen von Metadaten zu einem Image.
- **USER:** Festlegen des zu verwendenden Benutzers bzw. der Benutzer-ID.

- **VOLUME:** Erstellen eines benannten Volumes und Verbinden des Containers mit diesem.

```
# Verwendet das offizielle Alpine-Image als Basis
FROM node:14-alpine

# Setzt das Arbeitsverzeichnis im Container
WORKDIR /app

# Kopiert den lokalen Code ins Arbeitsverzeichnis
COPY . .

# Setzt eine Umgebungsvariable
ENV NODE_ENV=production

# Führt Befehle aus, um Abhängigkeiten zu installieren
RUN npm install

# Öffnet Port 3000 im Container
EXPOSE 3000

# Fügt Metadaten hinzu
LABEL version="1.0" description="Beispiel Dockerfile"

# Befehl beim Start des Containers ausführen
CMD ["npm", "start"]
```

Skript 3.1: Ein einfaches Dockerfile Beispiel

Die Speicherung eines bestehenden Containers als Image kann genutzt werden, um einen Container zur Laufzeit zu testen und Instanzen als Image zu exportieren. Das Debuggen zur Laufzeit innerhalb einer interaktiven Konsole ist einfacher und der Export ermöglicht die Sicherung und Replizierung eines speziellen Szenarios. Eine genaue Beschreibung des Ablaufs kann in der Dokumentation [48] von Docker gefunden werden.

Steht ein Image zur Verfügung, gibt es mehrere Möglichkeiten, einen Container zu starten. Die einfachste Möglichkeit ist das Ausführen eines „docker run“-Befehls in der Konsole:

```
docker run -d --name mein-container mein-container:
    latest
```

Skript 3.2: Ausführen eines Containers

Eine weitere Methode ist die Verwendung von Docker Compose, das es ermöglicht, Container-Anwendungen zu definieren und zu verwalten. Dazu werden die Konfigurationen der Container, Netzwerke und Volumen in einer YAML-Datei spezifiziert. Der Codeausschnitt 3.3 zeigt ein Beispiel, in dem eine Webanwendung und eine Datenbank gestartet werden. Eine andere Möglichkeit ist die Verwendung von Kubernetes zur Verwaltung und Ausführung von Containern. Diese Möglichkeit wird im Kapitel 4 weiter untersucht.

```
# Docker-Compose-Version
version: '3'

# Dienste, die in dieser Datei definiert sind
services:
  # Der Web-Dienst, der das nginx-Image verwendet
  web:
    # Verwendetes Docker-Image
    image: nginx:latest
    ports:
      # Mapping: Host-Port 80 auf Container-Port 80
      - "80:80"

  # Der Datenbank-Dienst, der das postgres-Image
     verwendet
  db:
    # Verwendetes Docker-Image
    image: postgres:latest
    environment:
      # Umgebungsvariable: Datenbankname
      POSTGRES_DB: mydatabase
```

Skript 3.3: Eine einfache Docker-Compose-Datei

3.6 Zusammenfassung

Dieses Kapitel beschreibt die Analyse von Diensten und deren Anforderungen, wie Hardwareunabhängigkeit, Beobachtbarkeit, automatische Updates, verteilte Dienste und Fernwartbarkeit, im Kontext von CPES und Stationsgeräten. Aufbauend auf diesen Informationen wurden Konzepte zur Virtualisierung von Diensten, ihre Eigenschaften und bestehende Ansätze in die Erstellung des ersten Artefakts, dem Virtualisierungskonzept, integriert. Insgesamt besteht das Konzept aus drei Schichten. Die Infrastrukturschicht abstrahiert physische Ressourcen und stellt diese in isolierten Instanzen zur Verfügung. Diese Instanzen werden in der Diensteschicht zur Bereitstellung und Ausführung von Diensten genutzt. Abgeschlossen wird das Konzept durch eine Verwaltungsschicht, in der Dienste und Systemressourcen verwaltet werden. Im Rahmen der Konzeptionierung wurden unterschiedliche Technologien analysiert und bewertet. Die Umsetzung erfolgte unter anderem auf zertifizierten Stationsgeräten der Firma KoCoS, deren Ausstattung an die Anforderungen der notwendigen Hardware und Betriebssysteme angepasst wurde. Abschließend wurde in diesem Kapitel die Erstellung von Container-Instanzen und die Integration von Diensten in diese Instanzen beschrieben.

4 Verwaltung von Diensten

Die mit einem Smart Grid verbundenen Digitalisierungsprozesse führen zu neuen Herausforderungen in der Stationsautomatisierung. Der Bedarf an vollständiger Beobachtbarkeit und Steuerbarkeit in der Verteilnetzebene beeinflusst die Komplexität der IT-Infrastruktur erheblich. Wartbarkeit, Skalierbarkeit, Fehlertoleranz und Performanz sind Beispiele für Anforderungen an eine zukünftige Infrastruktur im Smart Grid.

Die Bereitstellung und Verwaltung von virtualisierten Diensten in Stationen kann durch den Ansatz Infrastructure as Code (IaC) (deutsch: codebasierte Beschreibung) den Anforderungen gerecht werden. Im ersten Schritt erfolgt eine Analyse verwandter Arbeiten zum Thema Orchestrierung von Diensten. Anschließend wird das Virtualisierungskonzept aus Kapitel 3.2.3 um ein Verwaltungskonzept erweitert. Durch eine Anforderungs- und Technologieanalyse wird die Umsetzung einer Technologieinfrastruktur zur Verwaltung von Diensten in Stationen beschrieben, die als Vorbereitung für die Evaluation im Kapitel 6 dient. Abschließend erfolgt eine Zusammenfassung unter Berücksichtigung der Forschungsfragen.

4.1 Verwandte Arbeiten

Die Arbeiten [19, 4] zur GFV beschreiben neben einem Konzept zur Virtualisierung von Diensten auch die Orchestrierung und Verwaltung von Diensten. Eine vollständige Darstellung der Architektur ist in Abbildung 4.1 zu finden. Ergänzend zur virtualisierten Bereitstellung von Diensten auf Standardhardware und der lokalen Verwaltung von Diensten stellt die Management-Schicht die Verwaltung und Orchestrierung über mehrere Geräte hinweg bereit. Der Orchestrator wird als ein Aspekt mit globaler Sicht auf die Dienste und deren zugehörigen VGFs beschrieben. Diese globale Sicht erfolgt durch die Überwachung von Systemressourcen mittels verschiedener Überwachungssysteme. Auf Basis der ermittelten Informationen können Abweichungen und Anomalien im Vergleich zum gewünschten Zustand erkannt werden. Eine weitere Aufgabe des Orchestrators ist die Verwaltung von Diensten.

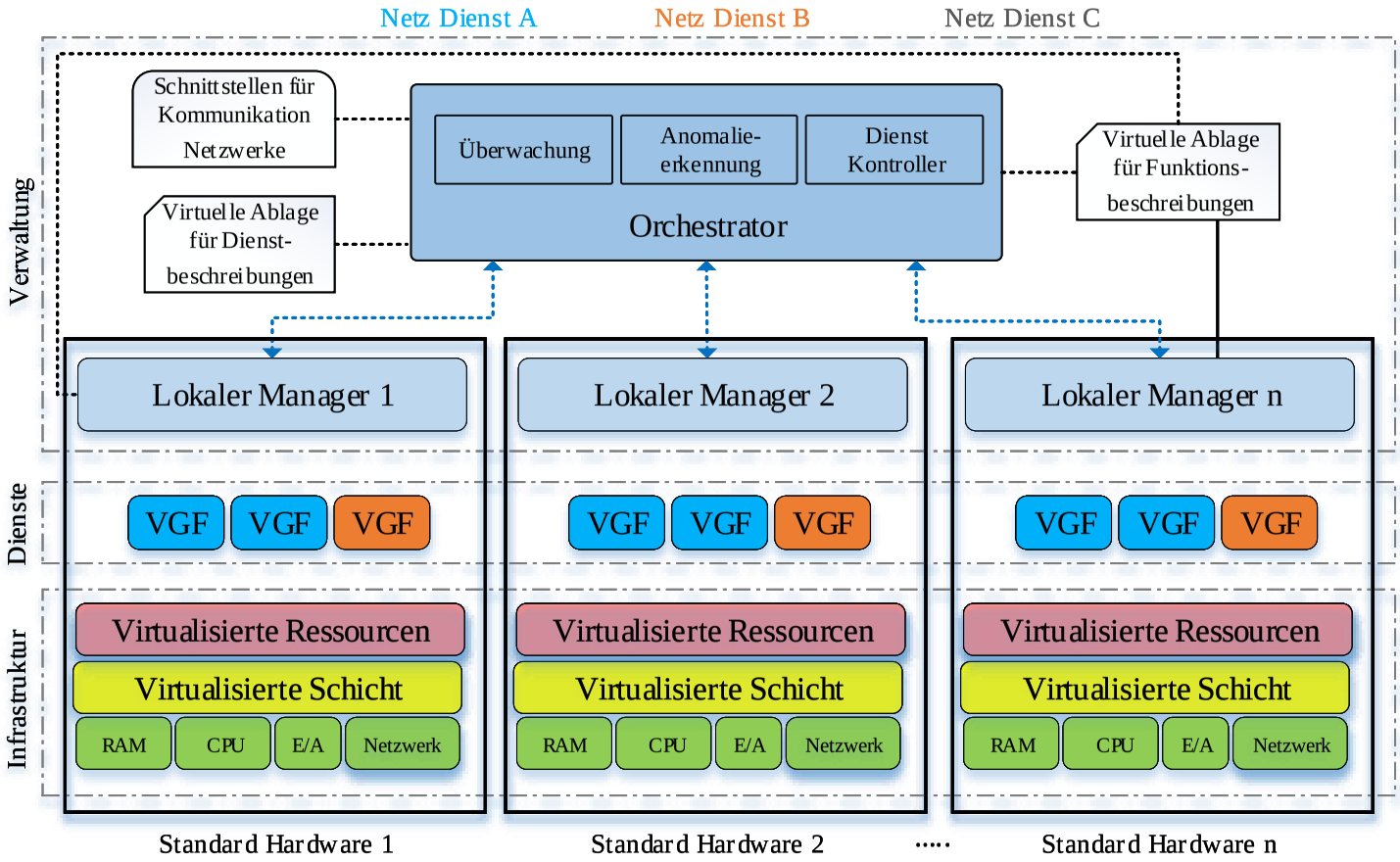

Abb. 4.1.: Grid Function Virtualization (GFV)-Architektur [19]

Beispiele hierfür sind das Starten, Stoppen, Verschieben oder Anpassen von Diensten. Der Orchestrator steht in direktem Austausch mit den lokalen Managern der einzelnen Geräte, um die Befehle umzusetzen. Unterstützt wird der Orchestrator durch den Zugriff auf die Beschreibungen von Diensten und VGFs. Insbesondere werden Informationen über den Dienst (z. B. Programmcode, Image etc.) und deren Quality of Service (QoS)-Anforderungen zur Verfügung gestellt.

Im Kontext der Arbeit [8] wurde die Umsetzbarkeit der beschriebenen GFV-Architektur gezeigt. Die einzelnen konzeptionellen Schichten wurden jeweils durch unterschiedliche Komponenten und Technologien umgesetzt. RPIs als Internet of Things (IoT) [1]-Geräte wurden zur Abbildung der Standardhardware verwendet. Eine Darstellung der Plattform ist in Abbildung 4.2 zu finden. Auf jedem Raspberry Pi (RPI) wurde eine Docker-Instanz implementiert, die zum einen die Aufgabe der Virtualisierung von Ressourcen übernimmt und zum anderen das lokale Management. Als Dienste wurden virtuelle RTUs, Datenerfassungsdienste, ein CVC-Dienst und ein SE als Docker-Container verwendet. Die virtuellen RTUs bilden die Schnittstelle zwischen der Energiesimulation und anderen Diensten. Informationen aus der Energiesimulation werden über UDP an die virtuellen RTUs auf Stationsebene gesendet. Anschließend erfolgt eine Umwandlung in das IEC 61870-5-104-Protokoll, welches als klassisches Austauschformat zwischen Stationen und Leitsystem gilt. Der Datenerfassungsdienst dient als Gegenstelle zur Aufnahme der 104er-Nachrichten und stellt diese den Diensten CVC und SE bereit. Die Erfassung der Energie- und Kommunikationssimulationen erfolgte über zwei echtzeitfähige Simulatoren (OPAL-RT und EXata). Das zentrale Management wurde in der Arbeit auf einem Linux-Server implementiert. Es besteht aus einem IKT-Überwachungssystem (Check-MK), einer anwendungsfallorientierten Anomalieerkennung und einer zentralen Dienstesteuerung über Docker Swarm. Das IKT-Überwachungssystem übernimmt die Erfassung der Statusinformationen wie Leistungs- und Speicherverbrauch der RPIs und der ausgeführten Dienste. Diese Informationen dienen der Anomalieerkennung als Berechnungsgrundlage zur Erfassung einer Fehlfunktion. Basierend auf den Dienste- und Funktionsbeschreibungen wird eine Entscheidung getroffen und diese durch das zentrale Management durchgeführt.

Als Anwendungsfall wird in der beschriebenen Veröffentlichung der Ausfall des CVC-Dienstes dargestellt, und untersucht, welcher Einfluss auf das CPES mit oder ohne die Verwendung einer GFV-Architektur existiert. Als Grundlage wurde ein CIGRE-Benchmark-Netz aus dem Mittelspannungsnetz mit insgesamt zwölf Sammelschienen genutzt. Der Datenaustausch zwischen Diensten wurde zusätzlich durch

[1] In dieser Arbeit wird der englische Fachbegriff Internet of Things (IoT) verwendet.

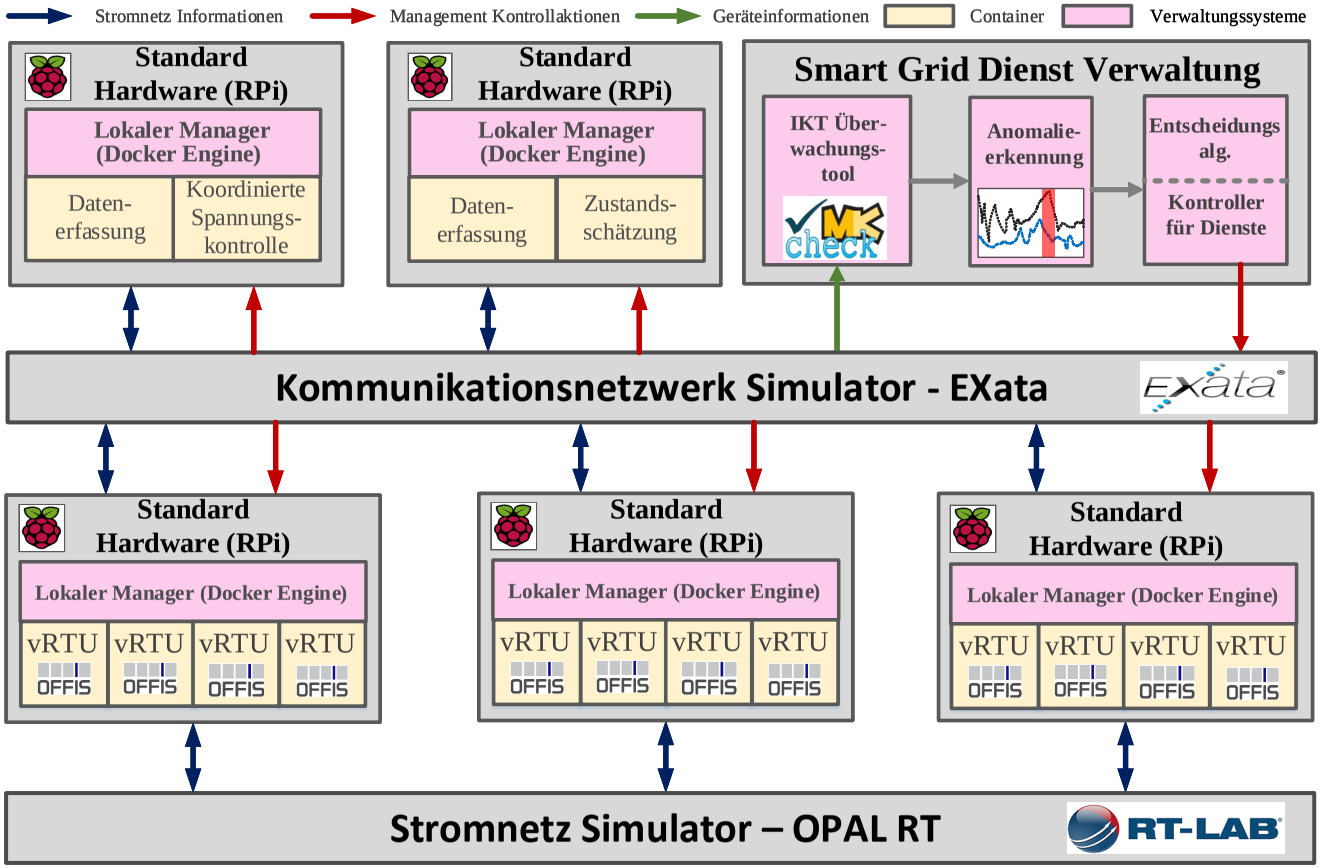

Abb. 4.2.: Grid Function Virtualization (GFV)-Plattform [8]

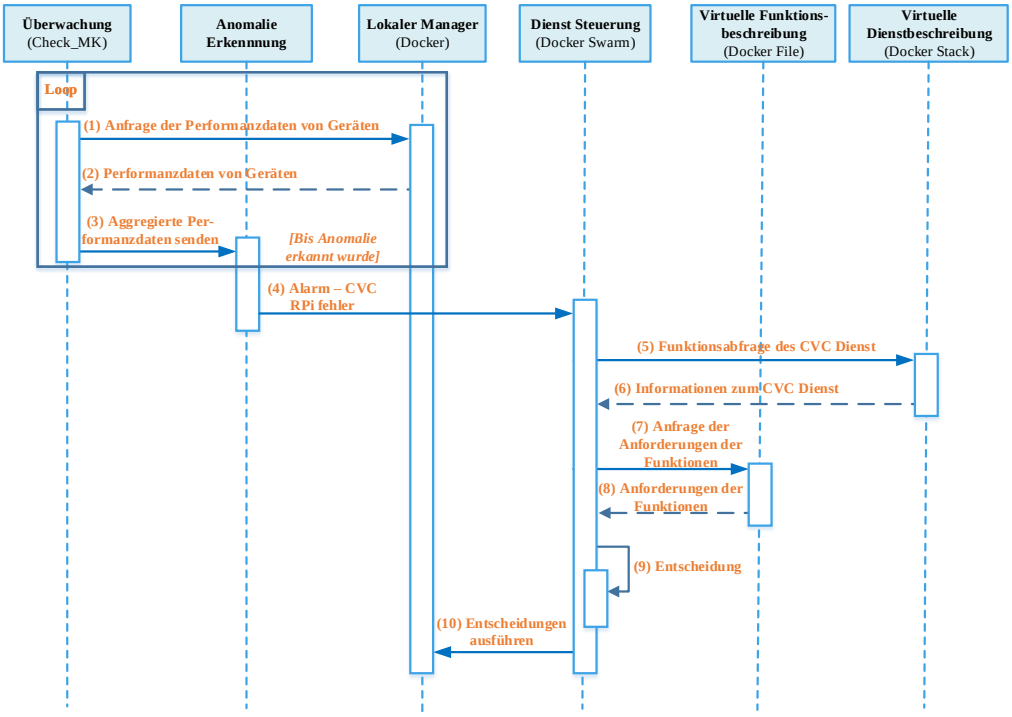

Abb. 4.3.: Sequenzdiagramm der umgesetzten Fallstudie [8]

eine Kommunikationsinfrastruktur simuliert. Eine detaillierte Darstellung ist in Abbildung 4.4 zu finden. Im Rahmen der Untersuchung wurde das Netz um die grün markierten steuerbaren Elemente und einen Kondensatorspeicher zur stufenweisen Bereitstellung einer Nominalspannung angepasst. Ein kompletter Prozessablauf einer Fallstudie ist im Sequenzdiagramm 4.3 dargestellt.

Bezogen auf den Anwendungsfall wurden kontinuierlich Messdaten aus der Energiesimulation über die virtuellen RTUs konvertiert und als 104er-Nachrichten an den SE- und CVC-Dienst weitergeleitet. Weiterhin wurde kontinuierlich der Zustand aller RPIs und der ausgeführten Dienste überwacht. Anschließend wurden zwei Szenarien des Anwendungsfalls durchgespielt, deren Ergebnisse in Abbildung 4.5 dargestellt sind. Im ersten Szenario ohne die Verwendung einer GFV und im zweiten Szenario unter Verwendung der GFV. Die Abbildung zeigt für jedes Szenario die Spannung jeder Sammelschiene, die Blindleistung gemessen am Transformator und die erhaltenen Messwerte am CVC-Dienst. Zusätzlich wurden unterschiedliche Ereignisse nummeriert (1-10) in der Abbildung markiert. Begrenzungen der Spannungsbänder und der maximalen gewünschten Blindleistung sind durch rote Bereiche gekennzeichnet.

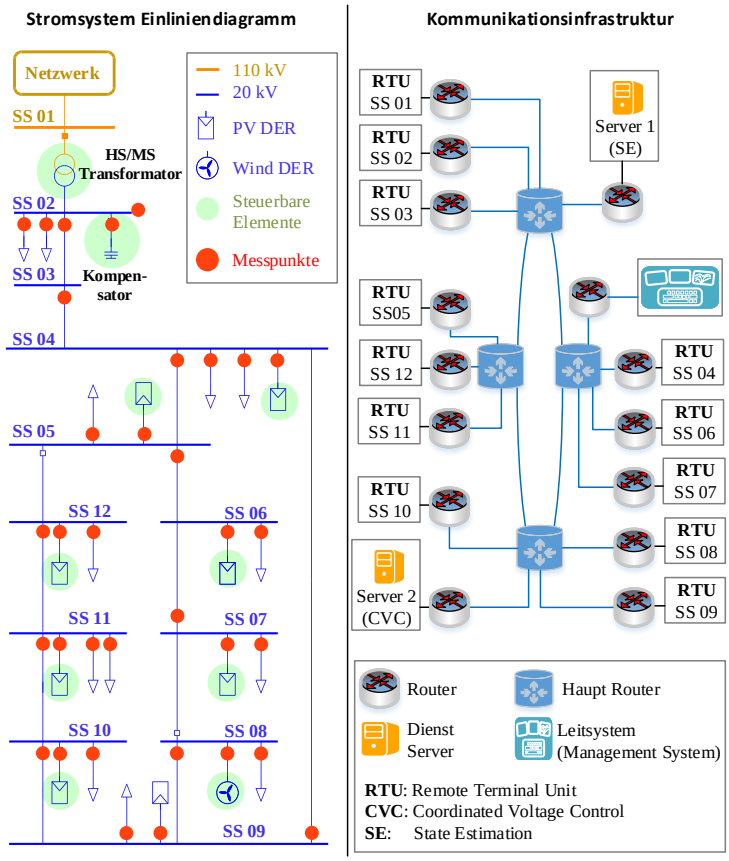

Abb. 4.4.: Angepasstes CIGRE-Benchmark-Netz und Kommunikationsinfrastruktur [8]

Im ersten Ereignis ist der Kondensatorspeicher ausgefallen, wodurch die Spannung kurzzeitig abgefallen ist und deutlich mehr Blindleistung am Transformator eingespeist werden musste. Im nächsten Ereignis hat die CVC steuernd eingegriffen und über die steuerbaren Elemente eine Erhöhung der Blindleistung bewirkt. Die Arbeit [49] beschreibt im Detail die Funktionsweise einer koordinierten Spannungsregelung und ermöglicht ein größeres Verständnis in diesem Bereich zu ergänzen. In den folgenden beiden Ereignissen wurde die Stufenschalterstellung angepasst, um die Gesamtspannung zu reduzieren. Zu diesem Zeitpunkt wurde aufgrund der CVC ein stabiler Zustand erreicht. Im fünften Ereignis wurde der Einfluss durch den Verlust dieses Dienstes dargestellt. Wichtig anzumerken ist, dass die letzten Steuersignale aus der CVC noch für 15 Sekunden in den steuerbaren Elementen erhalten bleiben. Das führt zum Zeitpunkt des sechsten Ereignisses zu einer Umstellung auf lokale Steuerung der Elemente. Dies führt erneut zu einer Spannungsbandverletzung, woraufhin die Stufenschalterstellung in den folgenden Ereignissen (7-10) schrittweise erhöht wird. Jedoch kann das Limit der Blindleistungseinspeisung nicht kompensiert werden, da weder der Kondensatorspeicher noch die zentrale Steuerung greift.

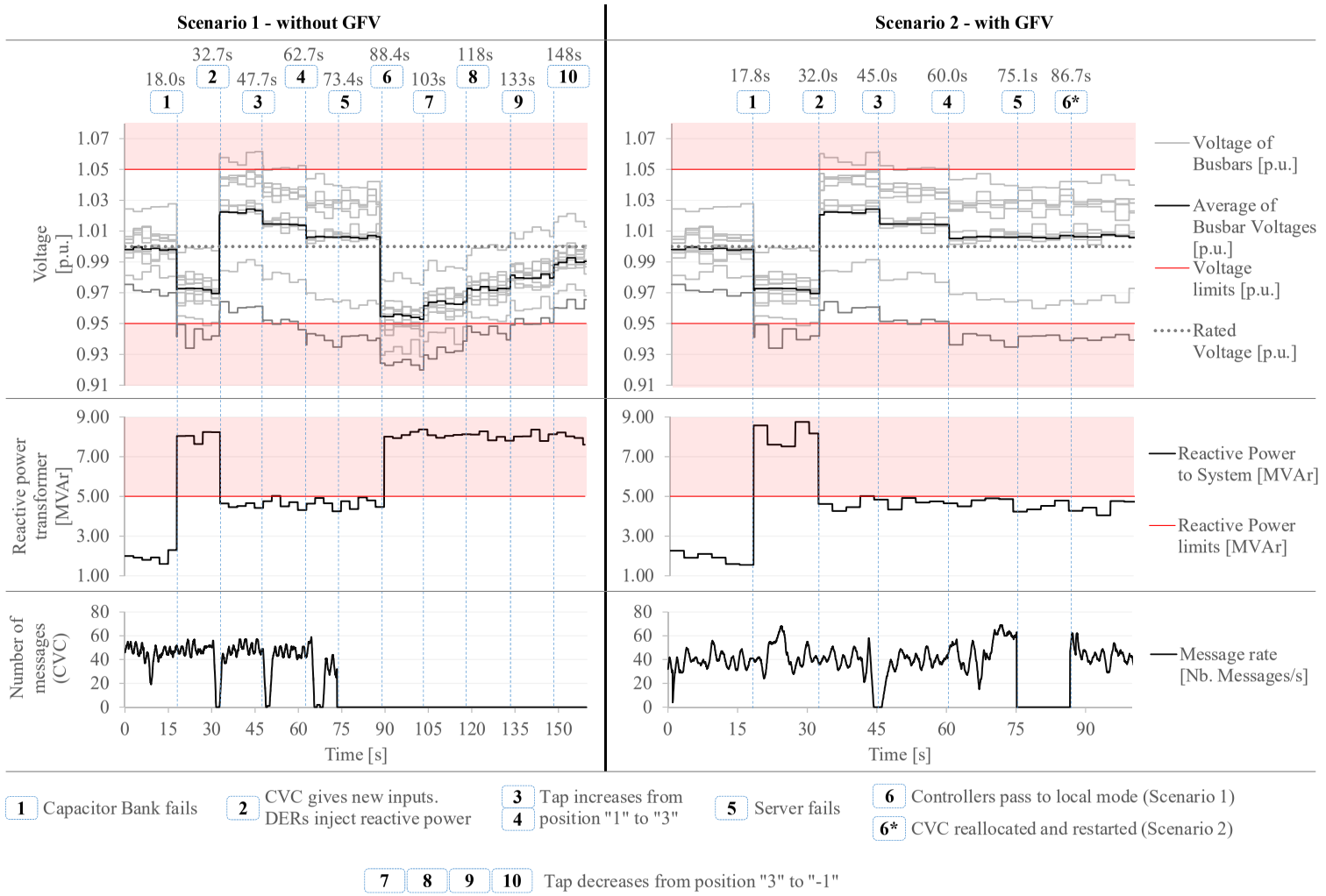

Abb. 4.5.: Ergbnisse aus der Fallstudie [8]

Im zweiten Szenario hingegen greift die GFV, um den Ausfall des CVC-Dienstes zu kompensieren. Dazu erfolgt eine Verschiebung des CVC-Dienstes auf eine neue Hardware-Instanz. Dieser Prozess entspricht dem Prozessablauf aus dem Sequenzdiagramm 4.3. Der Ausfall des CVC-Dienstes wird über das Überwachungssystem erkannt und der Orchestrierung mitgeteilt. Daraufhin werden Informationen zum Dienst und den notwendigen Funktionen abgefragt, auf deren Basis und den verfügbaren Hardware-Instanzen eine Entscheidung getroffen wird. In diesem Fall wird der Dienst auf ein anderes Hardware-Gerät verschoben und ausgeführt. Dazu werden Befehle an den lokalen Manager der neuen Hardware-Instanz gesendet.

Die aus den verwandten Arbeiten [19, 4, 8] gewonnenen Erkenntnisse beziehen sich auf den strukturellen Aufbau der GFV-Architektur und den prototypischen Aufbau einer Management-Plattform. Diese Erkenntnisse finden direkte Anwendung im nachfolgenden Verwaltungskonzept und der Umsetzung in einer geeigneten Infrastruktur 4.5.

Die Verwaltung und Bereitstellung einer Infrastruktur mit einem hohen Automatisierungsgrad erfolgt im Rahmen der Development and Operations (DevOps) [2]-Praktiken unter der Bezeichnung Infrastructure as Code (IaC). Anstelle manueller und physischer Hardwarekonfigurationen oder interaktiver Konfigurationswerkzeuge erfolgt die Konfiguration durch maschinenlesbare Beschreibungen. In der Arbeit von Guerriero et al. [50] wurde die Entwicklung von Unternehmen hin zu einem System mit IaC untersucht und wichtige Herausforderungen identifiziert. Zusätzlich wurden verschiedene Werkzeuge und deren Vor- und Nachteile beleuchtet. Im Rahmen eines Interviews [50] wurden 18 Fragen von 44 teilnehmende Institutionen aus unterschiedlichen Europäischen Projekten zu den verwendeten Praktiken, Werkzeugen und Herausforderungen beantwortet. Bezüglich der Verwendung unterschiedlicher Werkzeuge zeigt Tabelle 4.1 die Häufigkeit der Vor- und Nachteile, sowie die Summe im Verhältnis zueinander.

Bezogen auf die Herausforderungen wurden in der Veröffentlichung die folgenden sechs Kategorien untersucht:

1 - Programmierung
Bezieht sich auf die Leichtigkeit beim Erstellen von Skripten für ein verwendetes Werkzeug oder eine Sprache (z.B. deklarative oder imperative Beschreibung).

2 - Übertragbarkeit
Bezieht sich auf die Übertragbarkeit von Skripten auf verschiedene Zielinfrastrukturen (z.B. BS).

[2] In dieser Arbeit wird der englische Fachbegriff Development and Operations (DevOps) verwendet.

Tab. 4.1.: Verteilung der Verwendung von IaC-Werkzeuge nach [50]

Werkzeug	Nutzungen	Prozent
Docker	26	59.0%
Ansible	23	52.2%
Vagrant	19	43.1%
Kubernetes	18	40.9%
Chef	16	36.3%
Terraform	15	34.1%
Puppet	13	29.5%
Apache Brooklyn	9	20.0%
Packer	9	20.0%
CloudFormation	9	20.0%
TOSCA	8	18.2%
Salt	6	13.6%
Shell scripts	4	09.0%
Cloudify	3	06.8%
Octopus Deploy	1	02.3%
Azure DevOps	1	02.3%

3 - Automatisierung
Bezieht sich auf den Grad der Automatisierung beim Erstellen der Infrastruktur und der Möglichkeit Container, Virtuelle Maschinen (VMs) oder komplette Cluster automatisch mit Skripten zu erstellen.

4 - Benutzerfreundlichkeit
Bezieht sich auf die benutzerfreundliche Nutzung einer Sprache oder eines Werkzeugs.

5 - Erweiterbarkeit
Bezieht sich auf den Grad, in dem die Sprache oder das Werkzeug außerhalb seines ursprünglichen Kontexts verwendet werden kann und wie viele Freiheitsgrade es bietet.

6 - Reifegrad
Der Reifegrad bezieht sich auf die Stabilität und aktive Nutzung der Sprache oder des Werkzeugs sowie die Akzeptanz in der Community.

Insbesondere die Erkenntnisse zu den einzelnen verwendeten Werkzeugen und deren Vor- und Nachteile sind von Bedeutung für diese Dissertation. Im Kapitel 4.3 wird eine Technologieanalyse erfolgen, in der diese Ergebnisse einfließen werden. Außerdem werden die Erfahrungswerte zu den Herausforderungen sowie Vor- und Nachteile in die konzeptionelle Entwicklung des Verwaltungskonzepts aus dem Kapitel 4.2 einbezogen.

In der Arbeit [51] mit dem Titel „Monitoring and Managing IoT Applications in Smart Cities Using Kubernetes“ wird ein effizienter Ansatz zum Verwalten und

Tab. 4.2.: Vor- und Nachteile der Umfrage im Bezug auf IaC-Werkzeuge nach [50]

Werkzeug	1		2		3		4		5		6			
	+	-	+	-	+	-	+	-	+	-	+	-	∑	Verhältnis
Docker (compose)	1	2	2	–	–	–	1	2	–	–	–	–	8	0.00
Chef	–	2	–	2	–	–	–	–	1	–	2	–	7	-0.14
TOSCA	–	1	3	–	–	–	1	–	1	–	–	1	7	0.42
Ansible	1	2	2	–	–	–	–	–	1	–	–	–	6	0.33
Puppet	–	2	–	1	–	–	–	1	–	–	–	–	4	-1.00
Terraform	–	1	–	–	1	–	–	–	–	–	1	1	4	0.00
Vagrant	–	–	–	–	–	–	1	1	1	–	1	–	4	0.50
Shell-Script	–	1	1	–	2	–	–	–	–	–	–	–	4	0.50
Cloudify	1	2	–	–	–	–	–	–	1	–	–	–	4	0.00
CloudFormation	–	1	–	1	1	–	–	–	–	–	–	–	3	-0.33
Apache Brooklyn	–	–	1	–	–	–	–	–	–	1	1	–	3	0.33
Kubernetes	–	1	1	–	–	–	–	–	–	–	1	–	3	0.33
Saltstack	1	–	–	–	–	–	–	–	–	–	1	–	2	1.00
PowerShell	–	–	–	–	1	–	–	–	–	–	–	1	–	1.00
Packer	–	–	–	–	1	–	–	–	–	–	–	–	1	1.00

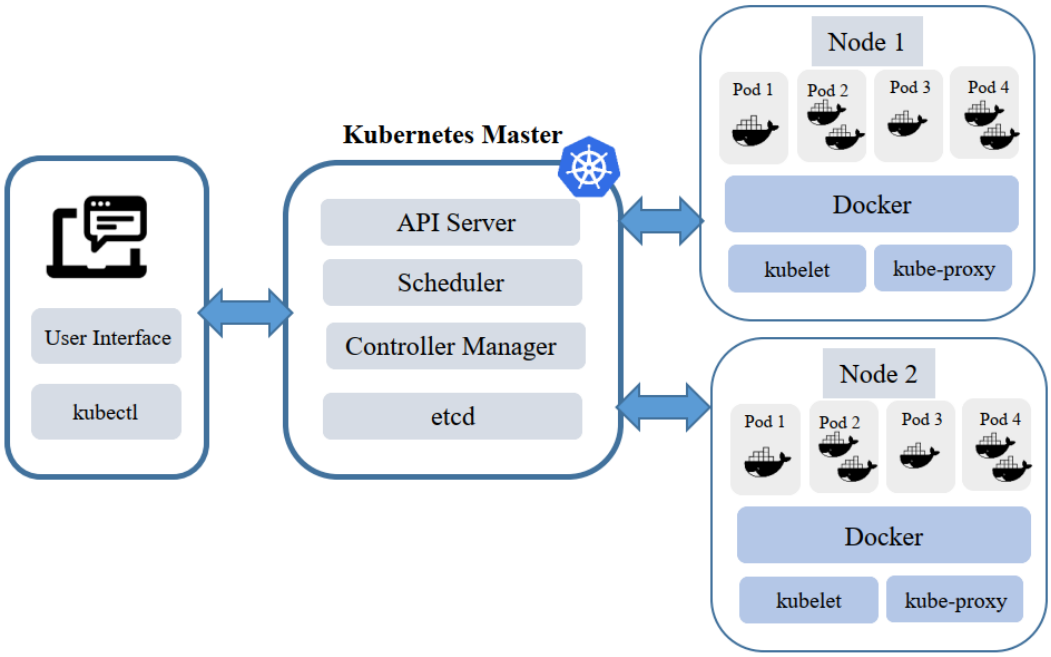

Abb. 4.6.: Kubernetes Architektur [51]

Überwachen von IoT-Anwendungen untersucht. Im Rahmen der Veröffentlichung wird die Verwendung von Docker als BS-Virtualisierung für Anwendungen analysiert. Die klusterübergreifende Verwendung von Containern wird mit dem Bedarf eines Cluster-Managements beschrieben. Einige Anforderungen an eine Cluster-Orchestrierung sind die Überwachung, Skalierung, Lastverteilung und Verwaltung von Diensten über Host-Systeme hinweg. Verwendet wird Kubernetes als Container-

Orchestrierungsplattform. Die Abbildung 4.6 zeigt den verwendeten Architekturaufbau, in dem eine zentrale Kubernetes-Instanz die Verwaltung von zwei Knoten übernimmt. Auf jedem Knoten werden Docker-Instanzen zur Verwaltung der Container eingesetzt. Die Ergebnisse aus dieser Arbeit sind insbesondere aufgrund der vergleichbaren Architektur und Hardwarekomponenten relevant. So wurden beispielsweise RPIs als IoT-Geräte verwendet. In Bezug auf die Überwachung der IoT-Dienste wurde eine webbasierte grafische Oberfläche auf Basis von Kubernetes Dashboard verwendet. Diese ermöglicht die Ermittlung und Darstellung der Systemressourcen für eine weitere Verarbeitung.

4.2 Verwaltungskonzept

Aufbauend auf dem Virtualisierungskonzept aus Kapitel 3.2.3, den ergänzenden Informationen aus der VGF-Architektur und den weiterführenden Informationen zu den IaC-Ansätzen wird in diesem Kapitel das Verwaltungskonzept beschrieben. Das Gesamtkonzept ist in Abbildung 4.7 dargestellt und enthält weiterhin die drei Schichten (Infrastruktur, Dienste und Verwaltung), wobei nur die Verwaltungsschicht erweitert wurde. Im Folgenden wird die Verwaltungsschicht detailliert beschrieben.

Verwaltungsschicht
Die Verwaltungsschicht übernimmt weiterhin die Aufgabe der lokalen Dienste-Verwaltung, wie im Virtualisierungskonzept beschrieben. Diese Betrachtung ermöglicht ausschließlich die Umsetzung lokaler Dienste, Konfigurationen und Prozesse. Dabei bildt die genutzte Hardware und das darauf ausgeführte BS die Grenzen für die lokale Betrachtung. Dieses Verwaltungskonzept ergänzt die lokale Betrachtung um eine zentrale Orchestrierung, welches die Möglichkeit bietet, weitere Dienste, Konfigurationen oder Prozesse dem Gerät über Schnittstellen bereitzustellen. Die Schnittstellen sind dabei essentiell und es müssen geeignete Formate zwischen beiden Instanzen der Dienste-Verwaltung bereitgestellt werden. Drei Teilaufgaben unterliegen der zentralen Orchestrierung: die Überwachung von Systemressourcen, die kontinuierliche Überprüfung von Automatisierungsroutinen und die Umsetzung von Steuerungsbefehlen. Zusätzlich werden Dienste-Beschreibungen und ein Dienste-Verzeichnis benötigt. Diese werden zusammen mit den Teilaufgaben nachfolgend genauer beschrieben:

Überwachung
Die Überwachung dient der Erfassung und Verarbeitung von Informationen über die

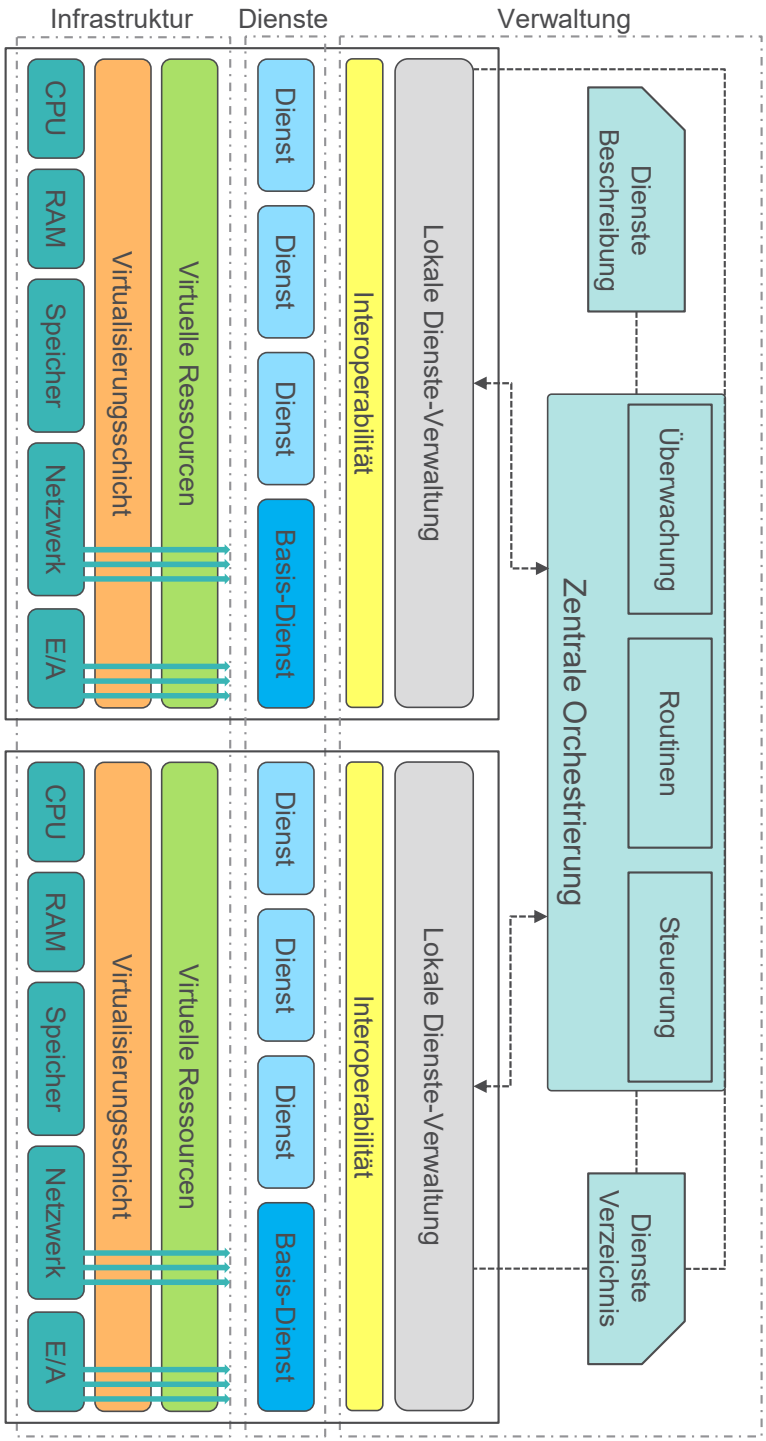

Abb. 4.7.: Verwaltungskonzept zur Orchestrierung von Diensten über mehrere Stationen hinweg

Systemressourcen verknüpfter Stationsgeräte und der darauf ausgeführten virtualisierten Dienste. Im Rahmen dieses Konzeptes sind CPU, RAM, Festplattenspeicher, Netzwerkkonfigurationen und E/A-Schnittstellen als Systemressourcen definiert. Die Erfassung beinhaltet die Ermittlung auf dem verknüpften Gerät, die Übertragung und Bereitstellung an die interne Verarbeitung über die Steuerung und Durchführung von Routinen. Die Interpretation der Systemressourcen durch Grenzwerte, Ausfälle oder Verbindungsprobleme fällt unter die Verarbeitung. Beispielsweise erfolgt eine Meldung der Nichtverfügbarkeit eines Gerätes im Fall von Verbindungsproblemen oder eine Meldung über die Nichtverfügbarkeit eines einzelnen Dienstes.

Routinen
Die Routinen bauen auf den erfassten und verarbeiteten Informationen der Überwachung auf und führen in diskreten Zeitschritten oder basierend auf Ereignissen Überprüfungen durch. Diese Überprüfungen dienen dem Erhalt eines spezifizierten Zustands. Beispielsweise soll ein Dienst mindestens auf zwei Geräten gleichzeitig ausgeführt werden und darf nur auf einer ausgewählten Menge von Geräten ausgeführt werden. Dann würde der Ausfall eines Dienstes folgende Routinen auslösen: Im ersten Schritt erfolgt eine Wiederherstellung des Dienstes auf dem gleichen Gerät. Notwendige Ausführungsbefehle werden dazu an die Steuerung weitergeleitet. Führt dies nicht zum Erfolg, können weitere zur Verfügung stehende Geräte ausgewählt werden. Der spezifizierte Zustand ist Teil der Dienste-Beschreibung und ist abhängig von dem gewünschten Anwendungsfall.

Steuerung
Die Steuerung übernimmt die Koordination von Befehlen an die einzelnen verknüpften Geräte, genauer den Austausch von Steuerungsbefehlen an die lokale Dienste-Verwaltung der einzelnen verknüpften Geräte. Befehle können dabei auf zwei unterschiedliche Arten erfolgen: durch die Bereitstellung automatisierter Routinen, die den spezifizierten Zustand aus der Dienste-Beschreibung erhalten, oder durch einen externen Eingriff wie die manuelle Spezifikation eines Benutzers über eine Befehlszeilenschnittstelle oder einer grafischen Oberfläche. Neben einfach verständlichen Steuerbefehlen wie Starten, Stoppen, Verschieben oder Replizieren eines Dienstes können Steuerbefehle mehrschichtige komplexe Konfigurationen beinhalten, beispielsweise Einschränkungen, Zusicherungen oder Flexibilität der genutzten Systemressourcen, aber auch Schnittstellen, Systemrechte oder Verschlüsselungen. Der Umfang möglicher Steuerungen ist abhängig von der technologischen Implementierung und muss auf die zu unterstützenden Anwendungsfälle ausgerichtet werden.

Dienste-Beschreibung
Die Dienste-Beschreibung beinhaltet alle notwendigen Informationen über einen Dienst. Diese Informationen umfassen die Konfiguration über die Ausführung des Dienstes und die gewünschte Instanz der Dienste. Wie bereits im Abschnitt 4.2 zur Steuerung beschrieben, kann eine Dienste-Beschreibung sehr komplexe Konfigurationen beinhalten, die abhängig von der implementierten Technologie unterschiedlich realisiert wird. Die Bereitstellung einer solchen Beschreibung ist essentiell zur Umsetzung der Dienste im Virtualisierungskonzept aus Kapitel 3.2.3.

Dienste-Verzeichnis
Das Dienste-Verzeichnis stellt einen Speicherort zur Ablage von Dateien und Informationen eines Dienstes bereit. Im Kapitel 3.3 wurde bereits der Begriff „Repository“ eingeführt und erklärt, der im Kontext der Technologieanalyse betrachtet wurde. Das Dienste-Verzeichnis kann als ein privates oder öffentliches Repository zur Bereitstellung von Daten eines Dienstes verstanden werden. Die genaue Umsetzung des Dienste-Verzeichnisses ist abhängig von der genutzten Technologie und kann auch eine einfache Ablage in einem Verzeichnis sein. Wichtig ist nur, dass Zugriffe durch die zentrale Orchestrierung und der lokalen Dienste-Verwaltung erfolgen können. Die zentrale Orchestrierung muss über die Verfügbarkeit von Diensten in Kenntnis gesetzt werden, während die lokale Dienste-Verwaltung die Möglichkeit haben muss, auf die Daten und Inhalte eines Dienstes zuzugreifen und diese für die Ausführung auf den Geräten zu verwenden.

4.3 Technologieanalyse

In diesem Abschnitt werden unterschiedliche Technologien hinsichtlich ihrer Eignung für das Verwaltungskonzept untersucht. Dazu werden nachfolgend die Vergleichskriterien und die Technologien näher beschrieben. Abschließend erfolgt eine Bewertung und Auswahl der genutzten Technologien.

4.3.1 Vergleichskriterien

Einige der Vergleichskriterien wurden bereits in Kapitel 3.3.1 im Vergleich unterschiedlicher Containertechnologien verwendet. Im Folgenden werden nur abweichende Kriterien beschrieben, die spezifisch für den Vergleich unterschiedlicher Orchestrierungstechnologien relevant sind.

Orchestrierungstyp
Das Kriterium Orchestrierungstyp beschreibt, welche Art der Arbeitslast durch eine Technologie verwaltet wird. Beispiele für Arbeitslasten sind die unterschiedlichen Virtualisierungstypen wie Hypervisor Typ 1, Typ 2 oder BS-Virtualisierung, auch als Container oder VMs bekannt, oder grundlegender einzelne Prozesse, Jobs oder Ressourcen.

Container-Unterstützung
In Bezug auf die Orchestrierung von Containern gibt es einige Orchestrierungslösungen, die nur spezifische Containertechnologien unterstützen. Die gängigsten Technologien wie Docker, CRI-O, Containerd und Rocket wurden betrachtet. Dieses Kriterium ist aus Sicht des Verwaltungskonzepts erst im Hinblick auf die Umsetzung im Gesamtsystem wichtig, da die ausgewählte Technologie des Virtualisierungskonzepts mit der ausgewählten Orchestrierungstechnologie kompatibel sein muss.

Funktionsumfang
Der Funktionsumfang beschreibt den Umfang der Dienstleistungen, die durch eine Orchestrierungstechnologie bereitgestellt werden. Die Einschätzung wird unterteilt in die Ausprägungen „Grundlegend“, „Umfangreich“ und „Sehr umfangreich“ und basiert auf eine Experteneinschätzung, da eine durchgängige Metrik nicht eindeutig erstellbar ist. Die Grundlage der Einschätzung beinhaltet Informationen aus der Community, den beschriebenen Dienstleistungen und Informationen aus der Dokumentation. Betrachtet wurden Grundlagen zur Verwaltung von Arbeitslasten wie Lastausgleich, automatische Skalierung, Selbstheilung, Überwachung, Rückabwicklung von Änderungen, Speicherverwaltung und weitere.

Komplexität
Die Komplexität ist im Zusammenhang mit dem benötigten Wissen oder auch der Benutzerfreundlichkeit zu verstehen. Die Einschätzung ist unterteilt in die Ausprägungen „Anfänger“, „Fortgeschritten“ und „Experte“. Als Grundlage für diese Einschätzung wurden eigene Erfahrungen mit diesen Technologien sowie Informationen aus der Community und der Dokumentation verwendet.

4.3.2 Technologieüberblick

Im Folgenden werden die unterschiedlichen Technologien kurz beschrieben.

Docker Swarm
Docker Swarm ist eine Orchestrierungstechnologie, die es ermöglicht, Containeranwendungen über Cluster von Maschinen hinweg zu verteilen, zu verwalten und zu

skalieren. Dabei ist Docker Swarm ein integrierter Dienst für Docker-Container und ein fester Bestandteil der Docker-Implementierung. Es wird eine dienstorientierte Architektur verwendet, sodass Container in Diensten organisiert werden und diese als logische Einheiten von Anwendungsbestandteilen fungieren. Docker Swarm bietet die grundlegenden Funktionalitäten einer Orchestrierungstechnologie wie automatische Lastverteilung, Skalierung, schrittweise Aktualisierungen und einige Sicherheitsfunktionen.

Kubernetes
Die Cloud Native Computing Foundation verwaltet die Entwicklung der Open-Source-Containerorchestrierungstechnologie Kubernetes, auch bekannt als „K8s“. Die automatisierte Bereitstellung, Skalierung und Verwaltung von Containeranwendungen ist die primäre Aufgabe von Kubernetes und unterstützt dabei unterschiedliche Containertechnologien wie Docker, CRI-O und Containerd. Ein zentrales Konzept ist der „Pod“ als kleinste Bereitstellungseinheit, die einen oder mehrere Container ausführen kann. Ein Pod ermöglicht die gemeinsame Nutzung von Ressourcen und Netzwerkinfrastruktur. Kubernetes bietet eine sehr umfangreiche Anzahl an Funktionalitäten, darunter ein umfangreiches Ökosystem, Erweiterung und Schnittstellen zu anderen Systemen wie Helm-Charts für eine einfache Bereitstellung von Anwendungen oder Prometheus für die Überwachung der Systemressourcen. Weitere Funktionalitäten beinhalten die Abstraktion von Netzwerkkonnektivitäten mittels eigener Netzwerkschnittstellen sowie die Möglichkeit, einen vorherigen stabilen Systemzustand wiederherzustellen. Kubernetes zählt zu den bekanntesten Technologien mit hoher Leistungsstärke und Flexibilität, unterstützt durch eine große Community. Allerdings erhöhen die bereitgestellte Flexibilität und die zahlreichen Möglichkeiten auch die Komplexität und erschweren die einfache Nutzbarkeit.

Apache Mesos
Apache Mesos ist keine reine Containerorchestrierungstechnologie, sondern stellt eine Plattform zur Ressourcenverwaltung und -isolation bereit, die für die Skalierung und Verwaltung von verteilten Anwendungen und Arbeitslasten entwickelt wurde. Unterstützt werden verschiedene Arten von Arbeitslasten wie Container, VMs oder auch herkömmliche Anwendungen. Die Architektur entspricht einem Master-Slave-Modell zur effizienten Verwaltung und Aufteilung von Ressourcen. Die Unterstützung der Containerorchestrierung erfolgt durch die Integration von Kubernetes oder Marathon. Damit ist Apache Mesos ein weiterer Layer und bietet eine hohe Flexibilität und Erweiterbarkeit. Weiterhin werden eine gute Skalierbarkeit und ein effektives Ressourcenmanagement ermöglicht. Aufgrund der vielschichtigen Architektur besteht eine hohe Komplexität der Technologie und setzt ein hohes Vorwissen voraus.

Nomad
Ähnlich zu Apache Mesos ist auch HashiCorp Nomad ein Orchestrierungssystem ohne Spezialisierung auf Container. Nomad übernimmt die Skalierung von Arbeitslasten in verteilten Umgebungen und bietet eine dezentrale, skalierbare und flexible Plattform für die Bereitstellung von Anwendungen. Die Anwendungen können als Container, VMs oder aus eigenständigen Binärdateien bestehen. Nomad unterstützt den grundlegenden Funktionsumfang wie automatische Lastverteilung, Skalierung, schrittweise Aktualisierungen und einige Sicherheitsfunktionen. Durch eine klare Syntax der deklarativen Job-Spezifikation zur Beschreibung der Arbeitslasten stellt Nomad einen benutzerfreundlichen Einstieg für Anfänger bereit und reduziert dadurch die Komplexität.

Amazon ECS
Amazon Elastic Container Service (ECS) ist eine vollständig verwaltete Containerorchestrierungstechnologie von Amazon Web Services (AWS). ECS ermöglicht die Bereitstellung, Verwaltung und Skalierung von Containeranwendungen ausschließlich unter Docker. Die Architektur ist in einzelne Bestandteile wie ECS-Cluster, ECS-Task, ECS-Service, etc. unterteilt und stark mit den Cloud-Diensten von Amazon verbunden. Auch wenn Teile des ECS Open Source verfügbar sind, benötigen einige Funktionen die Integration durch lizenzierte Amazon-Dienstleistungen. Unabhängig von den Nutzungsbedingungen bietet Amazon ECS sehr umfangreiche Funktionalitäten wie Beobachtbarkeit, Lastverteilung, Identitätsmanagement, rollenbasierte Zugriffskontrolle, Versionsverwaltung und Sicherungskopien. Die starke Einbettung in die Amazon-Dienstleistungen ermöglicht einen einfachen Einstieg in diese Technologie, solange die benötigten lizenzierten Cloud-Dienste genutzt werden.

Docker Compose
Das von Docker, Inc. entwickelte Werkzeug zur Definition, Konfiguration und Bereitstellung von mehreren Docker-Containern ist keine Orchestrierungstechnologie. Es werden nur eingeschränkte Funktionalitäten wie die Skalierung und Konfiguration auf einem Gerät zur Verfügung gestellt. Durch eine einfache Syntax und intuitive Befehle zur Beschreibung mehrerer auszuführender Container bietet Docker Compose eine geringe Komplexität mit einem anfängerfreundlichen Einstieg. Docker Compose eignet sich insbesondere als Entwicklungs- und Testumgebung, um anschließend in komplexere Orchestrierungstechnologien für eine Produktionsbereitstellung überführt zu werden.

4.3.3 Technologieauswahl

Tab. 4.3.: Auswertung der Technologieanalyse

Kriterien	Technologie		
	Docker-Swarm	**Kubernetes**	**Apache Mesos**
Orchestrierungstyp	Container	Container	Container, Jobs, VMs
Nutzungsbedingungen	Open Source	Open Source	Open Source
BS-Unterstützung	Win, Linux, Mac	Win, Linux, Mac	Linux, Mac
Veröffentlichungsstatus	Aktiv	Aktiv	Aktiv
Dokumentation	Hervorragend	Hervorragend	Akzeptabel
Community-Unterstützung	Gut	Hervorragend	Akzeptabel
Container-Unterstützung	Docker	Docker, CRI-O, etc.	Docker, rkt, etc.
Funktionsumfang	Grundlegend	Sehr umfangreich	Sehr umfangreich
Komplexität	Anfänger	Fortgeschritten	Experte

Kriterien	Technologie		
	Nomad	**Amazon ECS**	**Docker Compose**
Orchestrierungstyp	Container, Jobs, VMs	Container	Container
Nutzungsbedingungen	Open Source	Lizenziert	Open Source
BS-Unterstützung	Win, Linux, Mac	AWS-konforme BS	Win, Linux, Mac
Veröffentlichungsstatus	Aktiv (2022)	Aktiv	Aktiv
Dokumentation	Akzeptabel	Gut	Gut
Community-Unterstützung	Gut	Gut	Gut
Container-Unterstützung	Docker, CRI-O, rkt	Docker	Docker
Funktionsumfang	Grundlegend	Sehr umfangreich	Eingeschränkt
Komplexität	Anfänger	Anfänger	Anfänger

In Anbetracht der Kriterien aus Tabelle 4.3 und des Technologieüberblick wurde eine Auswahl der im Rahmen dieser Arbeit genutzten Orchestrierungstechnologien getroffen.

1. Docker-Swarm ist fester Bestandteil der Containertechnologie Docker und bietet damit ideale Voraussetzungen für eine minimale ressourcensparende Implementierung innerhalb einer Systemlandschaft. Die geringe Komplexität, hervorragende Dokumentation und eine gute Unterstützung innerhalb der Community ermöglichen Anfängern die leichte Verwendung. Im Hinblick auf den Funktionsumfang ist Docker-Swarm limitiert und im direkten Vergleich Technologien wie Kubernetes oder Amazon ECS unterlegen. Ebenso ist die Abhängigkeit von der ausschließlichen Verwendung von Docker-Containern eine Einschränkung.

2. Nahezu alle Kriterien der Technologie Kubernetes sind überdurchschnittlich erfüllt. Die erhöhte Komplexität hat aufgrund der hervorragenden Dokumentation und Community-Unterstützung nur einen geringen Einfluss auf die Benutzerfreundlichkeit.

3. Apache Mesos wird aufgrund der hohen Komplexität nicht weiter betrachtet.
4. Die unregelmäßige aktive Weiterentwicklung und der geringe Funktionsumfang bei Nomad schließen diese Technologie von der weiteren Betrachtung aus.
5. Eine wichtige Anforderung an die verwendeten Technologien ist die flexible und frei verfügbare Nutzung, die bei Amazon ECS durch die starke Einbindung in die Amazon Cloud-Dienstleistungen nicht gegeben ist.
6. Docker Compose wird aufgrund des geringen Funktionsumfangs und der lokalen Betrachtung nicht weiter betrachtet.

4.3.4 Struktureller Aufbau von Kubernetes

Im Folgenden wird Kubernetes als Orchestrierungstechnologie detailliert beschrieben und die bereitgestellte Architektur untersucht. Abbildung 4.8 zeigt die Architektur eines Kubernetes-Clusters. Die einzelnen Komponenten werden im Nachfolgenden näher erläutert.

Cluster
Ein Cluster beschreibt den physischen oder logischen Zusammenschluss von Computern oder Systemressourcen, die zusammenarbeiten, um Arbeitslasten als Container zu verarbeiten und zu verwalten. Innerhalb eines Clusters gibt es mindestens eine Steuereinheit (Control Plane) und beliebig viele Knoten (Nodes), die auch als „Worker Nodes“ bezeichnet werden.

Control Plane
Die Steuereinheit des Clusters ist für die gesamte Koordination und Verwaltung verantwortlich. Die Aufgaben der zentralen Steuereinheit sind in einzelne Teilkomponenten aufgeteilt. Es ist nicht zwingend erforderlich, dass alle Teilkomponenten auf einer Maschine ausgeführt werden. Die Steuereinheit kann auch verteilt agieren.

kube-api-server
Der API-Server ist das Front-End für die Kubernetes-Steuereinheit und nimmt Befehle von Benutzern, anderen Teilkomponenten und den Kubelets entgegen.

Scheduler
Der Scheduler trifft Entscheidungen über die Verteilung der Arbeitslast von Containern auf die einzelnen Knoten basierend auf den Ressourcenanforderungen, Richtlinien und anderen Faktoren.

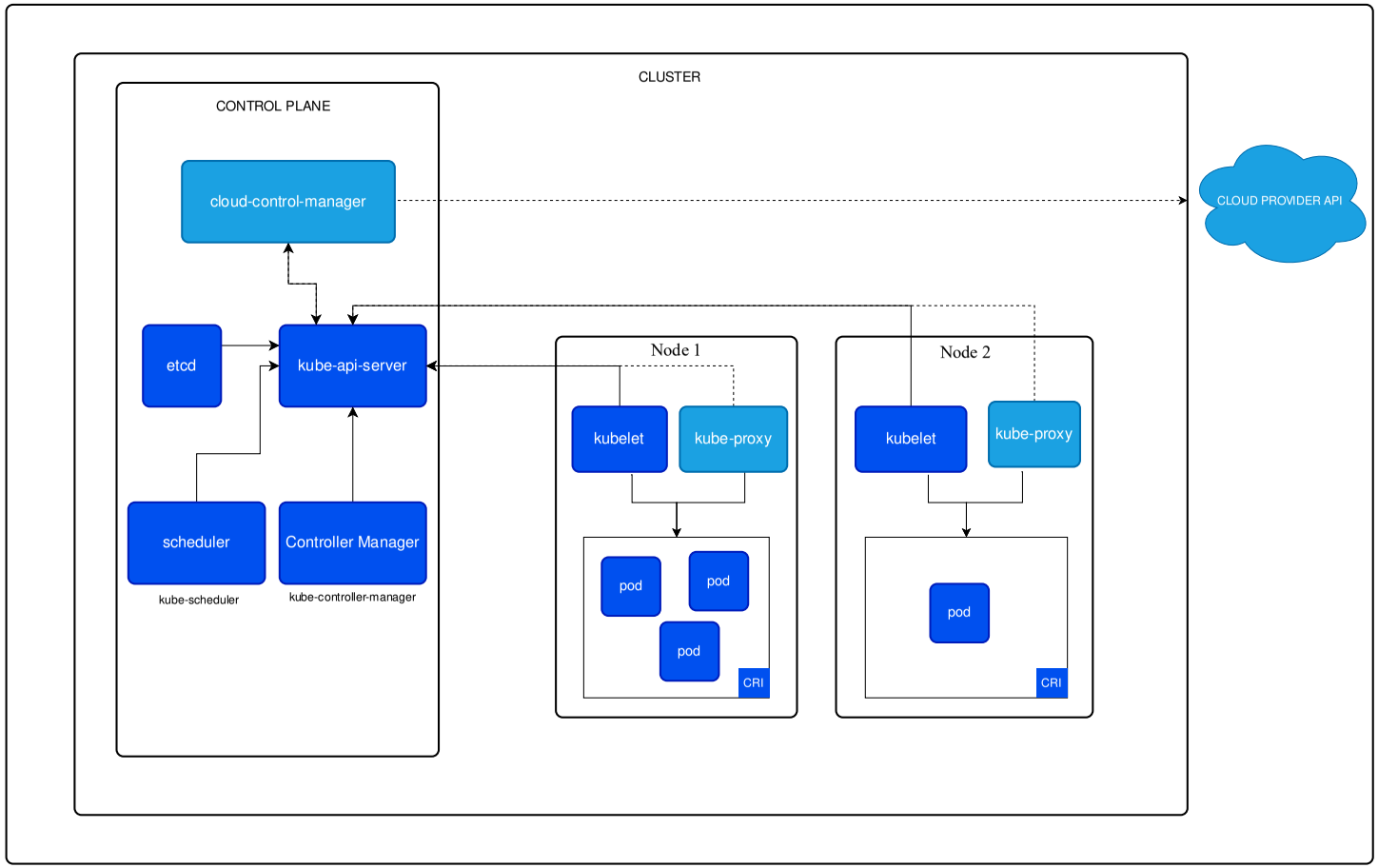

Abb. 4.8.: Kubernetes Cluster Architektur [52]

Control Manager
Der Control Manager überwacht alle laufenden Prozesse und Arbeitslasten und reagiert auf Änderungen im Cluster. Wenn der tatsächliche Zustand nicht mit dem gewünschten Zustand aus den Anforderungen, Richtlinien etc. übereinstimmt, wird eine Routine ausgelöst, um den gewünschten Zustand anzupassen.

Etcd
Etcd ist eine verteilte Datenbank, die den gesamten Zustand innerhalb eines Clusters speichert. Etcd wird von Kubernetes als zuverlässiger Speicher für Konfigurationen, Anforderungen und den aktuellen Status im Cluster genutzt.

Nodes
Die Knoten sind entweder physische oder virtuelle Maschinen und führen die Arbeitslasten als Container aus. Jeder Knoten ist mit der Steuereinheit verbunden und führt die nachfolgenden Komponenten aus.

kubelet
Das Kubelet überwacht die laufenden Container und überprüft, ob diese sich im gewünschten Zustand befinden. Jedes Kubelet steht im direkten Austausch mit der Steuereinheit und übernimmt die Kommunikation zwischen einem Knoten und der Steuereinheit.

Pod
Ein Pod ist in Kubernetes die kleinste bereitstellbare und ausführbare Einheit. Jeder Pod kann die Arbeitslast eines oder mehrerer Container enthalten und verwaltet die dem Pod zugewiesenen Systemressourcen.

kube-proxy
Der Kube-Proxy ermöglicht den Austausch zwischen den Pods eines Clusters über eine Netzwerkkommunikation, die es ermöglicht, dass Anwendungen sich gegenseitig erreichen.

Container Runtime Interface (CRI)
Das CRI ist eine Schnittstelle zur Interaktion zwischen der Steuereinheit und der Containertechnologie. Diese Schnittstelle wurde in Kubernetes integriert, um die Modularität zwischen unterschiedlichen Container-Laufzeitumgebungen zu fördern. CRI spezifiziert eine Reihe von API-Spezifikationen und Standards, die implementiert werden müssen, damit eine Containertechnologie an Kubernetes angebunden werden kann.

4.4 Anforderungen an die Hardware- und Software-Infrastruktur

Dieses Kapitel beschreibt die notwendigen Anforderungen der Hardware-Infrastruktur zur Umsetzung von Kubernetes in einem domänenspezifischen Kontext. Die Anforderungen beziehen sich auf die notwendigen Systemressourcen und das unterstützte BS. Anforderungen aus dem domänenspezifischen Kontext beziehen sich auf die verfügbaren Hardware-Geräte der Stationsautomatisierung und eventuelle Einschränkungen. Eine Analyse der benötigten Systemressourcen durch Kubernetes innerhalb eines Stationsgeräts ist in Kapitel 6 dargestellt.

4.4.1 Anforderungen an das Betriebssystem

Kubernetes lässt sich auf unterschiedlichen BS implementieren. Im Bezug auf Docker und Containerd wurde bereits eine Auswahl auf ein Linux-basiertes System in Kapitel 3.5.1 durchgeführt. Um Inkompatibilitäten zu vermeiden, werden weitere Untersuchungen für Kubernetes im Kontext eines Linux-basierten Systems durchgeführt. In Bezug auf die notwendigen und optionalen Anforderungen aus dem Abschnitt 3.4 benötigt Kubernetes die folgenden Kernel-Module.

- OverlayFS: Zur Erstellung eines Dateisystems der verwendeten Containertechnologie.
- Iptables: Zur Konfiguration der Netzwerkkommunikation und Sicherheitsrichtlinien zwischen Containern, Knoten und dem Host-Netzwerk.
- Ip_vs: Zur Ausführung des Lastausgleichs und der Verteilung des Netzwerkverkehrs innerhalb des Clusters.
- Cgroups: Zur Verwaltung und Zuweisung von Systemressourcen, so wie es innerhalb von Pods für die Container notwendig ist.

4.5 Umsetzung des Verwaltungskonzepts

In diesem Kapitel wird die Implementierung einer zentralen Orchestrierung basierend auf Kubernetes beschrieben. Diese Thematik ist in drei Bereiche unterteilt: die verwendete Hardware-Infrastruktur, die Implementierung von Kubernetes auf dieser Hardware und die Verwaltung von Diensten über die Management-Infrastruktur.

4.5.1 Hardware-Infrastruktur

Im Rahmen dieser Arbeit wird eine getrennte Hardware-Infrastruktur für die Control Plane und die Knoten von Kubernetes genutzt. Für die Knoten wird die gleiche Hardware-Infrastruktur wie auch für die Containertechnologien verwendet. Die genauen Angaben sind im Kapitel 3.5.1 beschrieben. Aufgrund der zentralen Betrachtung der Control Plane und der höheren Anforderungen an die Systemressourcen ist die Control Plane auf einem Server implementiert. Der Server basiert auf einem Ubuntu 20.04 LTS BS und hat die Möglichkeit, dynamisch mehr Ressourcen anzufordern.

4.5.2 Implementierung von Kubernetes

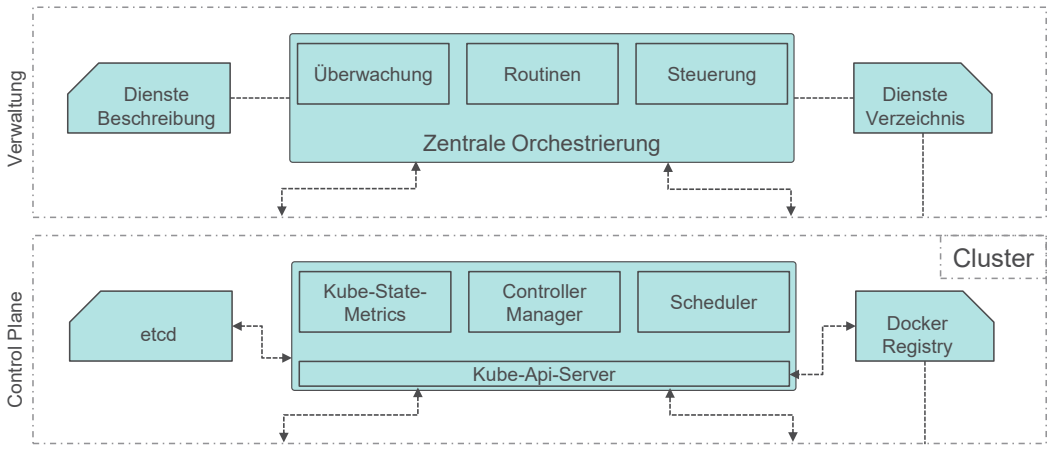

Abb. 4.9.: Umsetzung der Kubernetes Control Plane in das Verwaltungskonzept

Unter Berücksichtigung der Hardware-Infrastruktur wurde die Control Plane auf einem Server mit Ubuntu 20.04 LTS installiert. Die einzelnen Komponenten der Managementarchitektur wurden wie in Abbildung 4.9 dargestellt umgesetzt. Die Speicherung der Beschreibungen zu den Diensten sowie des Systemzustands erfolgt durch die verteilte Datenbank etcd. Weitere Informationen zu den Dienstebeschreibungen werden im Kapitel 4.5.3 erläutert. Die Funktionalität der Überwachung im Verwaltungskonzept wird durch Kube-State-Metrics übernommen, welches die grundlegende Überwachung von Containern, Pods und Systemressourcen bereitstellt. Kubernetes ermöglicht auch die Bereitstellung der Überwachung durch Drittanbietersoftware wie Prometheus oder cAdvisor. Die im Konzept beschriebenen Routinen werden im Wesentlichen durch den Controller Manager realisiert, der regelmäßig einen Abgleich zwischen Ist- und Sollzustand durchführt und Steuerungseingriffe

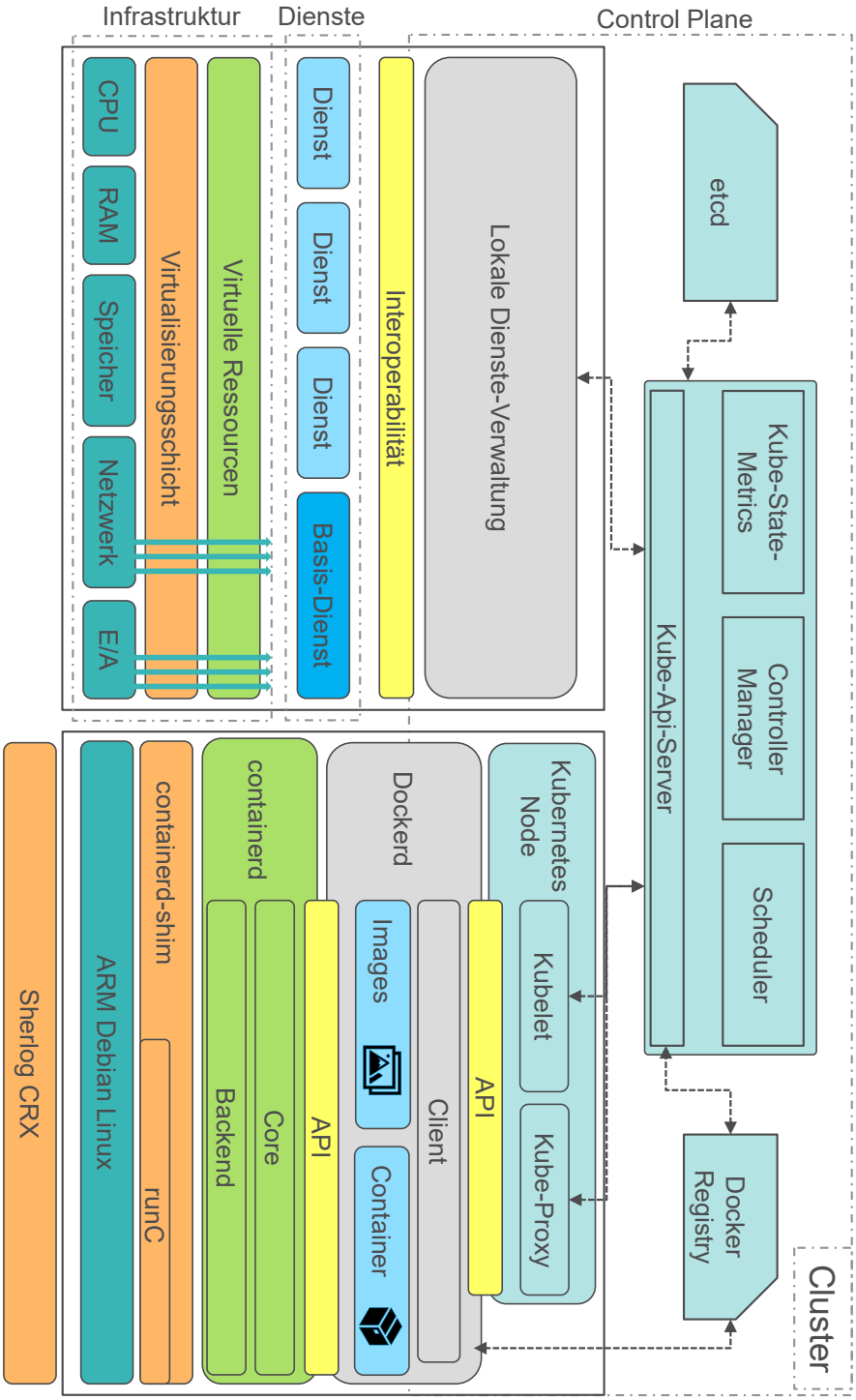

Abb. 4.10.: Umsetzung von Kubernetes in das Gesamtkonzept

anfordert, um Abweichungen entgegenzuwirken. Die Verteilung der Arbeitslast und die Umsetzung von Steuerbefehlen werden durch den Scheduler übernommen. Sämtlicher Datenaustausch zwischen den Komponenten und den Knoten erfolgt über den Kube-Api-Server. Das Diensteverzeichnis ist nicht fester Bestandteil von Kubernetes und wird durch eine Docker Registry bereitgestellt. Innerhalb der Registry werden Images in einer Umgebung mit Versionskontrolle abgelegt. Docker kann Images direkt aus der Registry abfragen und beziehen.

Die Abbildung 4.10 erweitert die Beschreibung zur Implementierung von Kubernetes um die Kubernetes-Knoten. Steuerungsbefehle werden über den Api-Server an die Kubelet-Instanz gesendet. Die Kubelet-Instanz führt alle Befehle der Control Plane aus, wie Anweisungen zum Starten, Stoppen und Überwachen von Containern. Kubelet ist für die Pods zuständig und steht im direkten Austausch mit der Containertechnologie über die standardisierte CRI-Schnittstelle. Der Kube-Proxy stellt sicher, dass die Netzwerkkommunikation und Arbeitslast zwischen den Pods entsprechend den Spezifikationen verteilt sind.

4.5.3 Verwaltung von Diensten

Die Verwaltung von Diensten über mehrere Stationen hinweg sollte idealerweise nicht durch manuelle Einstellungen oder grafische Oberflächen erfolgen, sondern durch codebasierte Beschreibungen, wie sie in der Beispielbeschreibung 4.1 dargestellt sind. Solche Beschreibungen können als Vorlage verwendet und entsprechend den Anwendungsfällen konfiguriert werden. Einige grafische Oberflächen unterstützen die Extraktion solcher Beschreibungen basierend auf manuell vorgenommenen Konfigurationen. Angesichts skalierender Systeme im Verteilnetz mit tausenden Stationsgeräten ist eine langfristige Integration dieser Beschreibungen in standardisierte Engineeringprozesse, wie sie im Kapitel 5 beschrieben werden, empfehlenswert.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: example-deployment
5    labels:
6      app: example
7  spec:
8    replicas: 3  # Anzahl der Replikate (Pod-Instanzen)
9    selector:
10     matchLabels:
```

```
11        app: example
12    template:
13      metadata:
14        labels:
15          app: example
16      spec:
17        containers:
18        - name: example-container
19          image: nginx:latest  # Docker-Image für den
              Container
20          ports:
21          - containerPort: 80  # Port, auf dem der
              Container lauscht
22          resources:
23            requests:
24              memory: "64Mi"   # Minimale anfängliche
                  Speicheranforderung für den Container
25              cpu: "250m"      # Minimale anfängliche CPU
                  -Anforderung für den Container
26            limits:
27              memory: "128Mi"  # Maximale Speichergrenze
                  für den Container
28              cpu: "500m"      # Maximale CPU-Grenze für
                  den Container
```

Skript 4.1: Beispiel einer Kubernetes-Konfigurationsdatei

Die IaC-Beschreibung lässt sich in verschiedene Teilbereiche aufteilen, die im Folgenden kurz beschrieben werden. Der Bereich 1-6 stellt allgemeine Metainformationen wie Version, Bezeichner, Namen und Labels bereit, die für die interne Verarbeitung innerhalb des Kubernetes-Clusters genutzt werden. Ab Zeile 7 können Spezifikationen eines Kubernetes-Pods vorgenommen werden. In diesem Beispiel ist spezifiziert, dass immer drei Instanzen eines Pods gleichzeitig im Kubernetes-Cluster laufen. Ab Zeile 16 erfolgt die Spezifikation der Container-Instanz. Diese Spezifikation ist in Kubernetes sehr umfangreich, und nicht alle Parameter sind hier aufgelistet. Eine vollständige Liste aller möglichen Parameter ist im [53] API-Überblick von Kubernetes zu finden. Die für diese Arbeit grundlegenden Parameter sind unter anderem das verwendete Image (Zeile 19), die Beschreibung der Ports (Zeile 20-21) und die Spezifikation der Ressourcenanforderungen und -einschränkungen (ab Zeile 22).

Die Verwaltung der Dienste im Hinblick auf die einzelnen Prozessschritte wird ausführlich im Abschnitt 5.3.3 beschrieben und an dieser Stelle nicht näher ausgeführt.

4.6 Zusammenfassung

Dieses Kapitel hat die lokale Bereitstellung von Diensten einzelner Stationsgeräte um die zentrale Orchestrierung über Stationen hinweg erweitert. Basierend auf bestehendem Wissen im Bereich Edge-Computing, Microservice-Management und einer Vielzahl von DevOps-Strategien wurde ein Verwaltungskonzept entwickelt. Dieses Verwaltungskonzept wurde unter Berücksichtigung einer ausführlichen Technologieanalyse durch Kubernetes umgesetzt und entsprechend des Konzepts die Umsetzung der einzelnen Prozesse erläutert. Anhand eines Beispiels wurde die Verwaltung von Diensten als Teil einer IaC-Beschreibung dargestellt. Die Schnittstellen zur vollständigen Integration in einen standardisierten Engineeringprozess wurden an vielen Stellen beschrieben und auf das nächste Kapitel verwiesen. Der Engineeringprozess ergänzt die bisher statische Beschreibung zur Orchestrierung von Diensten um einen automatisierten und kontinuierlichen Prozessablauf.

5 Engineeringprozess

Die Umsetzung eines Virtualisierungs- und Verwaltungskonzepts auf Stationsebene ermöglicht die Nutzung automatisierter Prozesse zur Bereitstellung von Diensten. Unter dem Verständnis von IaC lassen sich codebasierte Beschreibungen nutzen, um die Bereitstellung von Diensten sowie die benötigten Ressourcen schnell, flexibel, skalierbar und automatisiert durchzuführen. Um die Vorteile einer codebasierten Beschreibung vollständig auszuschöpfen, muss eine Integration in bestehende Engineeringprozesse erfolgen. Dieses Kapitel untersucht bestehende Prozessschritte in der Stationsautomatisierung und beschreibt einen kontinuierlichen Engineeringprozess zur Unterstützung einer codebasierten Bereitstellung von Diensten.

5.1 Verwandte Arbeiten

Die Modellierung eines Systems, dessen Prozessabläufe und deren Umsetzung sind komplexe Themen, die auch im Kontext eines Smart Grids Anwendung finden. Es gibt eine Vielzahl von Modellen, Engineeringprozessen und Systemarchitekturen aus unterschiedlichen Bereichen und Anwendungsgebieten. Im Folgenden erfolgt eine kurze Beschreibung des Smart Grid Architecture Model (SGAM) mit der IEC 62559 Anwendungsfallmodellierung, des V-Modell XT als Vorgehensmodell für die IT-Systementwicklung und des IEC 61850 Engineeringprozesses. Jedes dieser Verfahren umfasst eine umfangreiche Beschreibung und wird entweder als Standard gepflegt oder von vielen Organisationen verwendet. Um die Komplexität den Bedürfnissen dieser Arbeit anzupassen, wird basierend auf der Kurzbeschreibung eine Auswahl auf eines dieser Verfahren durchgeführt. Anschließend werden weitere verwandte Arbeiten in Bezug auf dieses Verfahren beschrieben.

Das SGAM [54] beschreibt einen strukturierten Ansatz zur Beschreibung und Planung von Smart Grid Lösungen, der dazu beiträgt, die Interoperabilität, Skalierbarkeit und Effizienz von intelligenten Energiesystem zu verbessern. Das SGAM bietet einen Rahmen, um die verschiedenen Technologien, Systeme und Interaktionen zu strukturieren und zu verstehen. Hierzu werden verschiedene Schichten wie „Komponenten, Kommunikation, Informationen, Funktionen, Business“ zur Einordnung

verwendet. Eine einzelne Schicht des SGAM ist eine zweidimensionale Ebene, die einerseits die Domänen (z.B. im Energiesystem) von Erzeugung bis zum Verbrauch und andererseits die hierarchischen Zonen für das Management elektrischer Prozesse von Markt bis zum Prozess berücksichtigt. Eine grafische Darstellung ist in Abbildung 5.1 zu sehen.

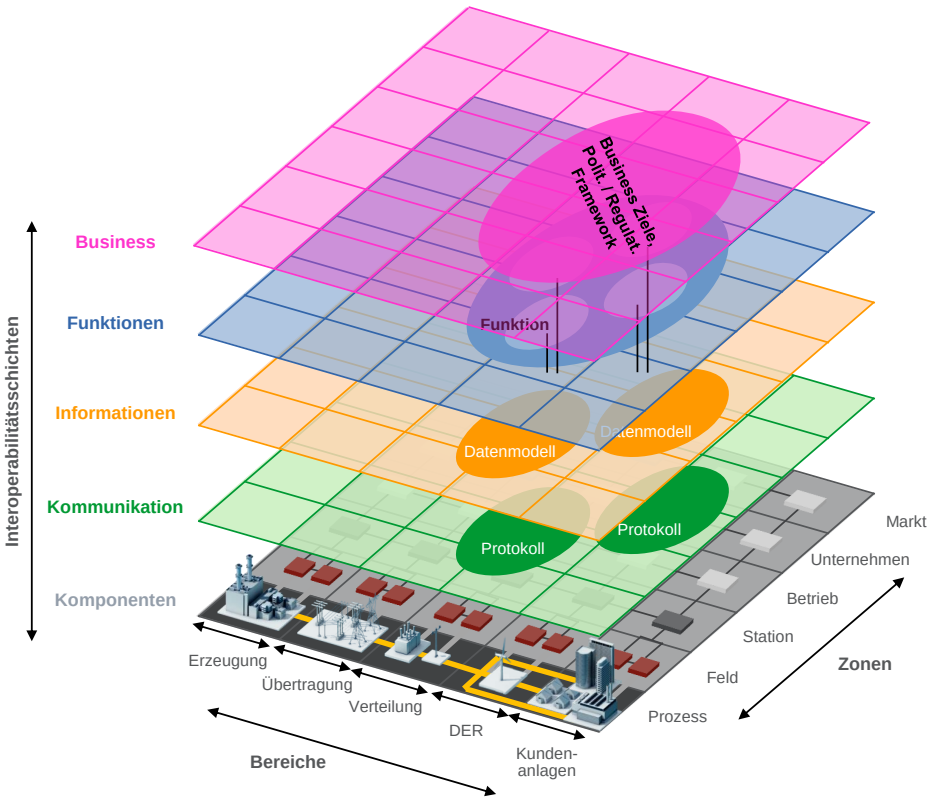

Abb. 5.1.: Das Smart Grid Architecture Model [54]

Ergänzend beinhaltet die Arbeit von [54] den Ansatz zur Beschreibung von Anwendungsfällen nach IEC 62559-2 sowie die Interoperabilität zum SGAM-Modell. Die im Standard festgelegte Anwendungsfallmethodik beinhaltet acht Beschreibungsbereiche wie die Beschreibung, eine grafische Darstellung, technische Details und weitere. Insbesondere die beiden Bereiche zur Beschreibung der Szenarien und des Informationsaustauschs bieten das Potenzial zur Umsetzung von codebasierten Beschreibungen. Ein Szenario ist dabei in mehrere Prozessschritte aufgeteilt, jeder mit Informationen zum Ereignis, Name der Aktivität und einer Beschreibung. Hinzu kommt die Beschreibung des Informationsaustauschs, welche fest definierte Begriffe wie „REPORT, GET, CREATE, CHANGE, EXECUTE“ nutzt und Sender und Empfänger als feste Akteure im System zuweist. In den Szenarien beschriebene Akteure,

Kommunikationsstandards und Informationsmodelle lassen sich anschließend in das SGAM-Modell einordnen.

Das V-Modell XT [55] als Weiterentwicklung des V-Modells wurde speziell für die IT-Projektentwicklung entworfen und findet unter anderem Anwendung in den Bereichen Verwaltung, Industrie sowie Luft- und Raumfahrt. Es zeichnet sich durch eine klar strukturierte Vorgehensweise aus. Das Grundprinzip des Vorgehensmodells ist in Abbildung 5.2 in Disziplinen dargestellt. Jede Disziplin kann einem der übergeordneten Bereiche Management (weiß), Entwicklung (dunkelblau) und Schnittstellen zu Auftraggeber und Auftragnehmer (hellblau) zugeordnet werden. Grundlegend eignet sich insbesondere der Managementbereich zur Beschreibung und Integration einer codebasierten Beschreibung, die wiederum im Entwicklungsbereich ihre Umsetzung findet. Eine pauschale Zuordnung zu einzelnen Disziplinen kann nicht erfolgen, sondern muss im Detail betrachtet werden und ist nicht Teil der aktuellen Untersuchung.

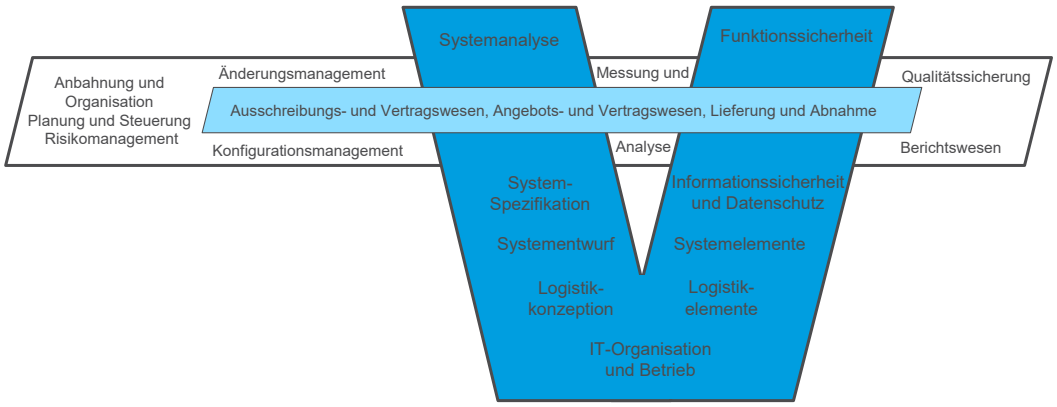

Abb. 5.2.: Überblick über die Disziplinen im V-Modell XT [55]

Der Standard IEC 61850 findet internationale Anwendung für die Kommunikation und die Datenmodellierung in der Leittechnik und Automatisierung elektrischer Energiesysteme. Der Standard ist in mehrere Teilbereiche aufgeteilt, deren grundlegende Informationen im Kapitel 2.1 beschrieben sind. Ein zentraler Aspekt ist die Datenmodellierung und der Engineeringprozess zur Beschreibung von Knoten, Objekten, Attributen und Methoden der verwendeten Geräte sowie deren Interoperabilität in einem elektrischen Energiesystem. Im Rahmen des DKE-Arbeitskreises 952.0.1 wurde eine Beschreibung des Engineeringprozesses [56] und der Anforderungen an IEC 61850 Engineeringwerkzeuge [57] erstellt. Eine Darstellung der Schritte im Engineeringprozess ist in Abbildung 5.3 abgebildet und beschreibt den Engineeringprozess für die Bereitstellung einer neuen Station. Der Engineeringprozess wird sukzessiv von der Spezifikation der Station bis zum Testen durchgeführt. Die Prozessschritte können wie folgt beschrieben werden:

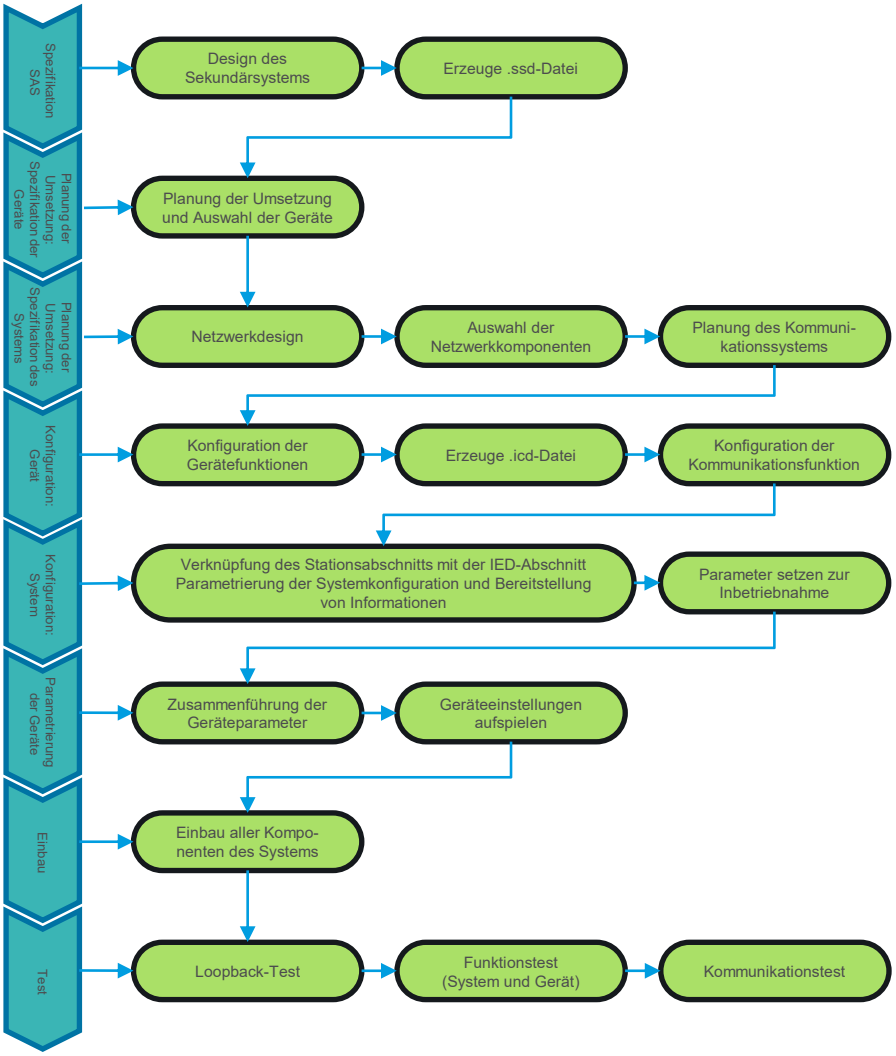

Abb. 5.3.: Engineeringprozessschritte nach [56]

Stationsspezifikation:
Im ersten Schritt erfolgt die Planung der Sekundärgeräte unter Verwendung eines Systemkonzepts, der Anforderungen an die Funktionen, dem „Single-line Diagram“ und weiteren technischen Richtlinien. Aus diesen Informationen wird nach IEC 61850 eine System Specification Description (SSD)[1]-Datei erzeugt, die die Systemspezifikationen beinhaltet.

Gerätespezifikation:
Auf Basis einer Liste von Funktionen, der SSD-Datei und den Anforderungen an die Projektspezifikationen werden die Geräte ausgewählt. Der Funktionsumfang ist entweder bereits auf den Geräten vorhanden oder wird auf diesen implementiert.

[1] In dieser Arbeit wird der englische Fachbegriff System Specification Description (SSD) verwendet.

Nach diesem Schritt liegt eine IEC 61850 IED Capability Description (ICD)[2] (Typ 1)-Datei für die weitere Verarbeitung vor.

Systemspezifikation:
Die Systemspezifikation erfolgt in drei Schritten. Zunächst erfolgt die Beschreibung eines Netzwerkdesigns anhand der ausgewählten Geräte und Funktionen. Anschließend werden dazu passende Netzwerkkomponenten ausgewählt und deren Kompatibilität untersucht. Die Systemspezifikation wird durch die Auswahl des Kommunikationsdienstes (z.B. GOOSE, MMS, etc.) und der notwendigen Kommunikationsparameter abgeschlossen.

Gerätekonfiguration:
Ziel der Gerätekonfiguration ist, dass aus den Systemspezifikationen und den Kommunikationsparametern eine gerätespezifische Konfiguration entsteht. Dazu werden im ersten Schritt aus den Anforderungen der Systemspezifikationen die funktionsbasierten Parameter eines Gerätes abgeleitet und in einem Zwischenschritt als IEC 61850 ICD (Typ 2)-Datei bereitgestellt. Anschließend erfolgt eine Beschreibung der Gerätespezifikation im Bezug auf die Kommunikation.

Systemkonfiguration:
Aus den Informationen der SSD, ICD, dem Netzwerkdesign und den Systemparametern zur Kommunikation wird eine IEC 61850 Substation Configuration Description (SCD)[3]-Datei erstellt. Ausgehend von dieser Datei und weiteren Projektspezifikationen (z.B. Netzwerkadressen und Verknüpfungen von Geräten) werden Testinformationen abgeleitet, die entweder als separate Dokumentation oder in die Systemkonfiguration eingepflegt werden.

Geräte-Parametrierung:
Auf Basis aller Informationen werden gerätespezifische Dokumente zur Parametrierung oder IEC 61850 ICD-Dateien erzeugt. Diese werden dann in die jeweiligen Geräte geladen, um diese zu konfigurieren.

Einbau:
In diesem Schritt erfolgt der Einbau und die Verknüpfung aller Geräte in der Station. Anschließend soll ein vollständig funktionierendes System bestehen.

Test:
In den letzten Schritten des Engineeringprozesses werden unterschiedliche Tests

[2] In dieser Arbeit wird der englische Fachbegriff IED Capability Description (ICD) verwendet.
[3] In dieser Arbeit wird der englische Fachbegriff Substation Configuration Description (SCD) verwendet.

durchgeführt. Dazu zählen die Überprüfung der Konnektivität durch Bit-Tests oder Loopback-Tests, Funktionstests und Kommunikationstests.

Dieser Engineeringprozess folgt dem „Top-Down“-Ansatz, ist jedoch nicht zwingend vorgegeben. Weitere Ansätze wie der „Bottom-Up“-Ansatz oder eine gemischte Form finden in aktuellen Implementierungen Anwendung. In der Arbeit [22] zu den Anforderungen an Engineeringwerkzeuge werden insgesamt fünf Varianten (A-E) unterschieden. Die Unterschiede beziehen sich primär auf die vorhandenen Informationen zu den Systemspezifikationen und den Gerätebeschreibungen. Die einzelnen Prozessschritte ähneln sich in ihrer Ausprägung, wobei keine chronologische Abfolge vorgeschrieben wird. Tabelle 5.1 zeigt die verschiedenen Engineeringvarianten und welche Spezifikationen Berücksichtigung finden.

Tab. 5.1.: Übersicht der Engineeringprozessvarianten (EP)

EP	**Systemspezifikation**		**Gerätebeschreibung**			**Systemkonfiguration**
	SSD (Station)	**SSD (IED)**	**ICD**	**IID**	**SCD**	**SCD**
A			X	X		X
B	X		X	X		X
C	X	X	X			X
D	X	X		X		X
E	X	X			X	X

Im Rahmen dieser Arbeit erfolgt an dieser Stelle eine Auswahl des Verfahrens, welches im weiteren Vorgehen genauer untersucht wird. Aufgrund der Komplexität und Möglichkeiten der einzelnen Verfahren ist eine Berücksichtigung aller nicht möglich. Die Integration und Umsetzung von codebasierten Beschreibungen in einen vollständigen Prozessablauf bei der Bereitstellung von Diensten in der Verteilnetzebene ist der Fokus. Das Vorgehensmodell V-Modell XT adressiert einen großen Bereich der Projektstrukturierung und ist damit flexibel einsetzbar. Einige der Disziplinen, wie die Systemspezifikation und der Systementwurf, könnten die Teilaufgaben, die durch die IaC-Beschreibung und die dahinterliegenden Verwaltungsaufgaben aus Kapitel 4 beschrieben wurden, in den Prozessablauf einordnen. Dennoch ist das Vorgehensmodell eine sehr abstrakte Darstellung und wurde bisher nicht auf die Anforderungen und Prozesse der Stationsautomatisierung in der Verteilnetzebene adaptiert. Notwendige Beschreibungen eines Dienstes in der Verteilnetzebene mit den darunterliegenden Funktionen, Gerätespezifikationen, Konfigurationen und Protokollen liegen außerhalb dieser Arbeit, sodass das Vorgehensmodell in der aktuellen Form nicht geeignet ist.

Das SGAM ist bereits Teil der Betrachtung von Energienetzen als Smart Grid Architekturmodell und bildet damit auch das Verteilnetz sowie die Modellierung von

Protokollen, Komponenten und deren Kommunikation ab. Im Zusammenhang mit der Beschreibung von Anwendungsfällen nach IEC 62559-2 können Prozessabläufe, Interaktionen, Datenaustausch und vieles mehr modelliert werden. Auch das SGAM stellt eine eher abstrakte, wenn auch für den Energiebereich spezifizierte Variante dar, und bestehende Engineeringprozesse zur Modellierung der Stationsgeräte, Parameter und Konfigurationen müssten an das SGAM angepasst werden oder dafür spezielle Werkzeuge bereitgestellt werden.

Das IEC 61850 ist im direkten Vergleich speziell für die Kommunikation und Datenmodellierung im Bereich der Leittechnik und Stationsautomatisierung ausgelegt. Im Vergleich zu den anderen beiden Varianten ist der Engineeringprozess im IEC 61850 weniger flexibel, insbesondere wenn nicht von dem Standard abgewichen werden soll. Dennoch wird die Möglichkeit geboten, eigene Modellierungen nach dem Schema des IEC 61850 vorzunehmen. Die XML-basierte Beschreibungsstruktur des IEC 61850 ermöglicht eine ähnliche Struktur wie die YAML Ain't Markup Language (YAML) [4]-Beschreibungssprache, die häufig für codebasierte Beschreibungen nach dem Prinzip IaC genutzt wird. Im Folgenden werden einige relevante Arbeiten im Kontext von Engineeringprozessen und dem IEC 61850 dargestellt.

In der Arbeit [20] erfolgt eine Beschreibung der Komponenten eines Applikationsmodells nach IEC 61850, wie in Abbildung 5.4 dargestellt, und erweitert diese um Elemente zur Beschreibung von Topologiedaten, Funktions- und Virtualisierungsinformationen.

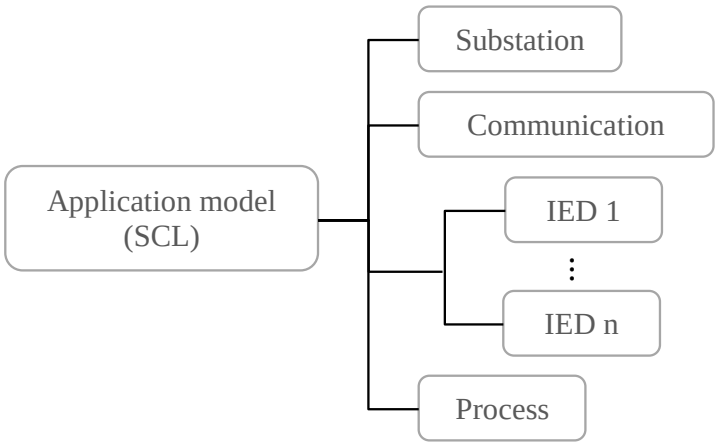

Abb. 5.4.: Aufbau eines Applikationsmodells in IEC 61850 [20]

Die hinzugefügten Elemente werden neben den standardisierten Elementen (Intelligent Electronic Device (IED), logisches Gerät, Funktionsgruppen, logische Knoten

[4] In dieser Arbeit wird der englische Fachbegriff YAML Ain't Markup Language (YAML) verwendet.

etc.) innerhalb eines IED als private Elemente dargestellt. Abbildung 5.5 zeigt die logische Struktur und die Referenzen zwischen den verschiedenen Elementen. Die Veröffentlichung beschreibt in Grundzügen den strukturellen Ansatz des Virtualisierungskonzepts (3.2.3) und des Verwaltungskonzepts (4.2). Die codebasierte Beschreibung wird aus dem IEC 61850 Applikationsmodell abgeleitet und als YAML-Datei Kubernetes zur Verfügung gestellt. Im Rahmen dieser Arbeit stellt der beschriebene Ansatz zur Abbildung von codebasierten Beschreibungen im IEC-61850-Applikationsmodell einen Prozessschritt des Engineering-Prozesses dar.

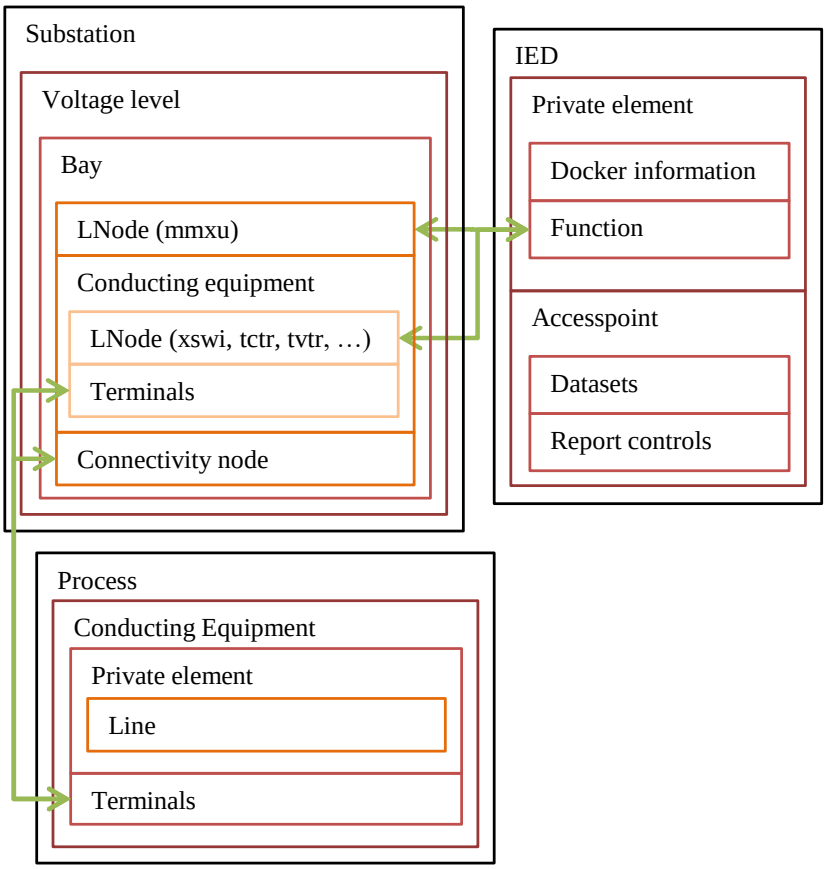

Abb. 5.5.: Anpassung des Applikationsmodells zur Integration von Virtualisierung [20]

Im Rahmen des Projekts i-Autonomous [13] wurde in einer gemeinsamen Veröffentlichung [21] ein Engineeringprozess zur Bereitstellung von virtuellen Diensten vorgestellt, der in Abbildung 5.6 dargestellt ist. Bestandteil dieses Engineeringprozesses ist die Beschreibung nach dem Standard IEC 61850, insbesondere die Verwendung des Applikationsmodells und die darauf basierende Parametrierung der Geräte. Einige der im IEC 61850 verwendeten Prozessschritte zur Erzeugung des Applikationsmodells und die Abstimmung mit der verwendeten Hardware wurden abstrahiert. Der Fokus dieser Veröffentlichung liegt in der Erweiterung bestehender

Prozessschritte und der Ergänzung um Schritte zur Bereitstellung virtueller Dienste. Der Gesamtprozess ist in drei unterschiedliche Bereiche unterteilt und beinhaltet die folgenden Prozessschritte.

Parametrierung & Konfiguration:
Ausgehend von Informationen zur Topologie und den Stationen erfolgt eine Auswahl des berücksichtigten Gebiets und der verwendeten Stationen. Anschließend wird ein Applikationsmodell gemäß IEC 61850-Vorgaben erzeugt. Im letzten Schritt erfolgt die Ergänzung um technische und operative Spezifikationen, die, wie in Abbildung 5.5 dargestellt, Informationen einer codebasierten Beschreibung und Interpretation durch Kubernetes als Orchestrator ermöglichen.

Verifizierungsprozess:
Der Verifizierungsprozess basiert auf Vorgaben, die im Rahmen des untersuchten Projektes entstanden sind und Rahmenbedingungen für eine Vorprüfung in einer Laborumgebung bieten. Im ersten Schritt erfolgt eine Anpassung der Applikationsmodelle an die Bedingungen im Labor, insbesondere die Anpassung der Kommunikation und der verwendeten Hardwaregeräte. Bereitgestellte Dienste (z.B. durch Hersteller) werden auf ihre Funktionsweise und -sicherheit erprobt. Zunächst wird nur der Dienst und die bereitgestellte Funktion in einer lokalen Instanz getestet. Anschließend erfolgt ein Integrationstest in den operativen Prozess unter Laborbedingungen.

Operativer Prozess:
Der operative Prozess beschreibt die Bereitstellung und kontinuierliche Überprüfung eines virtuellen Dienstes unter Berücksichtigung einer codebasierten Beschreibung, in diesem Fall als Teil des IEC 61850 Applikationsmodells. Im ersten Schritt erfolgt die Überführung der Parameter eines Dienstes in den Prozessablauf eines Verwaltungswerkzeugs. Abhängig von der Beschreibung umfasst der nächste Prozessschritt die Durchführung eines Verwaltungsbefehls. Die Übertragung des Applikationsmodells und der präqualifizierten Dienste erfolgt im Zusammenhang mit den ausgeführten Befehlen. Innerhalb der Gerätehardware werden die Befehle empfangen und ausgeführt. Zudem beschreibt der operative Prozess die kontinuierliche Überwachung und Analyse des Systemzustands der laufenden Dienste und der Geräte. Eine automatisierte Verarbeitung von Anpassungen bei Fehlerzuständen der Dienste wird ebenfalls berücksichtigt. Im Rahmen dieser Arbeit entspricht der operative Prozess vielen Prozessschritten, die im Verwaltungskonzept 4.2 dargestellt sind.

Die Veröffentlichung [11] beschreibt einen Teil des vorherigen Engineeringprozesses, insbesondere den operativen Prozess, und demonstriert eine technische Umsetzung. Abbildung 5.7 zeigt eine Prozesskette zur Bereitstellung eines virtuellen Dienstes.

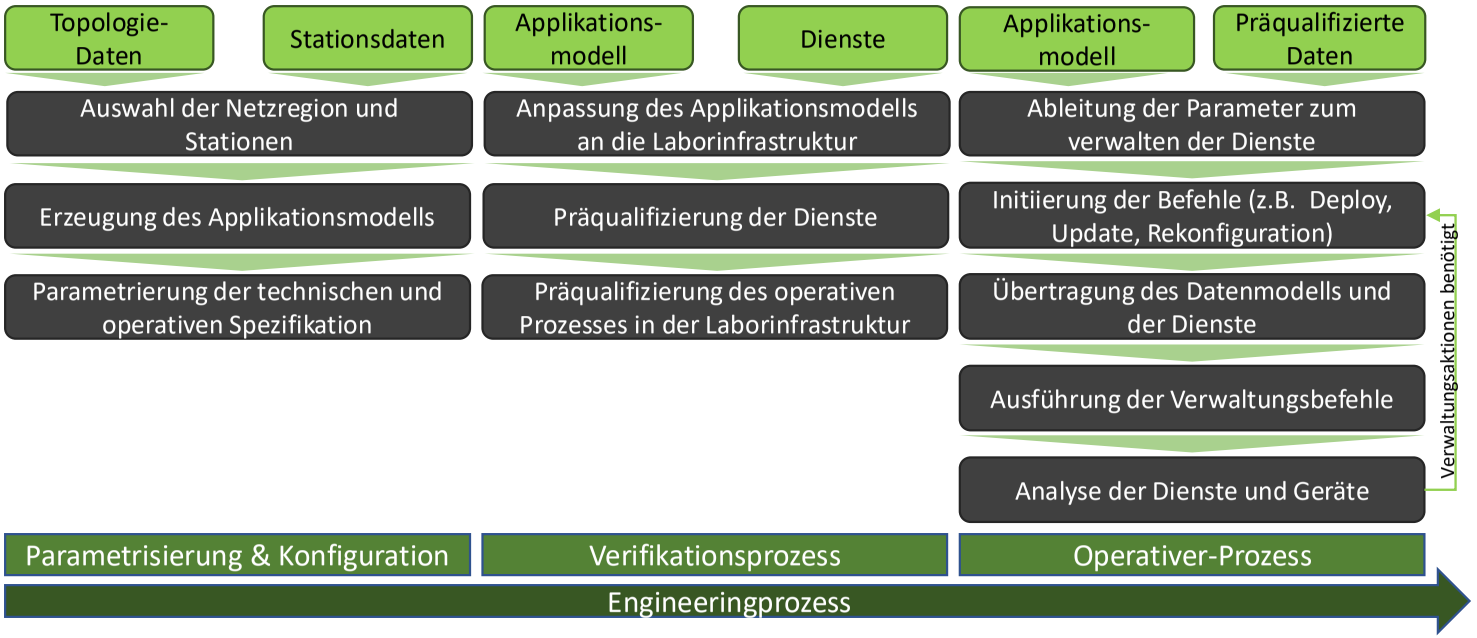

Abb. 5.6.: Engineeringprozess zur Bereitstellung virtueller Dienste [21]

Ergänzt wird der Prozess um weitere Sonderfälle; so ist es möglich, dass kein Transfer von Diensten oder Konfigurationsdateien notwendig ist, sondern lediglich Befehle ausgeführt werden. Zusätzlich wird unterschieden, ob ein Zustand der Dienste oder Geräte automatisch durch Routinen im Prozessablauf oder manuell gelöst werden muss. Im Rahmen der Veröffentlichung wird eine Testumgebung im Labor dargestellt und unterschiedliche Testfälle sowie Key Performance Indicators (KPIs) evaluiert. Abschließend werden die Ergebnisse der Testfälle im Bezug auf unterstützte funktionale und nicht-funktionale Anforderungen bewertet, darunter Skalierbarkeit, Flexibilität, Interoperabilität und weitere.

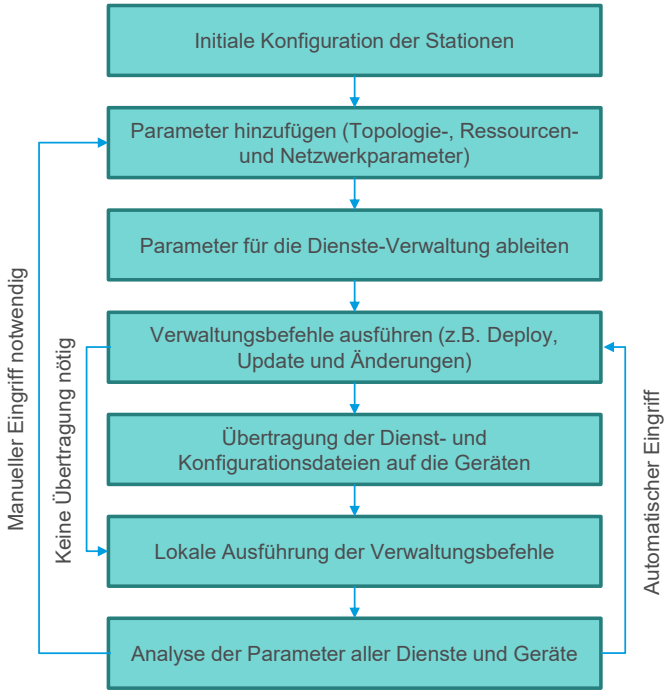

Abb. 5.7.: Prozesskette zur Bereitstellung virtueller Dienste [11]

5.2 Engineeringprozess

In diesem Kapitel wird ein vollständiger Engineeringprozess zur Bereitstellung virtualisierter Dienste in Stationen beschrieben. Ziel dieses Prozesses ist die Integration des Virtualisierungskonzepts aus Kapitel 3.2.3 und das Verwaltunkonzept aus Kapitel 4.7 unter Berücksichtigung einer codebasierten Beschreibung. Diese Beschreibung stellt die Schnittstelle zwischen dem manuellen und automatisierten Ablauf im Engineeringprozess dar. In Anlehnung an die Prozessschritte im IEC 61850 sowie die Arbeiten [9] und [22] wird der erste Teil als initialer Engineeringprozess

zur Bereitstellung der Infrastruktur innerhalb einer Station dargestellt. Der zweite Teil beschreibt den kontinuierlichen Engineeringprozess zur Integration virtueller Dienste und zur kontinuierlichen Überwachung der Geräte und Dienste.

5.2.1 Initialer Engineeringprozess

Der in Abbildung 5.8 dargestellte Engineeringprozess entspricht dem „Top-Down“-Engineeringprozess nach IEC 61850, unter Berücksichtigung einiger Änderungen, die in diesem Kapitel beschrieben werden. Jeder Block beschreibt einen der Prozessschritte (eckig) und deren Inhalte (abgerundet). Unterschieden werden Inhalte, die der Beschreibung nach [9] entsprechen (grün/ ohne Kennzeichnung), und Inhalte, die angepasst werden müssen (in türkis/ gekennzeichnet durch einen Plus). Der Datenaustausch erfolgt über die unterschiedlichen Dateiformate; die Beschreibung ist in den Grundlagen 2.1 zu finden. Im Folgenden werden die Prozessschritte und notwendigen Anpassungen beschrieben.

Systembeschreibung:
Die Systembeschreibung beinhaltet die grundlegenden Designentscheidungen einer Station. Darunter fällt auch die Bereitstellung von technischen Richtlinien, „Single-Line“-Diagrammen und Anforderungen an die Funktionen. Im Gegensatz zur bisherigen Bereitstellung von Funktionen bietet dieser Engineeringprozess die Möglichkeit, Dienste flexibel in einzelne Stationen zur Betriebszeit hinzuzufügen. Daher sollten Anforderungen entsprechend der aktuellen und zukünftigen Bedürfnisse eingeplant werden und im Hinblick auf die Geräteauswahl eher als technische Rahmenbedingungen wie die Planung von Systemressourcen beschrieben werden. Im Abschnitt 3.4 erfolgt eine Darstellung der Anforderungen an die Hardware.

Spezifikation:
Teil der Spezifikation ist die Auswahl der Geräte, der Netzwerkkomponenten sowie die Beschreibung des Netzwerkdesigns und der -parametrierung. Ergänzend zu den Funktionsanforderungen stellen die Hersteller von Geräten ICD-Dateien zur Spezifikation der unterstützten Datenmodelle, Kommunikationsdienste, logischen Knoten und weiteren Merkmalen eines IED bereit. Die Bereitstellung einer ICD-Datei widerspricht nicht dem Engineeringprozess; Funktionen können weiterhin initial auf den Geräten bereitgestellt werden. Wichtig ist die Einhaltung der Anforderungen an die Geräte-Hardware und der Abgleich mit den aus der Systembeschreibung gewünschten Informationen zu den Systemressourcen.

Konfiguration:
Die Konfiguration lässt sich in die Gerätekonfiguration und die Systemkonfiguration

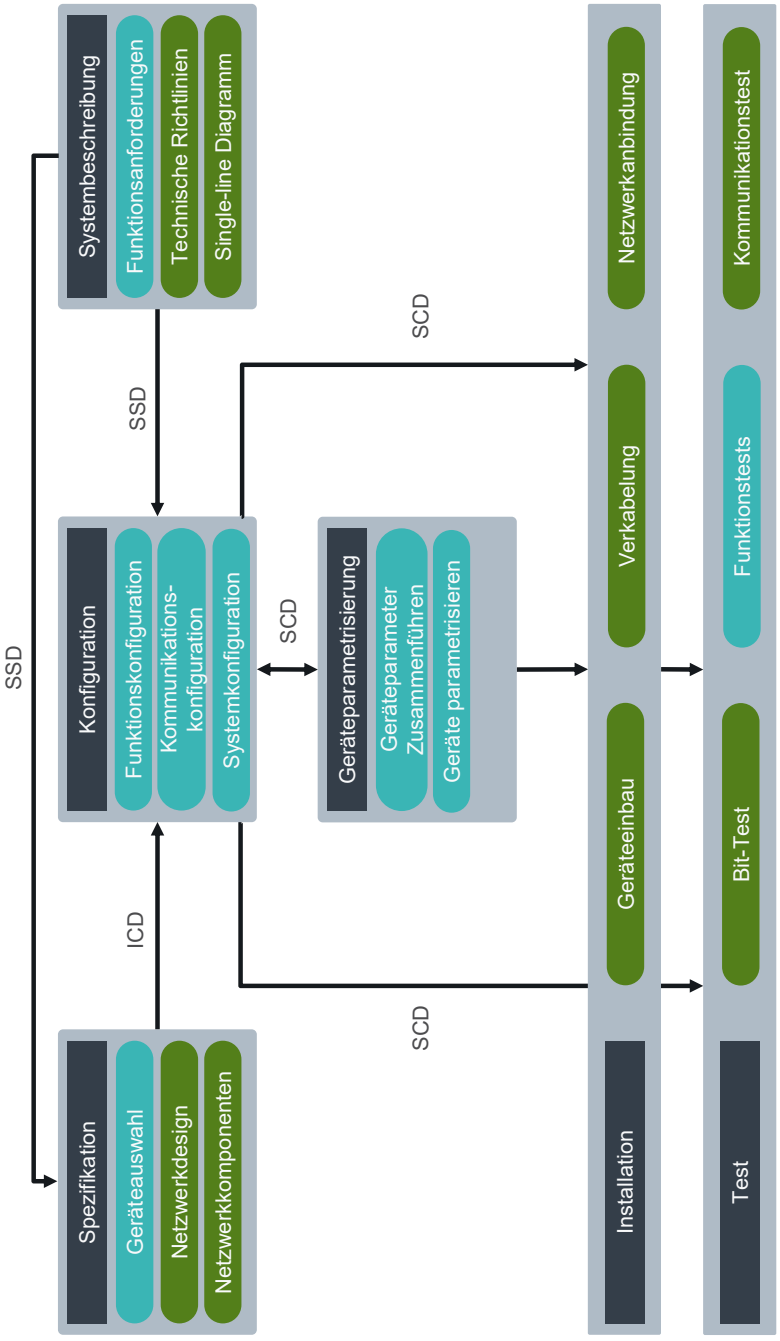

Abb. 5.8.: Initialer Engineeringprozess

unterteilen. Die Gerätekonfiguration ergänzt die ICD-Datei um notwendige Parameter aus der Systembeschreibung und der Kommunikationsbeschreibung. Dieser Teil der Konfiguration ist in diesem Engineeringprozess optional und kann ebenfalls über den kontinuierlichen Engineeringprozess nach dem Einbau der Stationsgeräte erfolgen. Die Systemkonfiguration erzeugt eine SCD-Datei, die Netzwerk- und Systemparameter sowie die ICD-Beschreibung verknüpft. Die Kommunikationsparameter müssen an dieser Stelle die Einbindung des Verwaltungskonzepts 4.7 berücksichtigen. Ob es sich dabei um eine zentrale oder dezentrale Verwaltungsschicht handelt, hängt von der Umsetzung ab und muss entsprechend im Netzwerkdesign berücksichtigt werden.

Geräte-Parametrierung:
Dieser Prozessschritt parametriert die ausgewählten Geräte unter Berücksichtigung der Änderungen in der SCD-Datei. Es ist möglich, dass initial keine Funktionen konfiguriert sind und nur die Beschreibung zur Integration in die Station Berücksichtigung findet. In diesem Fall erfolgt die Bereitstellung von Diensten über den kontinuierlichen Engineeringprozess.

Installation:
Der Geräteeinbau, die Verkabelung und die Netzwerkanbindung erfolgen wie im klassischen Engineeringprozess.

Test:
Mit Ausnahme der Funktionstests erfolgt die Verifizierung wie im klassischen Engineeringprozess. Die Funktionstests können nur für die initialen Funktionen durchgeführt werden. Dienste über den kontinuierlichen Engineeringprozess lassen sich ausschließlich über den Prozessschritt zur Überwachung der Dienste überprüfen.

5.2.2 Kontinuierlicher Engineeringprozess

Der kontinuierliche Engineeringprozess findet nach der initialen Einrichtung aller notwendigen Hardware-Geräte und Verkabelungen in einer Station statt. Dieser Prozess beschreibt die automatisierte Bereitstellung, kontinuierliche Überwachung und flexible Verwaltung von Diensten. Der kontinuierliche Engineeringprozess lässt sich in den manuellen und automatisierten Prozess unterteilen. Die Spezifizierung, Konfiguration und Prametrierung von Diensten ist Teil des manuellen Engineeringprozesses, wie in Abbildung 5.9 dargestellt. Anschließend erfolgt die automatische Verarbeitung der codebasierten Beschreibung eines Dienstes unter Berücksichtigung des Virtualisierungs- und Verwaltungkonzept aus Kapitel 3.2.3 und 4.2. Eine Darstellung des automatisierten Engineeringprozesses ist in Abbildung 5.10 zu finden.

5.2.3 Manueller Engineeringprozess

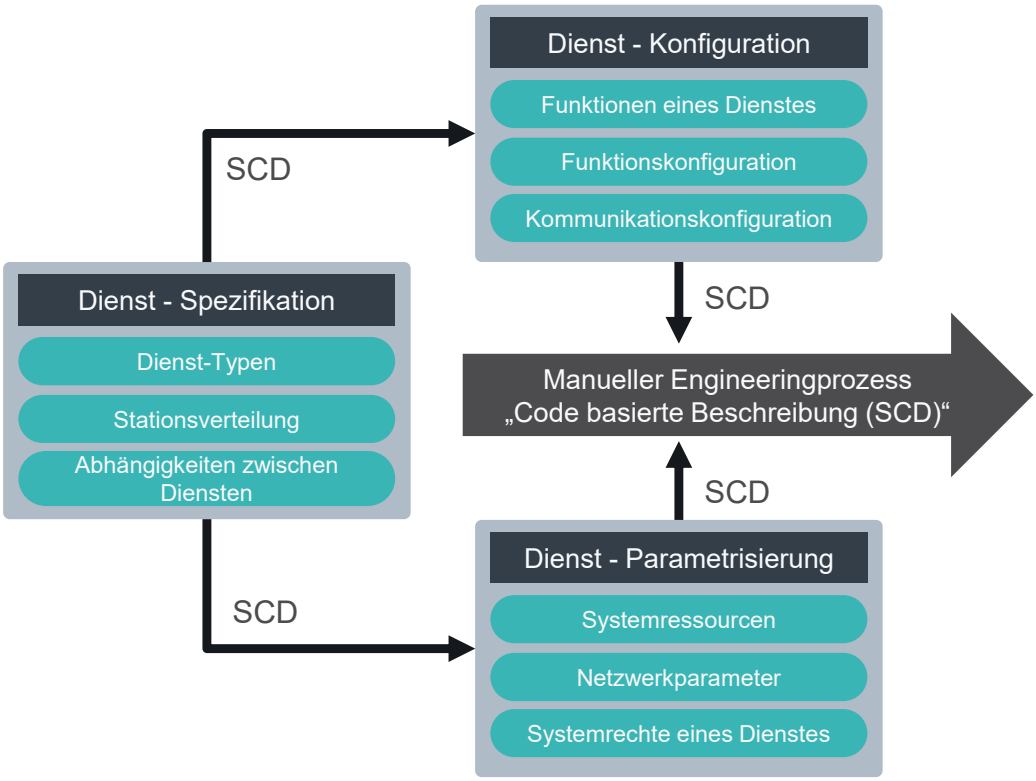

Abb. 5.9.: Manueller Engineeringprozess

Die Beschreibung eines Dienstes basiert auf einer codebasierten Beschreibung, umgesetzt innerhalb einer SCD-Datei. Berücksichtigt werden die Prozessschritte Spezifikation, Konfiguration und Parametrierung, die der Beschreibung einer Funktion im klassischen Engineeringprozess nach IEC 61850 ähneln. Alle notwendigen Informationen werden in der SCD-Datei eingetragen und zwischen den Prozessschritten weitergegeben. Nachfolgend erfolgt eine Beschreibung aller Prozessschritte.

Dienst-Spezifikation:
Im ersten Schritt erfolgt die Auswahl eines Dienst-Typs, wobei jeder Dienst eine eindeutig identifizierbare Typ-Bezeichnung besitzen muss. Zum Beispiel einen Namen mit Versionierung (Netzzustandsschätzung in Version 1). Anschließend erfolgt eine Beschreibung zur Verteilung von Diensten auf die unterschiedlichen Hardware-Geräte. Neben einer 1:1-Zuweisung sind auch 1:n- oder n:n-Zuweisungen möglich, d.h. ein Dienst kann genau einem IED oder mehreren Geräten zugewiesen oder mehrere Replikationen eines Dienstes können einem oder mehreren IED zugewiesen werden. Die Verteilung der Dienste (im weitesten Sinne auch Lastverteilung genannt) erfolgt in Abhängigkeit von der verwendeten Technologie und Konfiguration. Eben-

falls können Abhängigkeiten zwischen Diensten modelliert werden, beispielsweise ist der SE-Dienst abhängig von der Bereitstellung von Messwerten anderer Stationen.

Dienst-Konfiguration:
Im Vergleich zur Spezifikation, die eine stationsübergreifende Sicht aller Dienste im betrachteten Bereich einnimmt, bezieht sich die Konfiguration nur auf die Beschreibung eines einzelnen Dienstes. Wie bereits in der Definition eines Dienstes unter 3 beschrieben, kann ein Dienst aus mehreren Funktionen bestehen. Die Konfiguration der Funktionen und ihre Kommunikation erfolgt wie im klassischen Engineeringprozess und vervollständigt die SCD-Datei.

Dienst-Parametrierung:
Wie bei der Konfiguration erfolgt die Parametrierung für jeden einzelnen Dienst und beschreibt die Systemressourcen, Netzwerkparameter, Systemrechte usw. eines Dienstes. Systemressourcen können neben zugewiesener CPU-Leistung, RAM und Systemspeicher auch Limitierungen, dynamische Ressourcen oder Bedingungen enthalten. Die Netzwerkparameter beziehen sich primär auf die Verwaltung von Internet Protocol (IP)-Adressen, Port-Freigaben, Netzwerkschnittstellen und bereitgestellte Netzwerkbereiche eines Gerätes. Die Systemrechte sind vielfältig und beziehen sich auf Rechtegruppen des zugrunde liegenden BS. Eine Beschreibung möglicher Systemrechte eines Linux-Betriebssystems ist unter [34] zu finden.

Die Reihenfolge, in der die Beschreibungen in die SCD-Datei ergänzt werden, ist nicht vorgegeben. Es besteht lediglich eine Abhängigkeit zwischen der Spezifikation und den beiden Prozessschritten Konfiguration und Parametrierung. Die vollständige codebasierte Beschreibung liegt als XML-basierte SCD-Datei vor.

5.2.4 Automatisierter Engineeringprozess

Der automatisierte Engineeringprozess vereint die in dieser Arbeit beschriebenen Konzepte zur Virtualisierung und Verwaltung von Diensten. Der in Abbildung 5.10 dargestellte Prozess umfasst vier Prozessschritte, die kontinuierlich oder ereignisorientiert ausgelöst werden. Im Folgenden werden die Prozessschritte, Prozessübergänge und Auslöser beschrieben.

Dienst-Verwaltung:
Der Prozessschritt zur Verwaltung von Diensten knüpft an das beschriebene Verwaltungskonzept aus Kapitel 4.2 an. Die im Konzept beschriebenen Aufgaben der Verwaltungsschicht zur Überwachung, Steuerung und Routineausführung wurden in

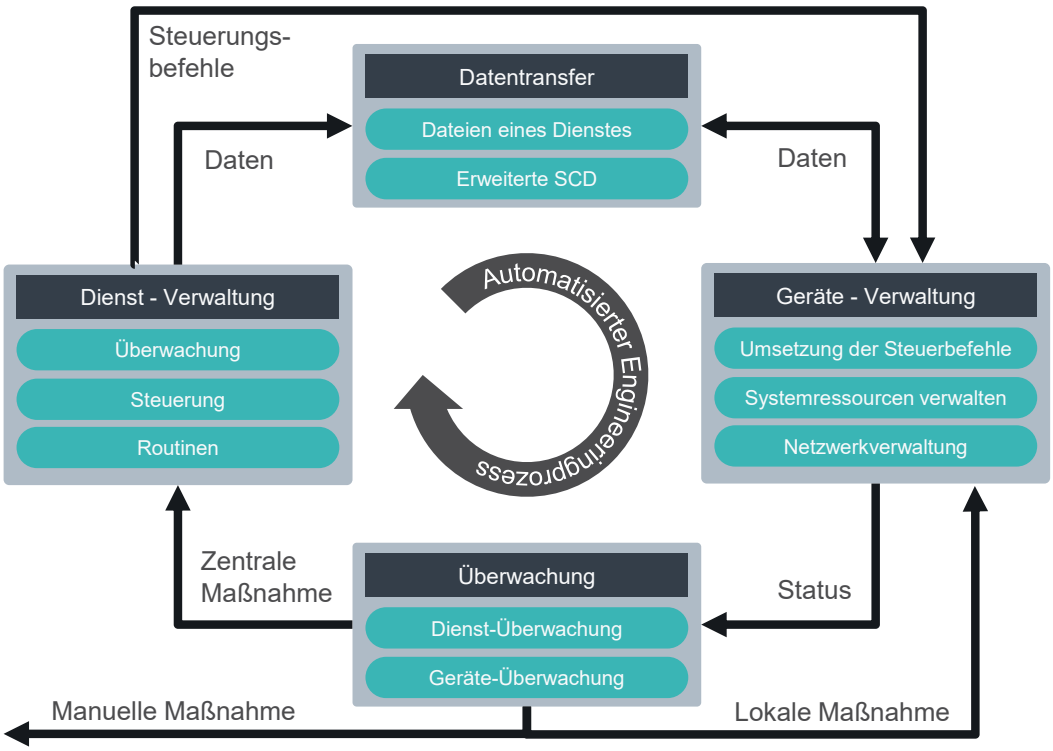

Abb. 5.10.: Automatisierter Engineeringprozess

den automatisierten Engineeringprozess übernommen. Die Dienstebeschreibung entspricht der codebasierten Beschreibung, die aus dem manuellen Engineeringprozess als SCD-Datei bereitgestellt wird. Die Nutzung der Beschreibung ist abhängig von der genutzten Technologie und muss gegebenenfalls in ein geeignetes Format überführt werden. Ausgehend von dieser Beschreibung aktiviert sich die Steuerung und führt eine Reihe von Überprüfungen durch. Im ersten Schritt wird die Beschreibung auf Änderungen zu bestehenden SCD-Dateien überprüft. Sollte die Datei nicht im Diensteverzeichnis existieren oder ein Unterschied als Update vorliegen, wird die komplette Beschreibung erneut durchgeführt. Abhängig von der bereitgestellten Konfiguration werden Steuerbefehle an die Geräteverwaltung gesendet, die Übertragung der Dienste und der SCD-Datei veranlasst, lokale Routinen eingerichtet und Anpassungen an den Grenzwerten der Überwachung durchgeführt. Die Beschreibung zum Diensteverzeichnis aus dem Konzept ist abhängig von der genutzten Technologie bereitzustellen.

Datentransfer:
Der Datentransfer wird durch die Dienstverwaltung ausgelöst und beinhaltet die Übertragung aller notwendigen Daten eines Dienstes oder die Bereitstellung der codebasierten Beschreibung als SCD-Datei an das IED-Gerät innerhalb der Station. Eine weitere Alternative ist die Abfrage eines Datentransfers ausgehend von der

Geräte-Verwaltung. Letztere Variante hat den Vorteil, dass Dateien von Diensten und Beschreibungen nur angefordert werden, wenn diese lokal nicht verfügbar sind. Beide Varianten sind im Engineeringprozess abgebildet und können abhängig von der gewählten technologischen Umsetzung variieren.

Geräte-Verwaltung:
Dieser Prozessschritt entspricht den Aufgaben der Verwaltungsschicht aus dem im Kapitel 3.2.3 beschriebenen Virtualisierungskonzept. Im Fall von neuen Steuerbefehlen aus der Dienstverwaltung werden diese durch die Verwaltungsschicht der IED umgesetzt. Die kontinuierliche Überwachung der Dienste und der Geräte kann ebenfalls lokale Maßnahmen auslösen. Wurde ein Dienst beispielsweise ungewollt beendet, kann dieser automatisiert neu gestartet werden. In dem Fall muss die Dienstverwaltung als zentrale Instanz nicht eingreifen.

Überwachung:
Die Überwachung als Prozessschritt ist Teil der beiden Verwaltungsschichten aus dem Verwaltungskonzept und dem Virtualisierungskonzept. Jedes Gerät überwacht seinen eigenen Status und stellt diese Informationen der zentralen Instanz zur Verfügung. Abhängig von der verwendeten Technologie, der codebasierten Beschreibung eines Dienstes und deren Grenzwerten werden unterschiedliche Maßnahmen ausgelöst. Lokale Maßnahmen führen zur Umsetzung von Steuerbefehlen (z.B. Neustart) innerhalb des betrachteten Stationsgerätes. Zentrale Maßnahmen lösen Routinen und Steuerbefehle auf der zentralen Dienstverwaltung aus, woraufhin neue Steuerbefehle an die Stationsgeräte gesendet werden. Ein Beispiel wäre das Starten eines Dienstes auf einem anderen Gerät. Sollten diese automatisierten Maßnahmen nicht ausreichen oder nicht umsetzbar sein, ist eine manuelle Maßnahme erforderlich, die erneut den manuellen Engineeringprozess durchlaufen muss.

Eine vollständige Darstellung des kontinuierlichen Engineeringprozesses ist in Abbildung 5.11 dargestellt. In dieser Darstellung wird die Überführung der codebasierten Beschreibung in die Dienstverwaltung ersichtlich, sowie die Rückführung über notwendige manuelle Maßnahmen.

5.3 Umsetzung des Engineeringprozesses

Die Umsetzung des Engineeringprozesses erfolgt basierend auf den Informationen zu verwandten Arbeiten und den technologischen Umsetzungen aus den Kapiteln 3.5 und 4.5. Im Folgenden wird die Umsetzung des initialen, manuellen und automatisierten Engineeringprozesses beschrieben.

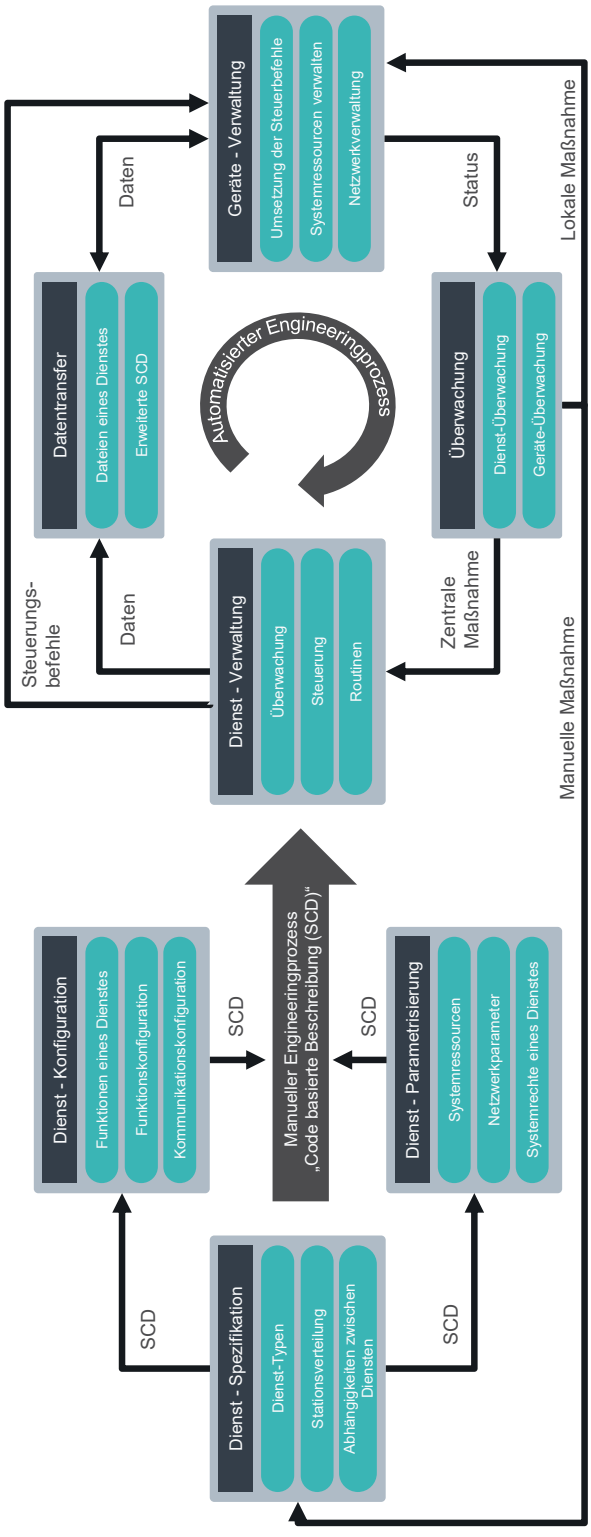

Abb. 5.11.: Kontinuierlicher Engineeringprozess

5.3.1 Umsetzung des initialen Engineeringprozesses

Die Umsetzung des initialen Engineeringprozesses kann angelehnt an den ursprünglichen Engineeringprozess nach IEC 61850 erfolgen. Dabei werden in der Regel die unterschiedlichen Dateitypen (SSD, ICD und SCD) erzeugt und ineinander überführt. Die Erstellung dieser Dateien wird üblicherweise durch Werkzeuge unterstützt. Ein Beispiel für die einzelnen Formate, Inhalte und Werkzeuge ist in Abbildung 5.12 dargestellt und entspricht den Anforderungen an IEC 61850 Engineeringwerkzeuge nach [56]. Kurz beschrieben: ICD-Dateien werden aus einer Datenbank oder durch das Gerät eines Herstellers bereitgestellt. Das Werkzeug eines Systemkonfigurators bündelt Informationen zur Spezifikation SSD zusammen mit den Informationen der Gerätespezifikation ICD. Als Ergebnis wird eine vollständige SCD-Datei bereitgestellt. Zusätzlich wird in den Anforderungen ein weiterer Prozess durch ein Engineeringwerkzeug abgebildet, dem IED-Konfigurator. Dieser übernimmt die Aufgabe, relevante Beschreibungen für ein IED aus der SCD-Datei zu extrahieren und auf das Gerät zu instanziieren. Die genauen Inhalte der SCD-Datei wurden bereits im Abschnitt 5.1 beschrieben.

Diese Engineeringwerkzeuge werden durch verschiedene Hersteller wie ABB, Phoenix Contact, H&S oder Aucotec bereitgestellt. Alternativ gibt es auch Open-Source-Softwareplattformen wie OpenSCD [58], die Werkzeuge zur Erstellung, Bearbeitung und Validierung von SCD-Dateien für die Konfiguration von IEDs anbieten. Eine Einführung zur Nutzung dieser Werkzeuge und Bereitstellung der unterschiedlichen Dateien liegt außerhalb des Rahmens dieser Arbeit. Zusätzlich gibt es ausreichend Informationen in der Community, durch Schulungen oder durch den Vertrieb der zuvor genannten Hersteller. Die verwendete SCD-Datei mit der Spezifikation der für diese Arbeit genutzten Stationsgeräte ist im Rahmen des Forschungsprojektes i-Autonomous entstanden und durch das Konfigurationswerkzeug der Firma H&S erfolgt.

Der in Abschnitt 5.8 beschriebene und in Abbildung 5.8 dargestellte optionale Prozess zur Beschreibung eines Dienstes wurde im Rahmen dieser Arbeit im kontinuierlichen Engineeringprozess umgesetzt. Das bedeutet, die Stationsgeräte wurden vollständig in die Station installiert und alle notwendigen Verkabelungen durchgeführt. Auch wurden Bit-Tests und Kommunikationstests durchgeführt. Nur alle Beschreibungen zu Funktionen oder Diensten wurden ausgelassen, sodass ein Stationsgerät keinen Dienst ausführt. Berücksichtigt wurden die Anpassungen der Stationsgeräte basierend auf den Informationen zur Umsetzung des Virtualisierungs- und Verwaltungskonzepts.

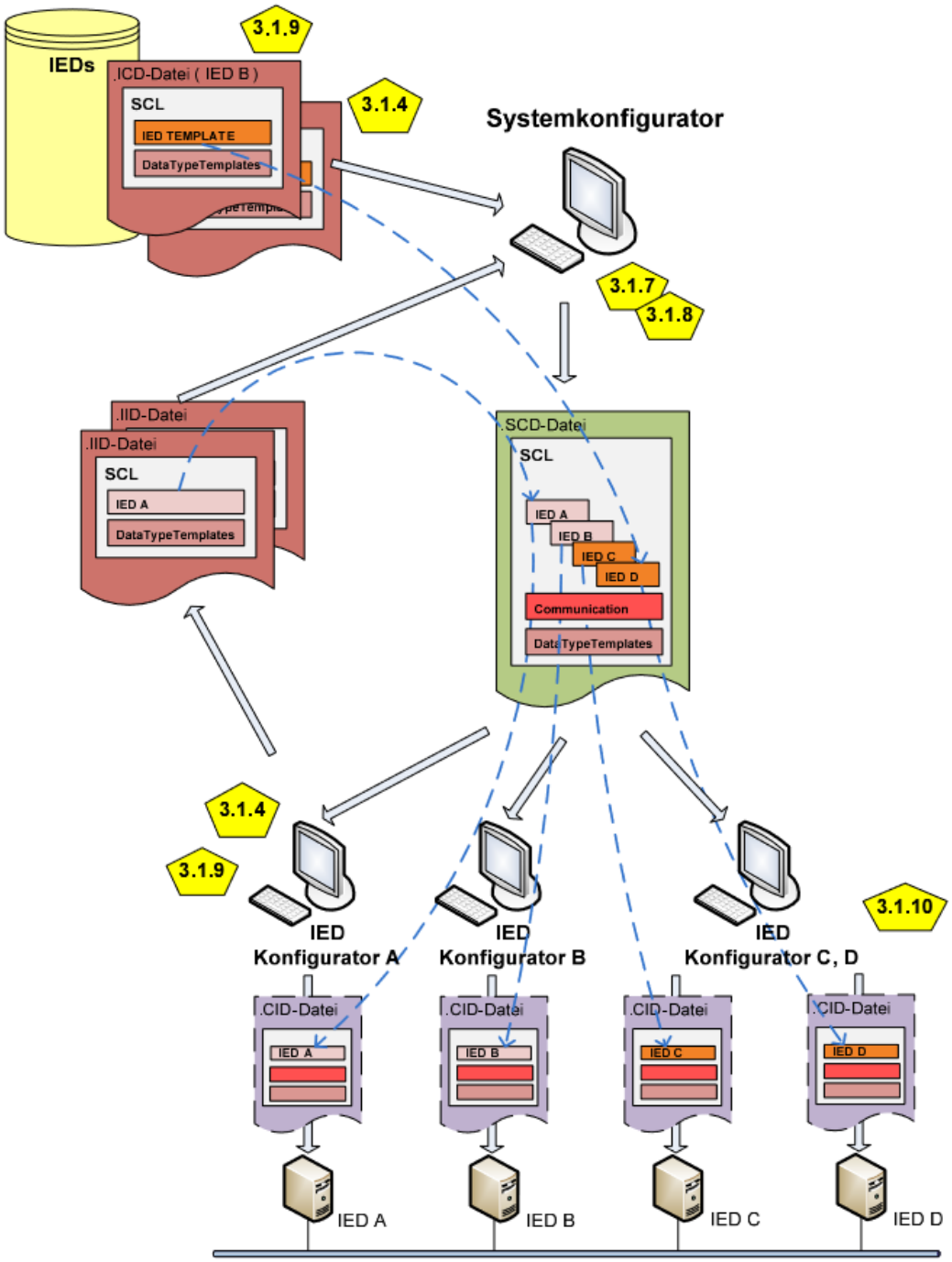

Abb. 5.12.: Übersicht zu Substation Configuration Language (SCL)-Formaten, Inhalten und Werkzeugen im Engineeringprozess nach [22]

5.3.2 Umsetzung des manuellen Engineeringprozesses

Im Gegensatz zum initialen Engineeringprozess, der primär für die Einrichtung und strukturelle Veränderungen einer Station Anwendung findet, erfolgt der manuelle Engineeringprozess bereits als kontinuierlicher Teilprozess. Dieser Teilprozess ergänzt die bestehende Beschreibung der Station um die auszuführenden Dienste. Die notwendigen Ergänzungen sind noch nicht durch die Beschreibung im IEC 61850 umgesetzt, wodurch keine Unterstützung durch bestehende Engineeringwerkzeuge bereitgestellt wird. Aus diesem Grund erfolgten sämtliche Anpassungen direkt am XML-Dokument unter Verwendung eines dafür typischen Texteditors oder durch implementierte Schnittstellenadapter. Im Folgenden werden die drei Hauptbestandteile in ihrer Umsetzung beschrieben.

Dienst-Spezifikation:
Ein Beispiel zur Spezifikation von Diensten in einer XML-Struktur ist im Skript 5.1 dargestellt. Entsprechend der Struktur aus Abbildung 5.5 bleibt die Struktur eines IED erhalten, und jeder Dienst wird wie ein IED modelliert. In Zeile 2 erfolgt eine Beschreibung und Referenzierung des gewählten IED. Die Struktur der Namensgebung hängt von der Spezifikation eines Netzbetreibers ab und ist zum Beispiel eine Kombination aus Netzbetreiberkürzel (NB), Netzgebiet (Zahl), Stationsbezeichner, Referenz zum Spannungslevel und Verortung (z.B. Kabelverteilerschrank) mit ID. Der Bereich vier bis neun beschreibt den privaten Bereich im IEC 61850. Dieser Bereich kann flexibel genutzt werden, wird jedoch nicht im Standard vorgegeben und daher auch nicht im Bezug auf Interoperabilität berücksichtigt. Der private Bereich wurde im Rahmen dieser Umsetzung genutzt, um die Beschreibung des genutzten Dienstes zu spezifizieren. Inhaltlich wird dafür eine Docker-Instanz genutzt mit dem Image-Namen und der dazugehörigen ID. Dies ermöglicht einem Netzbetreiber die Spezifikation und Auswahl eines oder mehrerer Dienst-Typen. Die Abhängigkeiten zwischen Diensten erfolgen über die Kombination mehrerer zugeordneter Kubernetes-Beschreibungen, wie in den Zeilen sechs und sieben. Die Verteilung der Dienste auf unterschiedliche Stationen ist in diesen Kubernetes-Beschreibungen enthalten, wie in Zeile 21 des Abschnitts 5.3. Aus Kubernetes-Sicht sind die Stationsgeräte jeweils einzelne Knoten, die z.B. als Worker bezeichnet werden. Die Abhängigkeiten zwischen Diensten werden in der SCL[5]-Beschreibung nicht aufgeführt. Eine Möglichkeit ist jedoch im Ausschnitt 5.2 dargestellt. Der Container „myapp“ wird erst nach Erfüllung der Kriterien des initialen Containers durchgeführt. Die Bedingungen sind in Zeile 4 dargestellt.

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

[5]In dieser Arbeit wird der englische Fachbegriff Substation Configuration Language (SCL) verwendet.

```
<IED desc="IED Beschreibung" name="
    NB_Netzgebiet_Stationsbezeichner_Spannungsebene und
    Verteilertyp_Nummerierung">
     <Private type="Station.SCL.PrivateContents.tIED">
         <Docker>
             <Image name="Image_Name" id="Docker_Image_ID"/>
             <KubeConfig1></KubeConfig1>
             <KubeConfig2></KubeConfig2>
         </Docker>
         <Functions></Functions>
     </Private>
     <Services></Services>
     <AccessPoint name="AccessPoint1"></AccessPoint>
</IED>
```

Skript 5.1: Beispiel der Dienst-Spezifikation in der SCL-Datei

```
  initContainers:
  - name: wait-for-service
    image: busybox
    command: ['sh', '-c', 'until nslookup myservice; do
        echo waiting for myservice; sleep 2; done;']
  containers:
  - name: myapp
    image: myapp:latest
```

Skript 5.2: Beispiel einer Kubernetes Konfigurationsdatei

Dienst - Parametrierung:
Die Parametrierung der Dienste wurde an die Kubernetes spezifische Beschreibungssprache, wie im Kapitel 4.5.3 beschrieben, angelehnt. Der Ausschnitt 5.3 zeigt ein Beispiel zur Umsetzung einer Kubernetes-Beschreibung in die SCD-Struktur. Im Bereich der Zeilen 3 bis 5 erfolgt die Bereitstellung von Meta-Informationen zur logischen Einordnung innerhalb von Kubernetes. Anschließend erfolgt die Spezifizierung der einzelnen Parameter eines Dienstes. Die Systemressourcen werden in den Zeilen 9 und 10 beschrieben. In diesem Beispiel wird dem Dienst „100Mi“ Arbeitsspeicher (Mi steht für Mebibytes) und „100m“ Rechenleistung (m steht für Milli-CPU und entspricht einem Tausendstel der Leistung einer CPU) fest zugewiesen, und ein oberes Limit von „200Mi“ Arbeitsspeicher und „250m“ Rechenleistung gewährt. In den Zeilen 15 bis 17 erfolgt die Zuweisung eines „Volume“, genauer eine Zuweisung des Speicherortes, über den der Dienst verfügt.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
2  <KubeConfig1>
3     <version apiVersion="v1" kind="Pod" />
4     <metadata name="SE_Pod" namespace="Namespace 1" />
5     <Spec>
6       <containers name="SeContainerInstance" image="
           Docker_Image_ID">
7         <ports containerPort="102" />
8         <resources>
9           <limits memory="200Mi" cpu="250m" />
10          <requests memory="100Mi" cpu="100m" />
11        </resources>
12        <securityContext>
13          <capabilities add="SYS_NICE, NET_BIND_SERVICE" />
14        </securityContext>
15        <VolumeMounts>
16          <mountPath path="/cache" name="cache-volume" />
17        </VolumeMounts>
18      </containers>
19      <Volumes name="cache-volume" emptyDir="{}" />
20      <imagePullSecrets name="reg-cred-secret" />
21      <nodeSelector worker="worker01" />
22      <hostNetwork value="true"/>
23    </Spec>
24  </KubeConfig1>
```

Skript 5.3: Beispiel der Dienst-Parametrierung in der SCL-Datei

Die Netzwerkparameter sind über mehrere Zeilen verteilt, angefangen mit der Zeile 7, in der die Port-Eigenschaften für den Container beschrieben werden. Ergänzt wird in Zeile 22 die Eigenschaft, dass der Container auf das Host-Netzwerk des Gerätes zugreifen darf. Dieser Zugriff ist notwendig, um die Netzwerkschnittstelle zum Gerät bereitzustellen. Der Ausschnitt 5.4 ergänzt die Eigenschaften der Pod-Beschreibung um Serviceseigenschaften, die das Netzwerk in Kubernetes näher beschreiben. Die Abhängigkeit dieser beiden Ausschnitte ist durch die Kompatibilität der Zeile sechs dieses Abschnitts mit der Zeile 4 des vorherigen Abschnitts gewährleistet. Die Zeile 7 ergänzt die Port-Spezifikation des Pod um eine Limitierung einer TCP-Verbindung über den Port 102. Die zur Verfügung gestellte IP ist in Zeile acht dargestellt. Die Systemrechte eines Dienstes sind Teil der Sicherheitsbeschreibungen und in Zeile 16 im zweiten Abschnitt dargestellt. Das Recht „SYS_NICE“ ermöglicht die Änderung der Prozesspriorität des BS und ist notwendig, damit der Dienst SE Prozesse priorisieren kann. Das andere Recht „NET_BIND_SERVICE“ ermöglicht die Portfreigabe der Ports unter 1024. Ebenfalls im Bezug auf Sicherheitseigenschaften wird in Zeile 20 ein

Image-Sicherheitscode „reg-cred-secret“ definiert und ermöglicht das Herunterladen von Images über das Repository (beschrieben im Abschnitt 4.5.2).

```
<?xml version="1.0" encoding="UTF-8"?>
<KubeConfig2>
    <version apiVersion="v1" kind="Service" />
    <metadata name="SE_Service" />
    <Spec>
      <selector app="SE_POD" />
      <ports name="port limit" protocol="TCP" port="102"
         targetPort="102" />
      <externalIPs ip="10.10.118.243" />
    </Spec>
</KubeConfig2>
```

Skript 5.4: Beispiel der Netzwerkkonfiguration in der SCL-Datei

Dienst - Konfiguration:
Die Konfiguration der Dienste umfasst primär die Beschreibung aller enthaltenen Funktionen, deren Konfiguration und Kommunikation. Dies entspricht der Basis SCD-Beschreibung des IEC 61850 und wird im Rahmen dieser Dissertation nicht detailliert untersucht. Ein Beispiel aus dem Projekt i-Autonomous [13] ist in den Abschnitten 5.5 und 5.6 aufgeführt. Der erste Abschnitt weist dem SE-Dienst einem Gerät (IED Name) zu. Die Spezifikation der Funktion erfolgt im zweiten Abschnitt, in dem die Struktur entsprechend dem Aufbau (logische Knoten, Typen, Datenklassen etc.) dargestellt ist.

```
<?xml version="1.0" encoding="UTF-8"?>
<Functions>
  <Function name="StateEstimation1" type="StateEstimation">
    <LN iedName="IED Name" ldInst="CTRL_K1KVS" lnClass="GAPC"
        lnInst="1" />
    <StateEstimationConfiguration path="StateEstimation/3-
       Knoten" />
  </Function>
</Functions>
```

Skript 5.5: Beispiel der Dienstbeschreibung in der SCL-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<LDevice inst="CTRL_K1KVS">
  <LN0 lnType="iA_LN0" lnClass="LLN0" inst="" />
  <LN lnType="iA_LPHD" lnClass="LPHD" inst="1" />
```

```
5     <LN desc="Generic Automatic Process Control" lnType="
          iA_GAPC" lnClass="GAPC" inst="1" />
6   </LDevice>
7   <LDevice inst="EstimatedStates">
8     <LN0 lnType="iA_LN0" lnClass="LLN0" inst="">
9       <DataSet name="DataSet1">
10        <FCDA ldInst="EstimatedStates" prefix="ST00074" lnClass
              ="MMXU" lnInst="1" doName="PhV.phsA" fc="MX" />
11        <FCDA ldInst="EstimatedStates" prefix="ST00074" lnClass
              ="MMXU" lnInst="1" doName="PhV.phsB" fc="MX" />
12        <FCDA ldInst="EstimatedStates" prefix="ST00074" lnClass
              ="MMXU" lnInst="1" doName="PhV.phsC" fc="MX" />
13      </DataSet>
14      <ReportControl></ReportControl>
15    </LN0>
16  </LDevice>
```

Skript 5.6: Beispiel der Struktur eines Dienstes in der SCL-Datei

5.3.3 Umsetzung des automatisierten Engineeringprozess

Die Umsetzung des automatisierten Engineeringprozesses erfolgt im Rahmen dieser Arbeit primär durch das Zusammenspiel der vorherigen Artefakte des Verwaltungs- und Virtualisierungskonzepts. Die Umsetzung der beiden Konzepte durch die Interaktion zwischen Docker, Kubernetes und anderen Hilfstechnologien ist in den Kapiteln 3.5 und 4.5 dargestellt. Um den Engineeringprozess vollständig zu realisieren, erfolgt nachfolgend die Umwandlung des SCL-spezifizierten Informationen zur Verwaltung von Diensten in ein von Kubernetes lesbares Format. Außerdem werden die in dieser Arbeit umgesetzten Übergänge und Zuordnungen der vier Prozessschritte aus Abbildung 5.10 durch Sequenzdiagramme dargestellt.

Extraktion aus dem SCL-Format:
Das für die Energiedomäne spezifizierte SCL-Format basiert auf einer spezifischen XML-Struktur und passt nicht zu der Kubernetes eigenen Spezifizierung im YAML-Format. Als Übergabe aus dem manuellen Engineeringprozess wird eine erweiterte SCD-Datei bereitgestellt. Entsprechend der Struktur aus Abbildung 5.12 bedarf die Übergabe an die Stationsgeräte nur der Beschreibung einer ICD-Datei. Aus diesem Grund erfolgt im ersten Schritt eine Aufteilung in die einzelnen ICD-Dateien. Im Rahmen dieser Arbeit wurde ein Engineering-Werkzeug der Firma H&S verwendet. Alternativ könnten Werkzeuge anderer Hersteller Anwendung finden.

Der zweite Schritt weicht vom Standard ab und wird derzeit durch kein Engineering-Werkzeug unterstützt. Daher wurde ein Konverter als Python-Programm implementiert, sodass die Kubernetes-Konfiguration aus der ICD-Datei in eine von Kubernetes unterstützte YAML-Datei überführt wurde. Die Basis des verwendeten Python-Programms ist im Anhang A.1 angefügt. Eine Repräsentation der Kubernetes-Konfiguration aus den oben aufgeführten Beispielen ist in der Liste 5.7 dargestellt.

```
  apiVersion: v1
kind: Pod
metadata:
  name: SE_Pod
  namespace: Namespace1
spec:
  containers:
  - name: SeContainerInstance
    image: Docker_Image_ID
    ports:
    - containerPort: 104
    resources:
      limits:
        memory: "200Mi"
        cpu: "250m"
      requests:
        memory: "100Mi"
        cpu: "100m"
    securityContext:
      capabilities:
        add: ["SYS_NICE", "NET_BIND_SERVICE"]
    volumeMounts:
    - mountPath: /cach
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
  imagePullSecrets:
  - name: regrcred
  nodeSelector:
    worker: worker01
  hostNetwork: true

```

```
---
apiVersion: v1
kind: Service
metadata:
  name: SE_Service
spec:
  selector:
    app: SE_POD
  ports:
  - name: port limit
    protocol: TCP
    port: 104
    targetPort: 104
  externalIPs:
  - 10.10.118.243
```

Skript 5.7: Beispiel einer Kubernetes-Konfigurationsdatei

Jeder der angewendeten Schritte ist in dem folgenden Sequenzdiagramm in Abbildung 5.13 abgebildet.

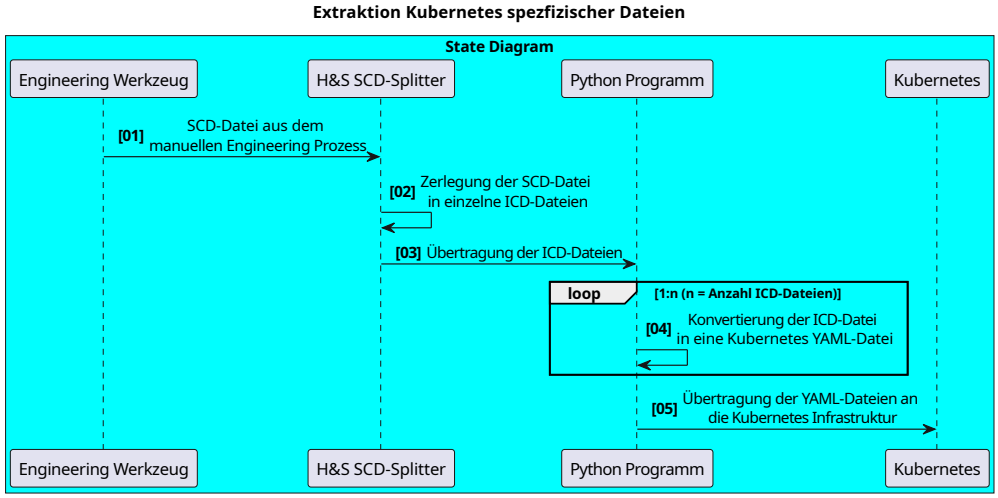

Abb. 5.13.: Sequenzdiagramm zur Darstellung der Konvertierung von SCD-Dateien in YAML-Konfigurationen

Verteilung von Diensten:
Die Verteilung von Diensten ist im Sequenzdiagramm in Abbildung 5.14 dargestellt und umfasst vier verschiedene Teilnehmer. Diese Teilnehmer sind den vier Prozessschritten zugeordnet. Die Dienst-Verwaltung entspricht der Kubernetes-Instanz, der

Datentransfer wird durch die Registry (Beschreibung siehe Kapitel 4.5.2) dargestellt, die Geräte-Verwaltung erfolgt innerhalb des Stationsgeräts, und die Überwachung bleibt entsprechend der Namensgebung gleich. Es ist zu beachten, dass eine Serverinstanz alle Bereiche mit Ausnahme der Geräte-Verwaltung umfassen kann, sodass die Namensgebung je nach Technologie oder Funktionsweise angepasst werden kann.

Das erste Sequenzdiagramm beginnt mit der Verarbeitung der YAML-Konfiguration. Basierend auf dieser Konfiguration werden die notwendigen Steuerbefehle ermittelt und festgelegt. Dazu gehören beispielsweise die Lastverteilung oder Anforderungsüberprüfung, die basierend auf der Konfiguration die für den Dienst geeignetsten Stationen auswählen (falls nicht fest vorgegeben). Die Übertragungen drei und vier im Diagramm können parallel ausgeführt werden und leiten die Steuerbefehle an die Stationsgeräte und die Konfiguration an die Registry weiter. Die Registry hinterlegt die ICD-Datei dem dazugehörigen Dienst als Image. Gleichzeitig verarbeitet das Stationsgerät die Steuerbefehle und fordert, falls nicht vorhanden, den Dienst als Image bei der Registry an. Diese stellt das Image dem Stationsgerät zur Verfügung. Weitere interne Verarbeitungsprozesse im Stationsgerät fordern Systemressourcen und Netzwerkkonfigurationen an, die es ermöglichen, den Dienst in einem Container auszuführen. Der Status wird der Überwachungssoftware mitgeteilt und der Status von Kubernetes intern verarbeitet.

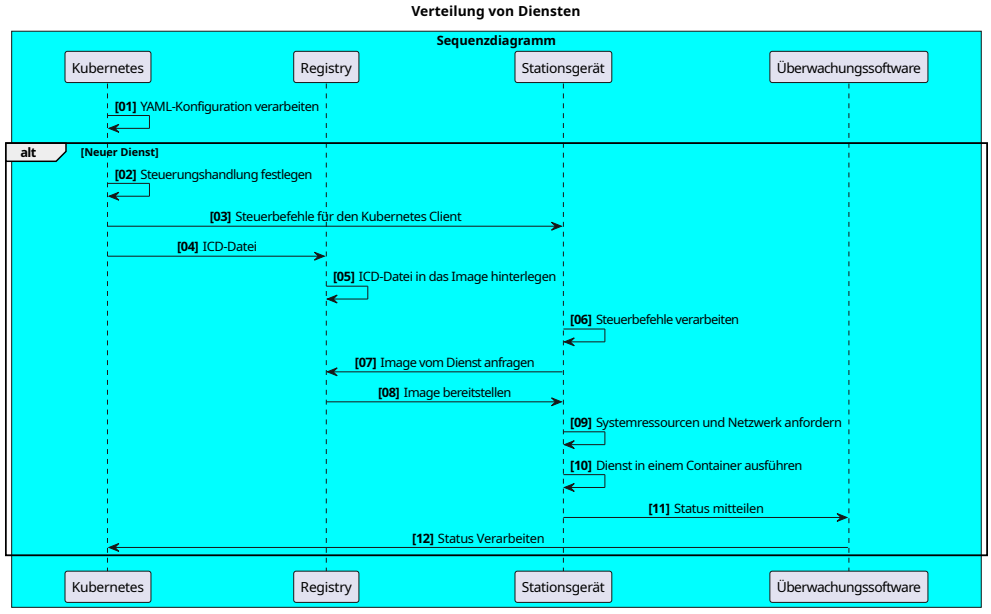

Abb. 5.14.: Sequenzdiagramm zur Darstellung der Verteilprozesse eines Dienstes

Überwachung des Systemzustands und der Systemressourcen:
Ergänzt wird die Verteilung von Diensten durch eine kontinuierliche Überwachung

der Systemzustände und Systemressourcen jedes Stationsgerätes. Im Sequenzdiagramm in Abbildung 5.15 erfolgt die Darstellung dieser Überwachung und der möglichen Reaktionen auf Abweichungen vom gewünschten Zustand. Die Überwachungssoftware führt kontinuierlich Abfragen an die Stationsgeräte durch, die ihrerseits den Systemzustand und die Systemressourcen zurückliefern. Werden Abweichungen zum gewünschten Systemzustand ermittelt, lassen sich drei unterschiedliche Aktionen ausführen. Im ersten Schritt versucht das Stationsgerät lokale Maßnahmen festzulegen. Sind diese erfolglos und erreichen zeitliche oder iterative Begrenzungen, erfolgt die Übermittlung an die zentrale Kubernetes-Instanz. Diese ist in der Lage, nicht nur Lösungspotentiale des einzelnen Stationsgeräts zu betrachten, sondern aller im Cluster eingebundenen Stationsgeräte. Zum Beispiel könnten Dienste neu verteilt oder auf anderen Geräten neu gestartet werden. Diese beiden Maßnahmen werden automatisiert anhand der zulässigen Parameter der YAML-Konfiguration ausgeführt. Sind diese nicht erfolgreich, muss ein manueller Eingriff erfolgen. Manuelle Eingriffe erfolgen ausschließlich über den manuellen Engineeringprozess, der durch einen Akteur wie den Operator durchgeführt wird.

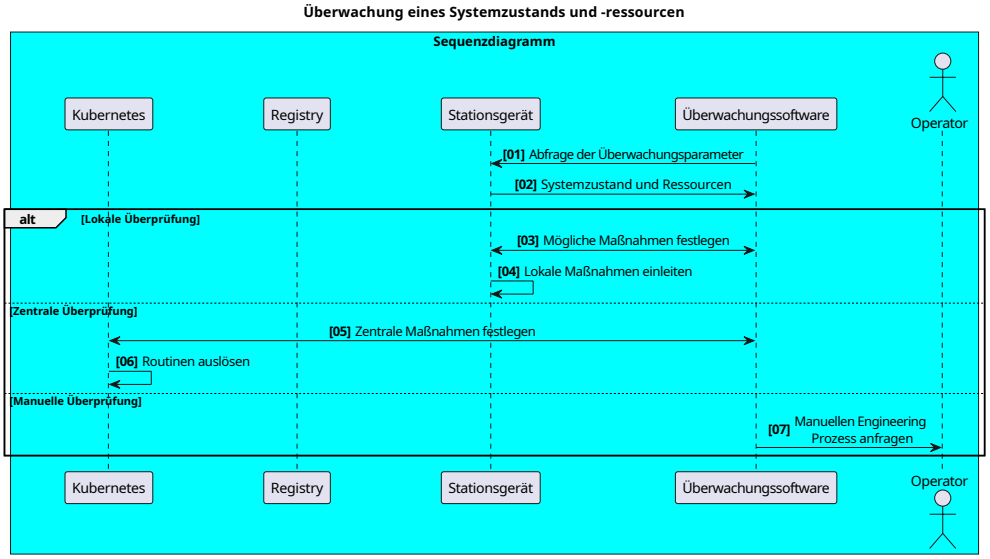

Abb. 5.15.: Sequenzdiagramm zur Darstellung der Überwachungsprozesse des Systemzustands und der Systemressourcen

Bereitstellung und Ausführung von Routinen:
Wie im vorherigen Abschnitt beschrieben, können automatisierte Prozesse als Reaktion auf Ereignisse ausgelöst werden. Diese Prozesse werden als Routinen aus den Parametern der Konfiguration entnommen. Im Sequenzdiagramm in Abbildung 5.16 erfolgt eine Darstellung zur Ausführung der Routinen anhand einiger Beispiele. Zunächst wird jede eingehende Konfiguration auf Routinen untersucht und diese

innerhalb von Kubernetes gespeichert. Tritt ein Event durch die Bereitstellung der Zustandsinformationen der Überwachungssoftware ein, erfolgt ein Abgleich mit den Routinen. In den Schritten fünf und sieben des Sequenzdiagramms werden drei unterschiedliche Maßnahmen beschrieben. In der ersten ist die Dienstmenge unterschritten, d.h. die Anzahl der Replikationen eines Dienstes wird nicht entsprechend ihrer Vorgaben ausgeführt. Zum Ausgleich leitet Kubernetes die Verteilung neuer Dienste ein. Die notwendigen Schritte sind im Sequenzdiagramm zur Verteilung von Diensten abgebildet. Die zweite Maßnahme basiert auf dem Ausfall eines Dienstes und löst den Steuerbefehl zum Neustarten eines Dienstes auf dem Stationsgerät aus. Ist allerdings eine Obergrenze der Anzahl erlaubter Neustarts überschritten, gibt es nur die Möglichkeit, den Dienst auf einem neuen Stationsgerät auszuführen oder eine Fehlermeldung zur manuellen Bearbeitung auszulösen.

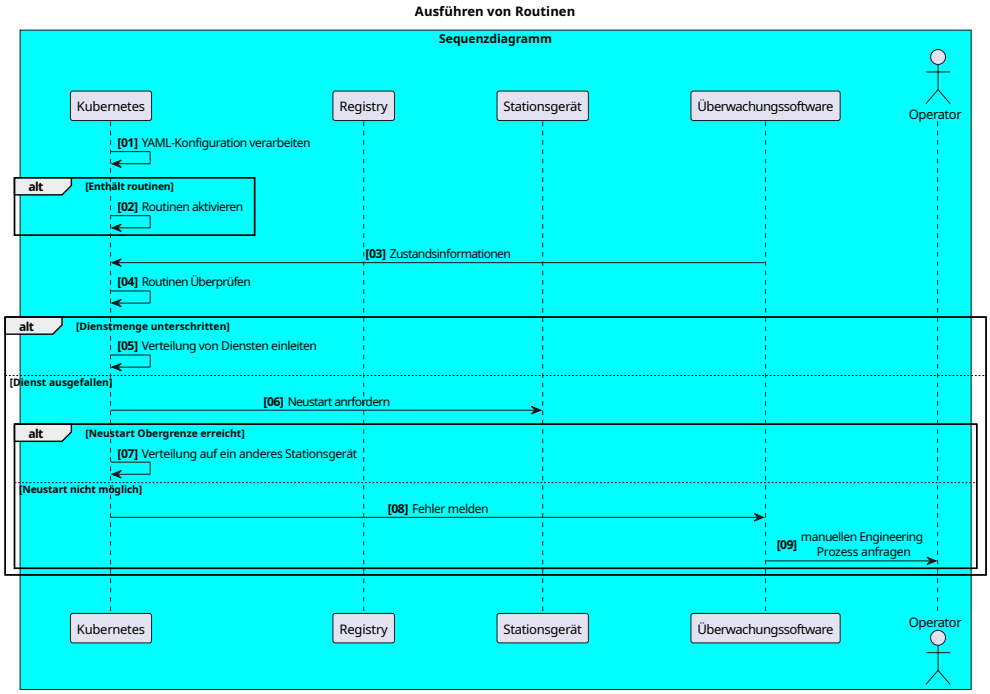

Abb. 5.16.: Sequenzdiagramm zur Darstellung der Abläufe von Routinen

Die dargestellten Sequenzdiagramme und die damit aufgeführten Beispiele wurden teilweise im Rahmen dieser Arbeit umgesetzt und für verschiedene Szenarien der Evaluation im Kapitel 6 verwendet. Die Funktionalität des Engineeringprozess ist stark abhängig von den genutzten Technologien.

5.4 Zusammenfassung

Dieses Kapitel hat sich mit der Interoperabilität des Virtualisierungs- und Verwaltungskonzepts in standardisierte Engineeringprozesse auseinandergesetzt. Dabei wurden das SGAM-Architekturmodell, das V-Modell XT und das IEC 61850 Engineering auf ihre Eignung untersucht. Eine Erweiterung des Top-Down-Ansatzes im IEC 61850 erfolgte durch die Aufspaltung initial statischer Prozesse und kontinuierlicher Prozesse. Ein wichtiger Bestandteil bei der Nutzung kontinuierlicher Prozesse ist die Umsetzung codebasierter Beschreibungen im Prozess zur Beschreibung von Diensten, angelehnt an die SCD-Beschreibung. Abschließend wurde in diesem Kapitel die Umsetzung des Engineeringprozesses unter Hinzunahme der Prozessschritte aus den vorherigen Kapiteln umgesetzt, sodass eine vollständige Verteilung von Diensten in CPES erreicht wurde. Im nächsten Kapitel wird diese Umsetzung evaluiert und die funktionalen sowie nicht-funktionalen Anforderungen überprüft.

6 Evaluation

Dieses Kapitel präsentiert die Evaluationsergebnisse im Kontext des Virtualisierungskonzeptes, Verwaltungskonzeptes und Engineeringprozesses. Die Implementierung der einzelnen Artefakte wurde bereits in den entsprechenden Kapiteln technologisch und prozessual beschrieben. Hier werden sie in zwei unterschiedlichen Umgebungen verifiziert. Die erste Umgebung entspricht einem Laboraufbau mit „Hardware-in-the-Loop“-Ansatz. Die zweite Umgebung basiert auf einer Feldtest-Erprobung in einem Verteilnetz. Verschiedene Szenarien evaluieren die funktionalen und nicht-funktionalen Anforderungen der Artefakte und tragen zur Überprüfung der Forschungsfragen bei.

6.1 Aufbau und Evaluation der Laborumgebung

Der Aufbau der Laborumgebung ist in Abbildung 6.1 dargestellt und lässt sich in zwei Bereiche unterteilen: die zentrale Instanz und die Repräsentation verschiedener Stationsgeräte in Ortsnetzstationen oder Umspannwerken. Die zentrale Instanz bündelt verschiedene Technologien auf einer redundanten Serverstruktur. Entsprechend der Umsetzung aus Abschnitt 4.5 wurden auf den Servern Docker als Laufzeitumgebung, Kubernetes als Verwaltungs- und Orchestrierungstool sowie eine Registry als Image-Repository implementiert. Im Rahmen der Laborexperimente wurde eine Kubernetes-Oberfläche genutzt, um einzelne Metriken wie den Systemstatus und die Systemressourcen anzeigen zu können. Ebenfalls wurden die notwendigen Engineering-Werkzeuge und Konvertierungstools aus Abschnitt 5.3 auf der Serverinfrastruktur bereitgestellt.

Die Repräsentation der IED innerhalb der Ortsnetzstationen erfolgte durch RPIs und industrielle Hardware entsprechend der Spezifikation in Abschnitt 3.5.1. Die Einbindung dieser Hardware in den Laboraufbau erfolgte nach dem „Hardware-in-the-Loop“ Ansatz, wodurch die industrielle Hardware die Aufgabe eines Stationsgerätes innerhalb der Laborsimulationen übernimmt. Jedes Stationsgerät ist entsprechend dem Virtualisierungs- und Verwaltungskonzept mit einer Kubernetes- und Docker-Instanz ausgestattet.

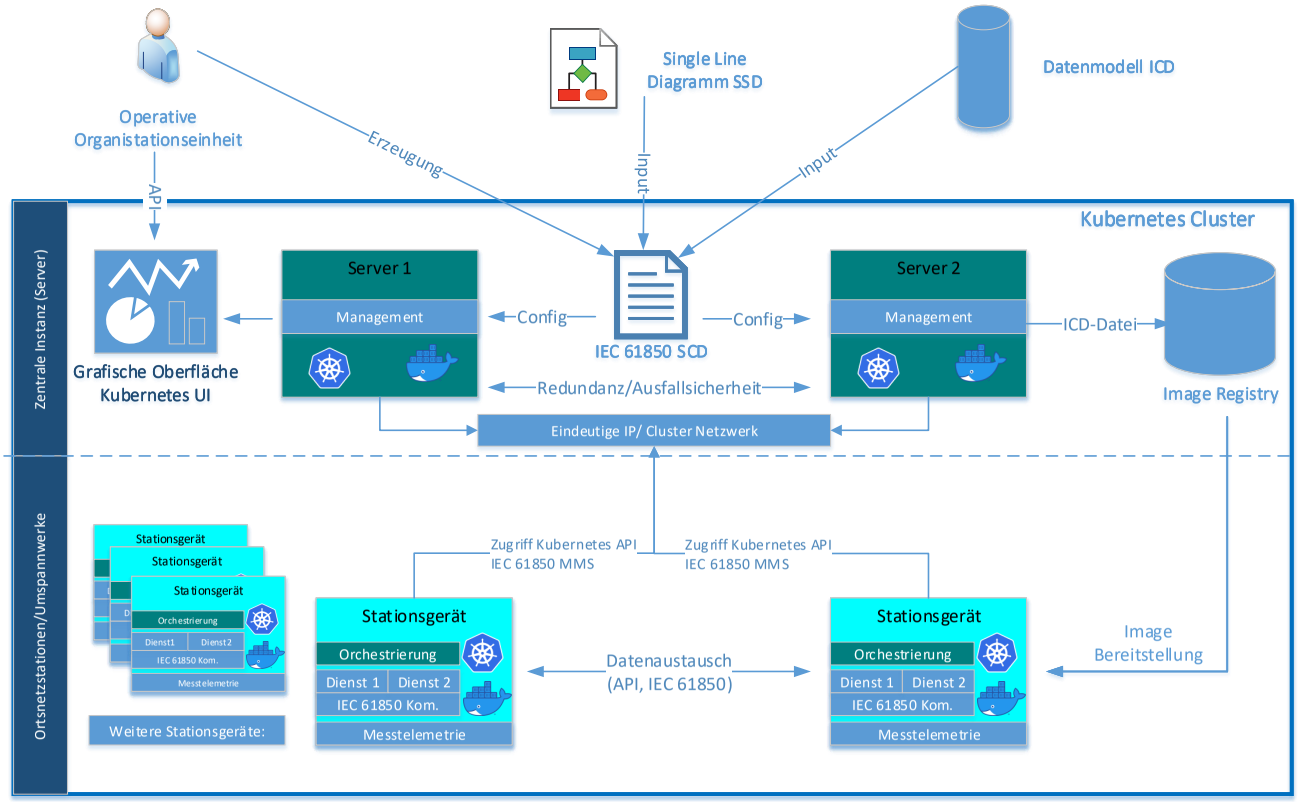

Abb. 6.1.: Struktureller Aufbau der Laborumgebung

Zusätzlich beschreibt die Laborumgebung die Input-Daten und Schnittstellen zur Erzeugung der SCD-Datei. Die Prozesse zur Erzeugung einer SCD-Datei sind bereits ausführlich in Abschnitt 5.3 dargestellt. Für die weiteren Experimente wird eine vollständig konfigurierte SCD-Datei als gegeben angesehen.

Im folgenden Unterkapitel erfolgt eine Beschreibung der genutzten Dienste im Rahmen der Laborexperimente.

6.1.1 Beschreibung der Dienste

Im Rahmen der Untersuchung wurden zwei Dienste mit geeigneten Schnittstellen und Datenaustausch zueinander verwendet. Darunter eine Netzzustandsschätzung, die auf Grundlage von synthetischen Daten eine statische Berechnung für ein kleines Verteilnetz durchführte und diese über IEC 61850 MMS Kommunikation an einen Datenempfänger sendete. Im Folgenden wird die Funktionsweise der beiden Dienste genauer erläutert.

Netzzustandsschätzung (SE) mit synthetischen Daten:
Ein wichtiger Bestandteil der Betriebsführung zukünftiger Verteilnetze ist die Fähigkeit, alle Netzzustände im Netz zu ermitteln. Die Netzzustandsschätzung findet vor allem in Übertragungsnetzen Anwendung, während in Verteilnetzen notwendige Messwerterfassungen der Knoten- und Strangströme bzw. Leistungen oft fehlen. Die Netzzustandsschätzung löst komplexe Netzgleichungen, um den Netzzustand aller im Netz befindlichen Knotenspannungen zu ermitteln. Im Gegensatz zur einfachen Lastflussberechnung werden bei der Netzzustandsschätzung zusätzliche Verfahren zur Verringerung von Messfehlern angewendet. Dies führt zu einer höheren Genauigkeit der berechneten Messergebnisse [29, 7]. Ein weit verbreitetes Verfahren ist das Verfahren der gewichteten kleinsten Quadrate (engl. Weighted Least Squares, WLS) und findet Anwendung in den SE-Algorithmen der Technischen Universität Dortmund, die im Rahmen der Projekte i-Automate [12] und i-Autonomous [13] zur Verfügung gestellt wurden und Grundlage dieser Evaluation sind. Anforderungen zur Nutzung dieses Algorithmus im Kontext der Verteilnetze ist die Bereitstellung dezentral erfasster Messwerte. In [29] wird angegeben, dass eine ausreichende Zeitsynchronisierung notwendig ist, damit die Asynchronität der Messwerte die Genauigkeit nicht zu stark beeinflusst.

Im Rahmen der Laborevaluation werden keine dezentral gemessenen Messwerte genutzt. Der Algorithmus ist mit synthetischen Daten konfiguriert und kann daher unabhängig ausgeführt werden. Die Ergebnisse der Netzzustandsschätzung werden über eine standardisierte IEC 61850 MMS Kommunikation übertragen.

Datenempfänger:
Der Datenempfänger übernimmt die Datenerfassung eines Leitsystems und basiert auf einer Software, die IEC 61850 MMS-Daten empfangen und als einfachen Print in die Kommandozeile ausgeben kann. Dieser Dienst ist ebenfalls Teil der Projektarbeit in i-Autonomous [13] und wurde von der Technischen Universität Dortmund bereitgestellt.

Weitere Dienste
Im Rahmen einiger Szenarien wurden Dienste genutzt, um die grundlegende Funktionalität der Virtualisierungs- und Verwaltungsaufgaben zu testen. Diese Dienste haben keine Relevanz im Bezug auf die Betrachtung von Smart Grids, eigenen sich jedoch gut um die Eigenschaften einer Container-Instanz zu prüfen. Im Folgenden werden diese Dienste kurz beschrieben.

Leerlauf-Dienst:
Dieser Dienst führt keinen Algorithmus aus und benötigt nur die Systemressourcen, die für die Erstellung und den Erhalt der Virtualisierung benötigt werden.

Netzwerk-Dienst:
Dieser Dienst wird nicht näher im Rahmen dieser Arbeit untersucht, ermöglicht aber die Bereitstellung einer einfachen Netzwerkstruktur mit Adressräumen, die von den anderen Diensten verwendet werden können. Eine Variante ist Flannel [59], die im Rahmen dieser Evaluation genutzt wurde.

Registry-Dienst:
Eine Container-Registry ist eine skalierbare serverseitige Anwendung, die von CI/CD-Systemen, Entwicklern und Testern genutzt wird, um während der Anwendungsentwicklung erstellte Images zu speichern. Diese Images werden in der Regel für Kubernetes, DevOps-Prozesse und die Entwicklung containerbasierter Anwendungen verwendet. Neben bekannten Beispielen für Container-Registries wie Docker Hub, Amazon Elastic Container Registry (ECR) und Microsoft Azure wird im Rahmen dieser Arbeit eine lokale Instanz ausgeführt, damit keine Anbindung an einen Drittanbieter notwendig ist.

Metriken-Dienst:
Dieser Dienst ergänzt die bereitgestellten Informationen zum Systemstatus der zentralen Instanz um detaillierte Informationen zu den Systemressourcen. Kubernetes stellt diesen Dienst als Kubernetes-Metrics-Server zur Verfügung [60].

6.1.2 Informationen zu Messungen und KPIs

Dieser Abschnitt definiert, in welcher Form die Messungen überwacht wurden und wie die in den Szenarien dargestellten Diagramme zu interpretieren sind.

Überwachung der Systemressourcen und Prozesszustände:
Im Rahmen der Laborumgebung wurden die Messwerte in zwei Varianten erfasst. Die erste Variante entspricht der Überwachung über die zentrale Kubernetes-Instanz, welche Metriken auf der Konsole ausgab, die dann in regelmäßigen Abständen (alle 10 Sekunden) durch eine Programm aufgezeichnet wurden. Die zweite und für die meisten Szenarien der Laborumgebung genutzte Variante ist die Entnahme der Prozessressourcen direkt aus der Betriebssystemüberwachung. Diese Variante musste lokal auf jedem Stationsgerät ausgeführt werden, konnte jedoch deutlich höhere Genauigkeiten und eine Abtastrate im Bruchteil einer Sekunde verwenden. Um einheitliche Messzeitpunkte zu erhalten, wurde eine Abtastrate von einer Sekunde gewählt.

Primär wurden die Messwerte durch KPIs gemessen, z.B. die genutzten CPU- und RAM-Leistung. Diese KPIs wurden als Diagramme in Form von Boxplots oder Liniengrafiken dargestellt. Bei den Boxplots wurden folgende Standardeigenschaften angewendet:

Eigenschaften der Quantile für Boxplots:
Ein Boxplot, auch bekannt als Kastendiagramm, ist eine Methode zur grafischen Darstellung von Datenverteilungen durch ihre Quantile. Die folgenden mathematischen Definitionen und Eigenschaften beschreiben die Standard-Quantile, die in Boxplots verwendet werden:

- **Minimum (Min):** Der kleinste Wert im Datensatz.
- **Erstes Quartil (Q1):** Das 25%-Quantil, auch unteres Quartil genannt.
- **Median (Q2):** Das 50%-Quantil, auch zweites Quartil genannt.
- **Drittes Quartil (Q3):** Das 75%-Quantil, auch oberes Quartil genannt.
- **Maximum (Max):** Der größte Wert im Datensatz.

Die Berechnung dieser Quantile kann wie folgt ausgedrückt werden:

$$Q1 = \left(\frac{1}{4}(n+1)\right) \text{-ter Datenpunkt} \tag{6.1}$$

$$Q2 = \left(\frac{1}{2}(n+1)\right) \text{-ter Datenpunkt} \tag{6.2}$$

$$Q3 = \left(\frac{3}{4}(n+1)\right) \text{-ter Datenpunkt} \tag{6.3}$$

wobei n die Anzahl der Datenpunkte im Datensatz ist.

Zusätzlich zu diesen Quantilen gibt es zwei wichtige Maße, die die Verteilung und Streuung der Daten in einem Boxplot beschreiben:

- **Interquartilsabstand (IQR):** Der Abstand zwischen dem dritten und ersten Quartil:
$$IQR = Q3 - Q1 \tag{6.4}$$
- **Whiskers (Antennen):** Diese zeigen die Extremwerte innerhalb von 1.5 mal dem IQR über oder unter den Quartilen an:
$$\text{Untere Whisker} = \max(\text{Min}, Q1 - 1.5 \cdot IQR) \tag{6.5}$$
$$\text{Obere Whisker} = \min(\text{Max}, Q3 + 1.5 \cdot IQR) \tag{6.6}$$

Datenpunkte außerhalb der Whiskers gelten als Ausreißer.

6.1.3 Szenarien und Ergebnisse

Dieses Kapitel beschreibt verschiedene Szenarien unter Verwendung der aufgeführten Dienste. Die unterschiedlichen Szenarien untersuchen die funktionalen Eigenschaften aller Artefakte unter Betrachtung der angegebenen KPIs. Die Bewertung der Ergebnisse in Bezug auf die funktionalen und nicht funktionalen Anforderungen erfolgt im Abschnitt 6.3.

Das erste Szenario ergänzt die Informationen aus dem Abschnitt 3.5 zur Umsetzung eines Virtualisierungskonzepts und der damit verbundenen Ausführung von Diensten auf Stationsgeräten. Ein weiterer wichtiger Aspekt zur Überprüfung der Anwendbarkeit des Virtualisierungskonzepts ist die Ermittlung der genutzten Systemressourcen und die Identifizierung möglicher, mit den Ressourcen verbundener Einschränkungen. Diese Evaluation der Systemressourcen erfolgt in den darauf folgenden Szenarien. Anschließend wird das Verwaltungskonzept anhand mehrerer Szenarien zur Verteilung von Diensten und der Betrachtung von resilienten

Eigenschaften durch die Bereitstellung automatisierter Prozesse bewertet. Die Laborexperimente werden durch die Betrachtung einiger Szenarien im Zusammenhang mit dem Engineeringprozess abgeschlossen.

Szenario 1: Ausführung von Diensten auf Stationsgeräten

Das Ausführen von Diensten auf Stationsgeräten kann auf unterschiedliche Weise erfolgen. Wird die Ausführung nur unter Betrachtung des Virtualisierungskonzepts aus Kapitel 3 durchgeführt, können Dienste direkt über die Konsoleneingabe ausgeführt werden, wie beispielsweise:

```
sudo docker run -d --name SE --cpus=".2" --privileged
--network SE_network --ip 192.168.5.13 -it SE-Dienst
```

Durch diesen Befehl lässt sich ein Image, in diesem Fall ein SE-Dienst, mit einer festen IP und privilegierten Systemrechten starten. Der Name SE wird festgelegt und die nutzbare CPU-Leistung auf 20 Prozent eines Kerns begrenzt. Ein Beispiel für die genutzten Dienste ist in Tabelle 6.1 dargestellt.

Tab. 6.1.: Ausschnitt der evaluierten Dienste

Container ID	Image	Befehle	Status	Namen
73a4cb01c193	SE-Dienst	/bin/sh -c ’./home/...	Up 3 seconds	SE-3
aca87985ecac	SE-Dienst	/bin/sh -c ’./home/...	Up 14 seconds	SE-2
f037ffdeec83	ubuntu	tail -f /dev/null	Up 28 seconds	Null-1
021ca02bcef5	SE-Dienst	/bin/sh -c ’./home/...	Up 34 seconds	SE-1

Weitere Möglichkeiten erfolgen über den kontinuierlichen Engineeringprozess und der damit verbundenen codebasierten Beschreibungen. Eine detailierte Beschreibung ist in Abschnitt 5.3.3 beschrieben.

Szenario 2: Ressourcenbedarf einer Container-Instanz

Dienste und Prozesse: Ein Leerlauf-Dienst, Laufzeitumgebung (Containerd) und Container-Verwaltung (Dockerd)
KPIs: CPU- und RAM-Bedarf einer Container-Instanz, Ressourcenbedarf durch Containerd und Dockerd
Ergebnisse:

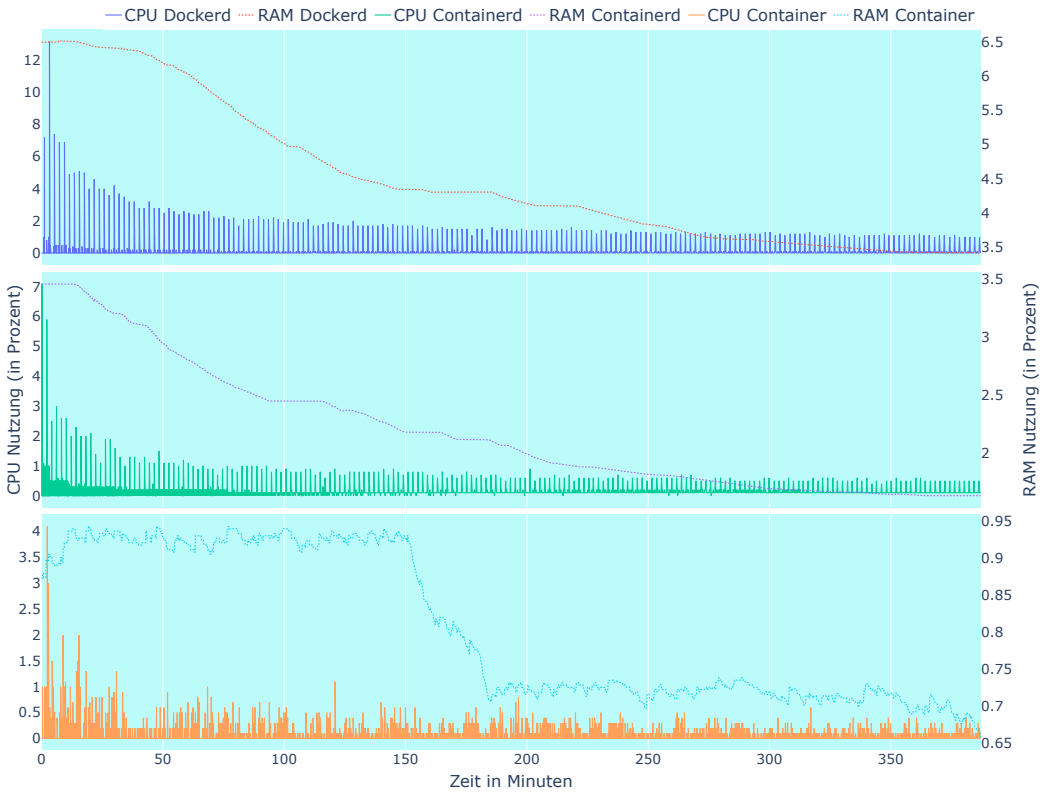

Abb. 6.2.: Ressourcenbedarf für eine Container-Instanz

Die Ergebnisse aus Abbildung 6.2 beziehen sich auf die Durchführung eines Leerlauf-Dienstes zur Überprüfung des entstandenen Ressourcenbedarf für eine Container-Instanz ohne weitere Funktionalität. Die Darstellung ist in drei getrennte Diagramme unterteilt: die Prozesse des ausgeführten Containers (unten), die Laufzeitumgebung *Containerd* (Mitte) und die Container-Verwaltung *Dockerd* (oben). Jeder Prozess wird durch die CPU-Nutzung (linke Y-Achse) und die RAM-Nutzung (rechte Y-Achse) in Prozent repräsentiert.

Die Systemressourcen beziehen sich in dieser Analyse primär auf die CPU-Leistung (angegeben in Prozent der Maximalleistung eines Rechenkerns) und die RAM-Leistung (angegeben in Prozent des gesamten Arbeitsspeichers). Die Angabe der CPU-Leistung in Bezug auf nur einen Rechenkern ermöglicht einen einfacheren Vergleich bei Mehrkern-Prozessoren.

Unter Berücksichtigung der benötigten Grundlast durch die beteiligten Technologien Containerd und Dockerd besteht ein Einfluss durch die Initialisierung eines Dienstes mit Spitzen von bis zu 20 Prozent CPU-Nutzung und etwa 10 Prozent RAM-Nutzung. Der Bedarf nimmt kontinuierlich über die Zeit ab. Dieser Trend lässt sich auch durch

den bereitgestellten Container-Prozess erkennen, wobei hier die Schwankungen deutlich geringer ausfallen.

Szenario 3: Ressourcenbedarf für zwanzig Container-Instanzen

Dienste und Prozesse: Zwanzig Leerlauf-Dienste
KPIs: CPU- und RAM-Bedarf für jede Container-Instanz
Ergebnisse:

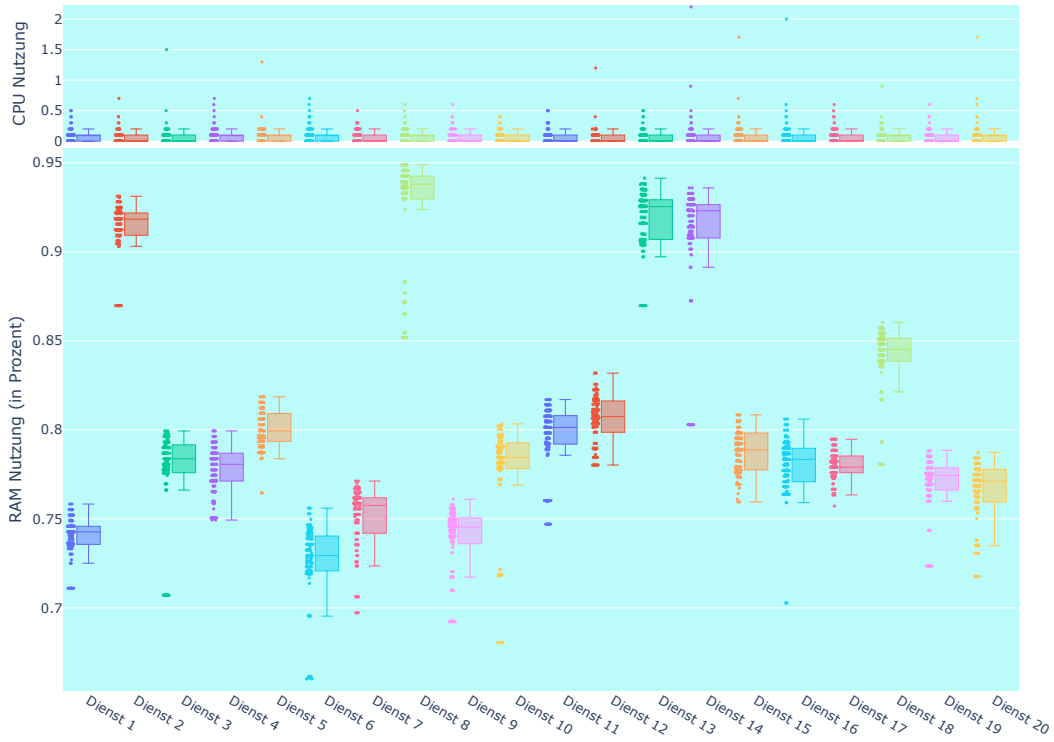

Abb. 6.3.: Ressourcenbedarf der Virtualisierung bei 20 Containern

Zur besseren Einordnung der Ressourcennutzung mehrerer parallel ausgeführter Container-Instanzen zeigen die Diagramme in Abbildung 6.3 insgesamt 20 Leerlauf-Dienste als Boxplots, jeweils für die CPU-Nutzung und die RAM-Nutzung. Die Schwankungen der RAM-Nutzung liegen im Bereich von sieben bis zehn Tausendsteln der verfügbaren Arbeitsspeicherleistung. Geringfügige Abweichungen sind vor allem zwischen den einzelnen Containern erkennbar, obwohl diese identisch konfiguriert sind. Die benötigte CPU-Leistung eines Containers ist nahezu vernachlässigbar und liegt bei weniger als einem halben Prozent der Leistung eines Kerns. Es fehlt jedoch noch die Betrachtung der Skalierung von Diensten in Bezug auf den Anstieg der

Ressourcennutzung der Technologien Containerd und Dockerd. Diese Betrachtung wird im nächsten Szenario ergänzt.

Szenario 4: Skalierender Ressourcenbedarf der beteiligten Prozesse

Dienste und Prozesse: 1-15 Leerlauf-Dienste, Laufzeitumgebung (Containerd) und Container-Verwaltung (Dockerd)
KPIs: CPU- und RAM-Bedarf für Dockerd und Containerd
Ergebnisse:

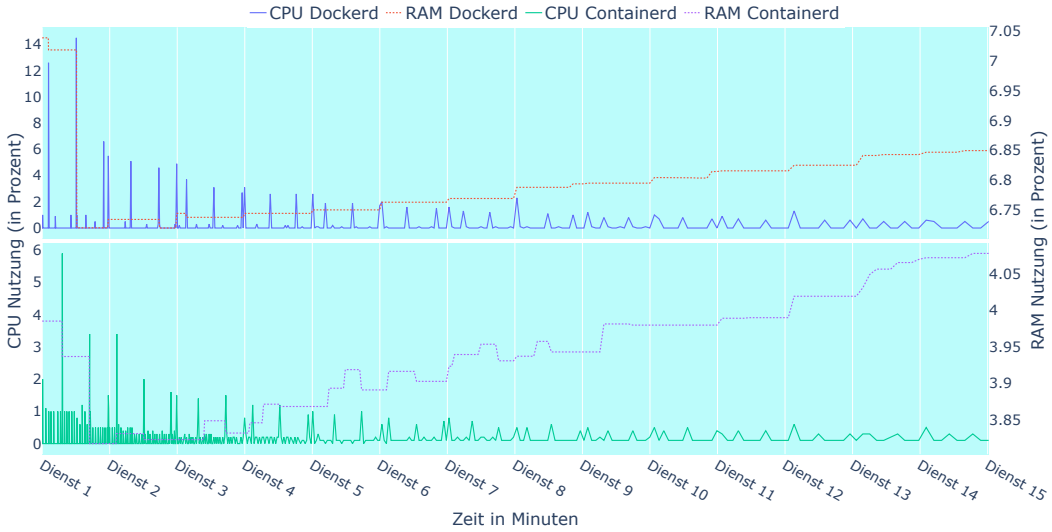

Abb. 6.4.: Ressourcenbedarf bei steigender Anzahl von Diensten

Ein Vergleich der ansteigenden Ressourcennutzung mit zunehmender Anzahl ausgeführter Container ist in Abbildung 6.4 dargestellt. Das untere Diagramm entspricht den KPIs der Laufzeitumgebung Containerd, während das obere Diagramm die Container-Verwaltung (Dockerd) darstellt. Für beide Prozesse wird die CPU- und RAM-Nutzung in zwei separaten Graphen dargestellt. Die X-Achse stellt die Anzahl gleichzeitig ausgeführter Leerlauf-Dienste in einem Container dar, wobei alle 5 Minuten eine Instanz hinzugefügt wurde. Ähnlich zu den Ergebnissen im Szenario 6.1.3 steigt die CPU- und RAM-Nutzung initial bei der Erstellung der ersten Container-Instanz an. Ein gleiches Verhalten lässt sich auch bei der Reduzierung des CPU-Bedarfs über die Zeit erkennen, trotz der Ergänzung weiterer Instanzen. Anders sieht es bei der RAM-Auslastung aus: Beide Prozesse steigern ihren Bedarf zur Bereitstellung der Laufzeitumgebung und der Verwaltung der Container mit der Anzahl ausgeführter Instanzen. Wird die Skala der RAM-Nutzung betrachtet, so ist ein Anstieg von weniger als einem halben Prozent für beide Prozesse bei insgesamt

15 ausgeführten Container-Instanzen zu erkennen. Im Vergleich zur RAM-Auslastung von annähernd einem Prozent pro Container-Instanz aus dem vorherigen Szenario ist dieser Anstieg zu vernachlässigen und bietet keine Einschränkungen in der Skalierung von Diensten.

Szenario 5: Ressourcenbedarf eines SE-Dienstes

Dienste und Prozesse: Ein SE-Dienst, Laufzeitumgebung (Containerd) und Container-Verwaltung (Dockerd)
KPIs: CPU- und RAM-Bedarf des SE-Dienstes, Ressourcenbedarf durch Containerd und Dockerd
Ergebnisse:

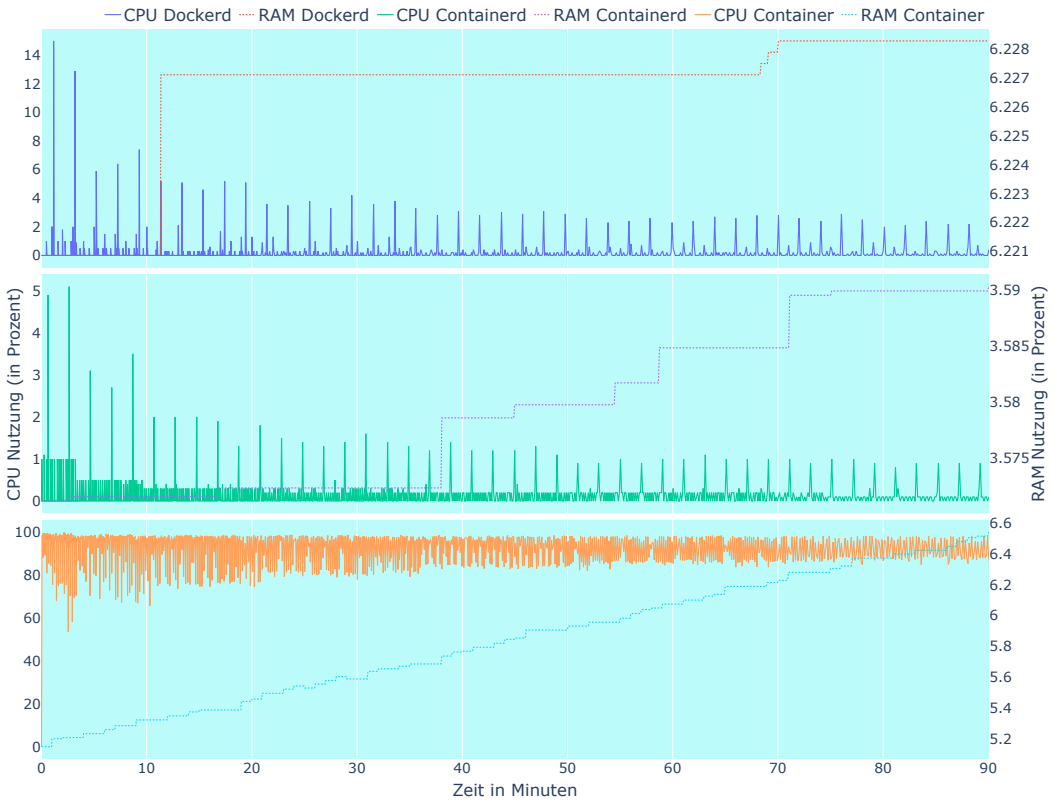

Abb. 6.5.: Ressourcenverbrauch beim Ausführen einer Netzzustandsschätzung

Dieses Szenario zeigt in Abbildung 6.5 den Ressourcenbedarf der Laufzeitumgebung (Containerd) und der Container-Verwaltung (Dockerd) unter der Ausführung einer Netzzustandsschätzung. Im Vergleich zum Leerlauf-Dienst, der nahezu keine Ressourcen benötigt, beansprucht die Netzzustandsschätzung in dieser Implementierung konstant zwischen 80 und 100 Prozent eines CPU-Kerns (unteres Diagramm).

Diese Implementierung führt kontinuierlich Simulationen durch und bewertet die einzelnen Abgänge eines Verteilnetzes. Ohne zeitliche Begrenzung werden so viele CPU-Ressourcen wie möglich genutzt. Im direkten Vergleich bleibt der CPU-Bedarf von Containerd (mittleres Diagramm) und Dockerd (oberes Diagramm) unverändert.

Anders verhält es sich bei der RAM-Auslastung: Im Gegensatz zum Leerlauf-Dienst, bei dem eine deutliche Reduzierung der RAM-Nutzung über die Zeit festgestellt wurde, bleibt die RAM-Auslastung der Netzzustandsschätzung nahezu konstant. Die RAM-Auslastung nimmt jedoch kontinuierlich zu, was auf eine fehlerhafte Implementierung des Speichermanagements des Dienstes hindeutet. Eine detailliertere Analyse dieser fehlerhaften Implementierung wird in der Evaluation im Verteilnetz durchgeführt.

Szenario 6: Parallel ausgeführte SE-Dienste

Dienste und Prozesse: Vier SE-Dienste
KPIs: CPU- und RAM-Bedarf der SE-Dienste
Ergebnisse:

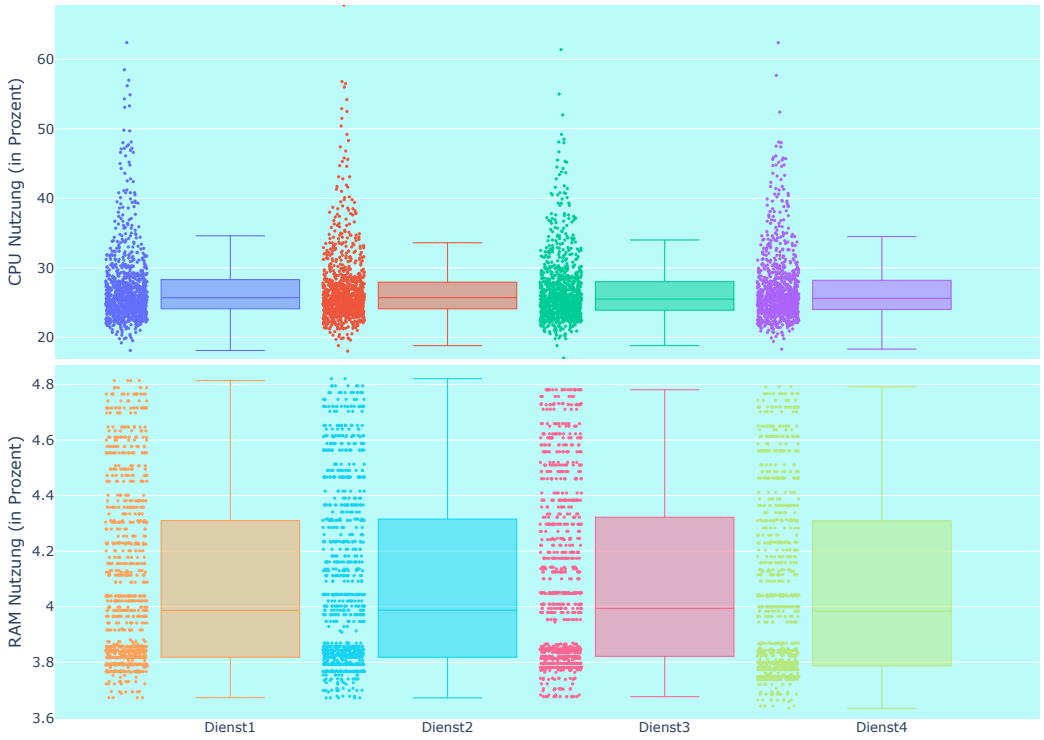

Abb. 6.6.: Parallele Ausführung von 4 Netzzustandsschätzungen auf einem Stationsgerät

Im Hinblick auf die parallele Ausführung mehrerer Dienste wurde die Netzzustandsschätzung viermal auf einem Stationsgerät ausgeführt. Die Nutzung der Systemressourcen ist in Abbildung 6.6 als Boxplot dargestellt. Das untere Diagramm zeigt die RAM-Nutzung, während das obere Diagramm die Nutzung der CPU-Leistung darstellt. Die X-Achse beschreibt die vier ausgeführten Dienste. Eine mögliche Interpretation der Ergebnisse ist die Einschränkung der Dienste in ihrer maximal nutzbaren CPU-Leistung. Genauer gesagt, fordert jeder Dienst so viel CPU-Leistung wie möglich an, wobei insgesamt nur zwei CPU-Kerne verfügbar sind. Dies führt zu stark variierender CPU-Nutzung und kann einen Einfluss auf andere systemrelevante Prozesse des Betriebssystems haben.

Eine weitere Betrachtung ist die Variation der RAM-Nutzung, die im direkten Vergleich zwischen den Diensten identisch ausfällt und anteilig an den verfügbaren Systemressourcen unkritisch erscheint. Betrachtet man den linearen Anstieg aus der vorherigen Abbildung und die Unterschiede in den Quartilen der Boxplots, die ebenfalls auf einen Anstieg der RAM-Nutzung hinweisen, dann kann ein Einfluss auf andere Prozesse, wie systemrelevante Betriebssystemprozesse, entstehen. Einschränkungen des verfügbaren Arbeitsspeichers können zu Fehlern bei Speicherallokationen führen, die zum Absturz der Prozesse beitragen. Entsprechend muss eine Beschränkung der zur Verfügung gestellten Systemressourcen je Container-Instanz ermöglicht werden.

Szenario 7: Einschränkungen der Ressourcennutzung

Dienste und Prozesse: Vier SE-Dienste
KPIs: CPU- und RAM-Bedarf eingeschränkter SE-Dienste
Ergebnisse:

In diesem Szenario wurde die CPU-Leistung aller laufenden Dienste auf 20 Prozent reduziert, um eine bessere Verteilung der verfügbaren CPU-Leistung zu ermöglichen. Abbildung 6.7 zeigt, wie im vorherigen Szenario, die CPU- und RAM-Nutzung von insgesamt vier Diensten zur Netzzustandsschätzung. Das obere Diagramm verdeutlicht die Begrenzung der CPU-Leistung auf etwa 20 Prozent, wobei Schwankungen entsprechend dem Completely Fair Scheduler (CFS) [61] durch die Bereitstellung eines virtuellen Laufzeitkontos entstehen können. Das bedeutet, dass durch die Verdrängung einzelner Prozesse ein Defizit entstehen kann, wodurch kurzfristig eine minimal höhere CPU-Nutzung zum Ausgleich des Defizits erlaubt wird. Es gilt zu beachten, dass durch die Nutzung von Virtualisierung und Threading-Verfahren auf Betriebssystemen keine vollständige Synchronität der CPU-Leistung erreicht werden

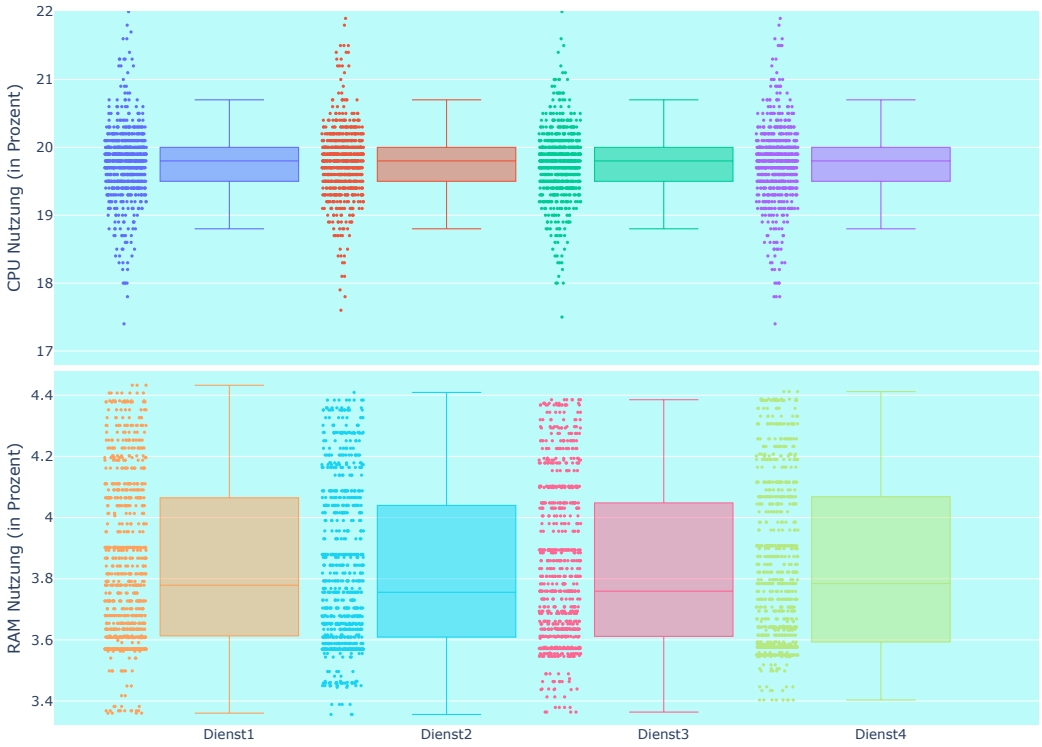

Abb. 6.7.: Einschränkung der CPU-Leistung auf 20 Prozent je Dienst

kann. Um diese Synchronität zu erreichen, müsste eine dedizierte Hardware-CPU nur einen Prozess ausführen. Folglich ist ein Szenario mit vier Diensten und nur zwei CPU-Kernen nicht in der Lage, diese Synchronität zu gewährleisten. Dennoch ist sichergestellt, dass im Mittel nicht mehr als 20 Prozent der CPU-Leistung genutzt werden. Aus Sicht der funktionalen Eigenschaften ist die Einschränkung der Ressourcennutzung essenziell, um Interferenzen zwischen ausgeführten Diensten zu vermeiden.

Szenario 8: Automatisierte Wiederherstellung von Diensten

Dienste und Prozesse: Ein SE-Dienst und Orchestrierungsprozess (Kubernetes)
KPIs: CPU- und RAM-Bedarf und Wiederherstellungszeit eines Dienstes
Ergebnisse:

Abbildung 6.8 zeigt die CPU- und RAM-Nutzung des Orchestrierungsprozesses Kubernetes (oberes Diagramm), der im Vergleich zur Laufzeitumgebung und Container-Verwaltung einen kontinuierlich hohen Bedarf an CPU-Ressourcen aufweist. Im Vergleich dazu bleibt die RAM-Nutzung mit etwa 6,6 Prozent konstant. Im unteren Diagramm ist ein auf 20 Prozent eingeschränkter Dienst dargestellt, der zweimal

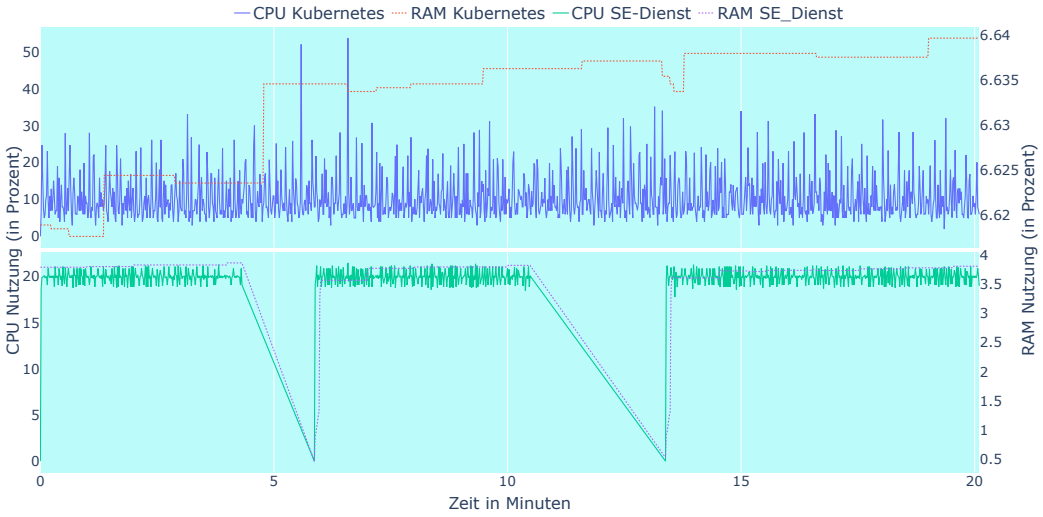

Abb. 6.8.: Darstellung einer Wiederherstellungsroutine

aktiv als Prozess terminiert wurde. Diese Terminierung wurde als Prozess „kill“ innerhalb der Prozessüberwachung im Betriebssystem vorgenommen. Die Erfassung, Bereinigung der alten Instanz und der Neustart mit Berechnung dauert einige Minuten. Die Bereinigung der alten Instanz ist essenziell, da der Netzzustandsschätzung besondere Eigenschaften wie Netzwerk (z.B. Ports), Namensbelegungen, die nur einmalig vorkommen dürfen, und Speicherorte für Dateien zugeordnet sind. Um Konflikte zu vermeiden, muss eine Instanz erst vollständig terminiert werden, bevor ein neuer Dienst gestartet werden kann. Die Fähigkeit, Dienste automatisiert wiederherzustellen, ist gegeben und unterstützt die Eigenschaften eines resilienten Systems.

Szenario 9: Anpassung der Parametrisierung eines Dienstes

Dienste und Prozesse: Ein SE-Dienst und Orchestrierungsprozess (Kubernetes)
KPIs: CPU- und RAM-Bedarf sowie Wiederherstellungszeit eines Dienstes
Ergebnisse:

Dieses Szenario demonstriert die Anpassung der Parameter eines Dienstes mittels Kubernetes-Orchestrierung. Während die vorherige Abbildung den Ausfall eines Dienstes darstellte, zeigt Abbildung 6.9 einen geplanten Neustart eines Dienstes. Das obere Diagramm illustriert die Ressourcennutzung durch Kubernetes, die keine signifikanten Veränderungen während der Parameteranpassung aufweist. Im unteren Diagramm ist erkennbar, dass die CPU- und RAM-Nutzung für etwa zwei Minuten unterbrochen ist und anschließend ein höherer Bedarf an CPU-Leistung angefragt

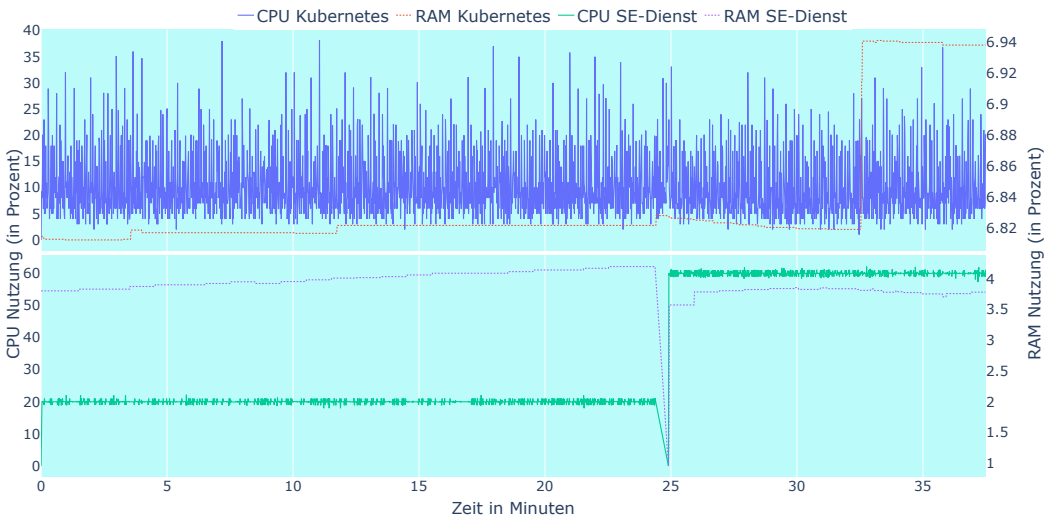

Abb. 6.9.: Anpassung der Konfiguration eines Dienstes

wird. Diese Anpassung entspricht der Dienst-Parametrisierung aus dem manuellen Engineering-Prozess 5.2.3 und wird anschließend als Teil des kontinuierlichen Engineering-Prozesses ausgeführt. Im Vergleich zum Absturz eines Dienstes erfolgt der Neustart durch Parameteränderung schneller, da die Fehlererfassung entfällt.

6.2 Aufbau und Evaluation im Verteilnetz

Die Evaluation in diesem Abschnitt basiert auf den Ergebnissen aus einem Feldtestgebiet im Verteilnetz eines deutschen Netzbetreibers. Der strukturelle Aufbau im Verteilnetz ist in Abbildung 6.10 dargestellt. Gezeigt wird ein Ausschnitt eines Mittelspannungsnetzes mit insgesamt 36 Ortsnetzstationen und zwei getrennten Netzabschnitten, die über zwei Trennstellen verbunden sind. Im Rahmen des BMWK-geförderten Forschungsprojektes i-Autonomous [13] wurden insgesamt elf (türkis dargestellt) Stationen digital ausgestattet. Diese Ausstattung umfasst Sensoren, Stationsgeräte und Router, die die Erfassung, Verarbeitung und Kommunikation von Messwerten ermöglichen. Die Kommunikation wurde ausschließlich über Mobilfunk realisiert, weshalb Antennen und Empfänger in der Darstellung integriert sind. Die gewählte Darstellung ist eine schematische strukturelle Darstellung und weicht aufgrund von Datenschutzgründen von der exakten Ausrichtung im Netzgebiet ab. Ergänzend zur Netzstruktur im Verteilnetz wurde eine IT-Infrastruktur des dazugehörigen Verteilnetzbetreibers genutzt. Neben einer sicheren Kommunikation über VPN-Netze wurden Server für die Bereitstellung der zentralen Verwaltungsinstanz

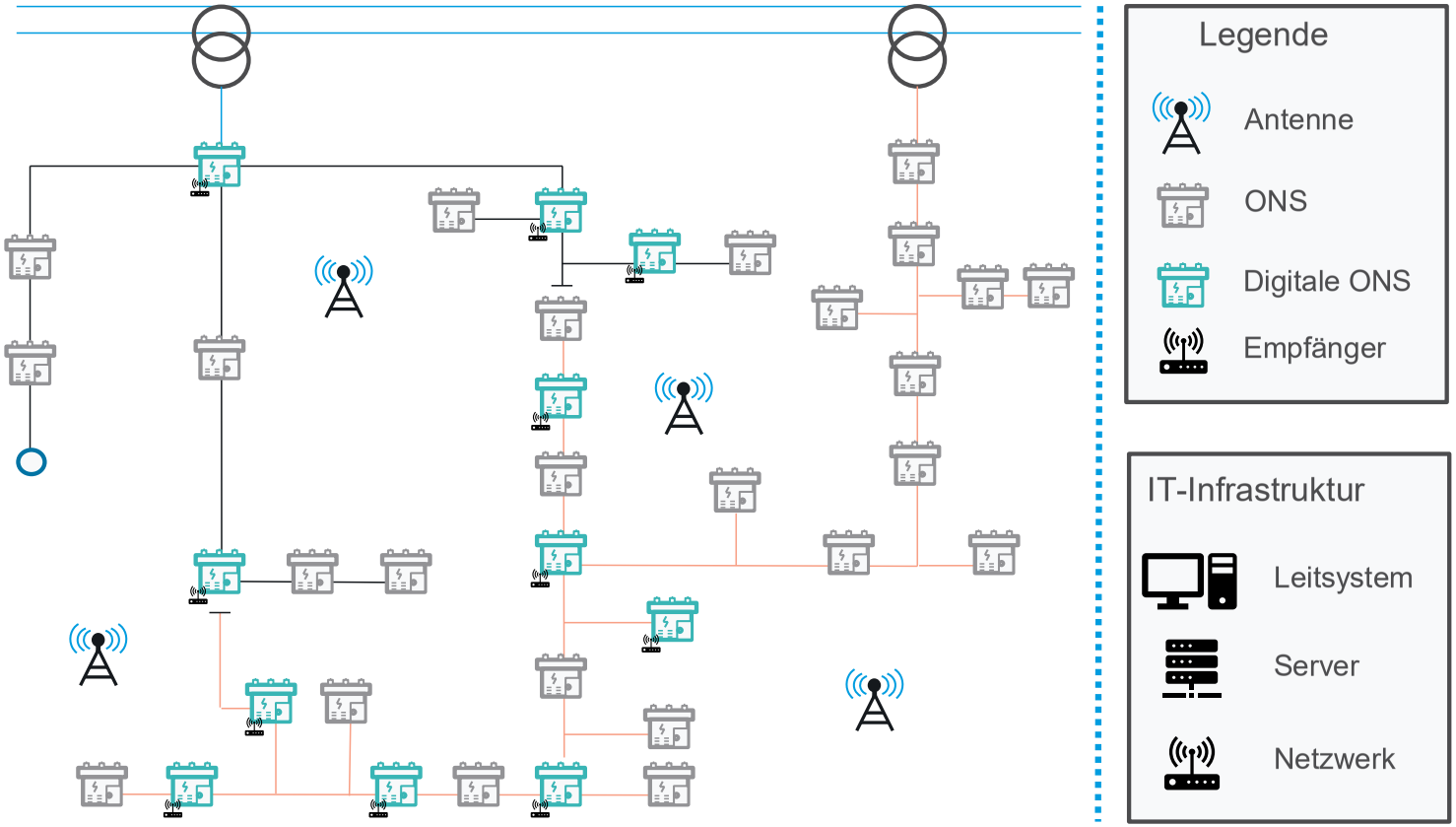

Abb. 6.10.: Struktureller Aufbau im Verteilnetz

bereitgestellt. Die endgültigen Ergebnisse wurden in einer Art Leitsystemstruktur visuell dargestellt.

Im Rahmen dieser Dissertation wurde die Evaluation im Verteilnetz genutzt, um die Funktionsweise der Artefakte in einem CPES zu demonstrieren. Insbesondere die realen Bedingungen durch die örtlich und physikalisch getrennte Verteilung der Dienste über alle Stationen hinweg und der Einfluss der regional schwankenden Verfügbarkeit der Kommunikation unterschieden sich vom Aufbau im Labor.

Im Folgenden wird die Versuchsdurchführung im Verteilnetz als ein gesamtes Szenario beschrieben. Dazu werden die Informationen zu den bereitgestellten Diensten, die Vorbedingungen, der Ablauf und die Ergebnisse beschrieben.

6.2.1 Vorbedingungen

In Vorbereitung auf die Durchführung im Verteilnetz wurden einige Vorbedingungen erfüllt. Dazu gehörten alle notwendigen Schritte zur Digitalisierung der Ortsnetzstationen. Dies umfasste den Einbau der Sensoren, Stationsgeräte und die Kommunikationsanbindungen. Die Stationsgeräte wurden vorkonfiguriert, sodass jedes Stationsgerät zum Einbau bereits alle notwendigen Technologien implementiert sowie eine VPN-Konfiguration und Netzwerkkonfiguration vorlag. Dadurch konnte gewährleistet werden, dass der Fernzugriff auf die Geräte ermöglicht wurde. Über diesen Fernzugriff wurden alle Stationsgeräte der einzelnen Ortsnetzstationen in das Cluster der Orchestrierungssoftware Kubernetes eingebunden und die Kommunikation überprüft. Einige IP- und Port-Freigaben mussten nachträglich noch erfolgen. Serverseitig mussten ebenfalls alle notwendigen Technologien wie Docker, Kubernetes, eine Registry und eine Kubernetes-Web-Oberfläche installiert und konfiguriert werden. Damit die Netzzustandsschätzung korrekt funktioniert, mussten alle Stationsgeräte und der Server zeitsynchronisiert werden. Hierzu wurde das Network Time Protocol (NTP) verwendet.

6.2.2 Beschreibung der Dienste

Ergänzend zu den bereits beschriebenen Diensten wurden für diese Evaluation zwei wichtige Dienste hinzugefügt: der Basiscontainer und ein auf echte Messwerte basierende Netzzustandsschätzung. Darüber hinaus wurden verschiedene Netzwerkdienste für die Bereitstellung der Orchestrierung mit Kubernetes und Dienste zur Überwachung der Dienste integriert. Eine Darstellung aller verwendeten Dienste ist

in der Tabelle 6.2 bereitgestellt, wobei die Dienste nachfolgend kurz beschrieben werden.

Tab. 6.2.: CPU- und RAM-Nutzung der verwendeten Dienste nach [50]

Datum Y:M:D	Uhrzeit	Dienst	CPU (in %)	RAM (in %)
23:04:21	07:20:00	Basiscontainer1	3.9	0.6
23:04:21	07:20:00	Basiscontainer2	4.8	0.6
23:04:21	07:20:00	Basiscontainer3	3.5	0.6
23:04:21	07:20:00	Basiscontainer4	4.7	0.6
23:04:21	07:20:00	Basiscontainer5	2.1	0.6
23:04:21	07:20:00	Basiscontainer6	4.8	0.6
23:04:21	07:20:00	Basiscontainer7	3.4	0.6
23:04:21	07:20:00	Basiscontainer8	4.7	0.6
23:04:21	07:20:00	Basiscontainer9	4.6	0.6
23:04:21	07:20:00	Basiscontainer10	3.5	0.6
23:04:21	07:20:00	Basiscontainer11	3.4	0.6
23:04:21	07:20:00	Netzzustandsschätzung	91.2	3.4
23:04:21	07:20:00	iautonomous-registry-pod	0.1	2.0
23:04:21	07:20:00	coredns	0.3	0.8
23:04:21	07:20:00	coredns	0.3	0.8
23:04:21	07:20:00	kube-apiserver	6.0	24.1
23:04:21	07:20:00	kube-controller-manager	1.5	5.2
23:04:21	07:20:00	kube-scheduler	0.3	1.8
23:04:21	07:20:00	kube-flannel	0.8	1.4
23:04:21	07:20:00	kube-flannel	0.9	0.8
23:04:21	07:20:00	kube-flannel	0.8	0.8
23:04:21	07:20:00	kube-flannel	1.0	0.8
23:04:21	07:20:00	kube-flannel	1.0	0.8
23:04:21	07:20:00	kube-flannel	0.9	0.8
23:04:21	07:20:00	kube-flannel	0.9	0.8
23:04:21	07:20:00	kube-flannel	0.8	0.7
23:04:21	07:20:00	kube-flannel	0.9	0.8
23:04:21	07:20:00	kube-flannel	0.8	0.8
23:04:21	07:20:00	kube-flannel	1.0	0.8
23:04:21	07:20:00	kube-flannel	0.3	1.4
23:04:21	07:20:00	kube-proxy	0.1	1.5
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.3	1.1
23:04:21	07:20:00	kube-proxy	0.3	1.1
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.1	1.9
23:04:21	07:20:00	kube-proxy	0.1	1.1
23:04:21	07:20:00	kube-proxy	0.2	1.1
23:04:21	07:20:00	metrics-server	0.5	1.9
23:04:21	07:20:00	kubernetes-dashboard	0.1	1.6

Basiscontainer:
Dieser Dienst stellt auf jedem der Stationsgeräte die Schnittstelle zwischen Container und Messtelemetrie bereit. Das bedeutet, dieser Dienst empfängt über die E/A-Schnittstelle die gewandelten Messinformationen als Kleinsignale und stellt diese über einen IEC 61850 MMS Protokollstack anderen Diensten auf dem eigenen oder anderen Stationsgeräten zur Verfügung. Der Dienst wird mit speziellen Systemrechten gestartet, um den Zugriff auf die E/A-Schnittstellen zu gewährleisten. Zudem nutzt der Dienst den für MMS üblichen Port 102.

Netzzustandsschätzer auf Basis echter Messwerte:
Der Netzzustandsschätzer entspricht algorithmisch der Netzzustandsschätzung aus der Laborevaluation. Die Unterschiede liegen hauptsächlich in der Konfiguration, die der Netzsimulation im Verteilnetz angepasst ist, sowie in der Verwendung realer Messwerte für jede digitale Ortsnetzstation. Diese Messwerte werden durch den Basiscontainer bereitgestellt und über IEC 61850 MMS empfangen. Auch die Netzzustandsschätzung verwendet den standardisierten Port 102 für die Kommunikation über MMS.

Netzwerk-Dienste:
Die Netzwerkdienste umfassen „coreDNS“, „kube-flannel“ und „kube-proxy“. „coreDNS“ kümmert sich innerhalb des Kubernetes-Clusters um die Namensauflösung und Verwaltung der IP-Adressen, sodass sich die Dienste innerhalb des Kubernetes-Netzwerks erreichen können. „kube-flannel“ ist eine Container-Netzwerk-Schnittstelle und ermöglicht in diesem Fall die Kommunikation zwischen Diensten auf unterschiedlichen Stationsgeräten im Cluster. Während „coreDNS“ nur lokal auf der zentralen Kubernetes-Instanz ausgeführt wird, muss „kube-flannel“ auf allen Stationsgeräten ausgeführt werden. Dies gilt auch für „kube-proxy“, der auf jedem Gerät ausgeführt wird und den ein- und ausgehenden Netzwerkverkehr steuert.

Überwachungsdienste:
Die Überwachungsdienste umfassen den „metric-server“ und die Weboberfläche von Kubernetes. Der „metric-server“ ermöglicht die Erfassung und Darstellung von Systemressourcen und Zustandsdaten der Stationsgeräte und Dienste. Die Weboberfläche kann in Zusammenarbeit mit dem „metric-server“ die Zustandsdaten und Systemressourcen visualisieren sowie Konfigurationen und Einstellungen direkt ausführen, anstatt eine Code-basierte Beschreibung zu verwenden.

Kubernetes-Dienste:
Intern verwendet Kubernetes mehrere Dienste wie „kube-apiserver“, „kube-controller-manager“ und „kube-scheduler“ für verschiedene Aufgaben, die zusammen die voll-

ständige Funktionalität von Kubernetes repräsentieren. Eine ausführliche Erklärung dieser Dienste ist im Abschnitt 4.3.4 bereitgestellt.

6.2.3 Ablauf der Versuchsdurchführung

Der Ablauf der Versuchsdurchführung ist im Sequenzdiagramm der Abbildung 6.11 skizziert. Die Versuchsdurchführung beginnt mit der Erstellung einer SCD-Datei und deren Konvertierung in eine YAML-Datei. Der genaue Prozess ist im Abschnitt des Engineeringprozesses 5.2.3 detailliert beschrieben. Diese Beschreibung umfasst die Ausführung der Basiscontainer auf allen elf Stationsgeräten. Das Stationsgerät fordert den benötigten Dienst als Image bei der Registry an, empfängt diesen und führt die Basiscontainer aus. Der SE-Dienst wird anschließend als weiterer Dienst ausgeführt, wobei dieser nur auf einem Stationsgerät betrieben wird. Dabei ist zu beachten, dass eine sinnvolle Netzzustandsermittlung nur durchgeführt werden kann, wenn alle Basiscontainer die Messwerte der Station bereitstellen. Ergänzend wird die kontinuierliche Überwachung gemäß des kontinuierlichen Engineeringprozesses 5.2.4 durchgeführt. Die entsprechenden Ergebnisse sind im nächsten Abschnitt dargestellt. Aus Sicht des Operators werden die Ergebnisse der Netzzustandsschätzung über eine Leitsystem-Applikation visuell angezeigt. Eine entsprechende Darstellung ist in Abbildung 6.10 zu sehen.

6.2.4 Ergebnisse

Dieser Abschnitt beschreibt die Ergebnisse der Versuchsdurchführung unter Berücksichtigung der genutzten Systemressourcen und stellt diese in einem Ausschnitt der Leitsystem-Applikation visuell dar.

Durch die kontinuierliche Überwachung des Systemstatus und der Systemressourcen wurde die Betrachtung der genutzten Ressourcen für alle Basiscontainer ermöglicht. Die Daten wurden entsprechend dem Format der Tabelle 6.2 bereitgestellt. Insgesamt wurden über einen Zeitraum von 72 Stunden etwa 2,4 Millionen Datenpunkte über alle Dienste in einer Auflösung von 10 Sekunden erfasst. Im Rahmen dieser Arbeit wird die Analyse und Darstellung auf die Betrachtung der Basiscontainer und der Netzzustandsschätzung begrenzt.

Abbildung 6.12 zeigt die CPU- und RAM-Nutzung für jeden Basiscontainer auf den Stationsgeräten. Mit Ausnahme des ersten Basiscontainers liegt die genutzte Arbeitsspeicher bei etwa sechs Prozent der verfügbaren Ressourcen, während die

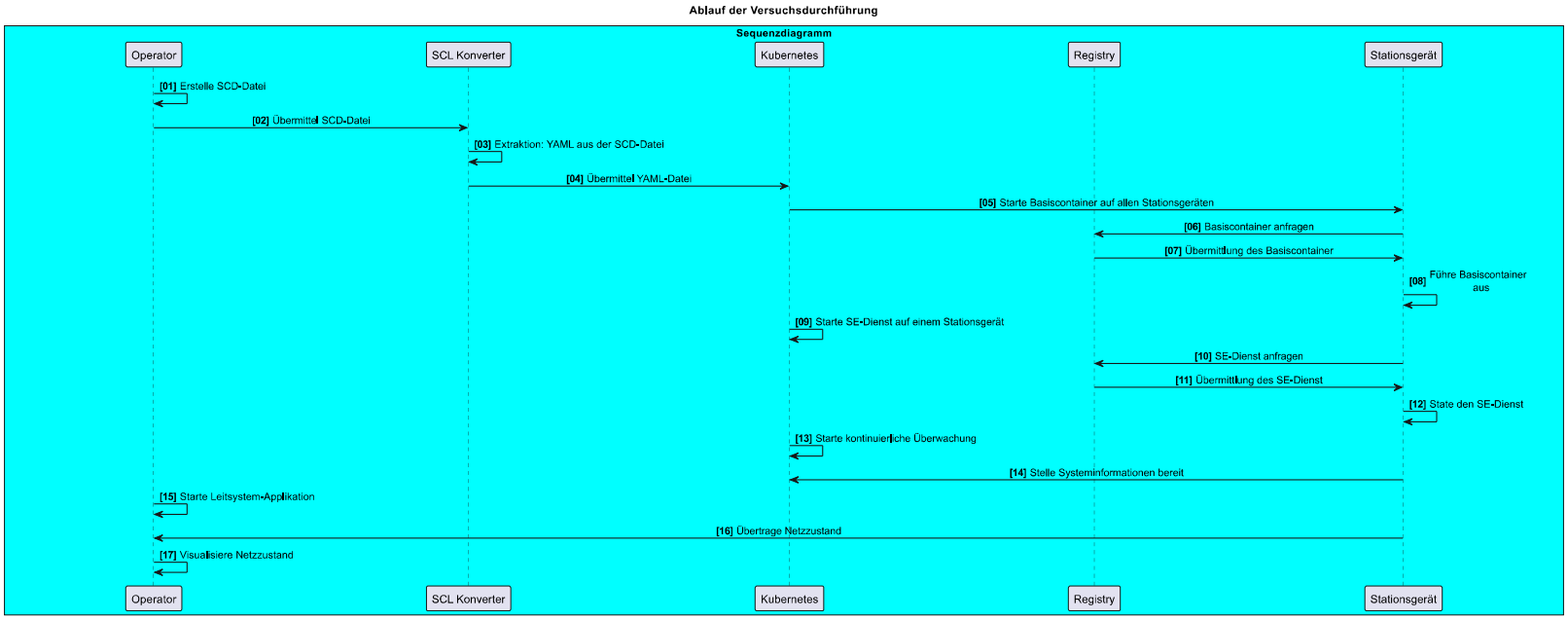

Abb. 6.11.: Sequenzdiagramm: Ablauf der Versuchsdurchführung

CPU-Nutzung stärker im Bereich von zwei bis sechs Prozent schwankt. Daraus lässt sich schließen, dass die Basiscontainer nur einen geringfügigen Verbrauch der Systemressourcen aufweisen und über einen längeren Zeitraum stabil laufen.

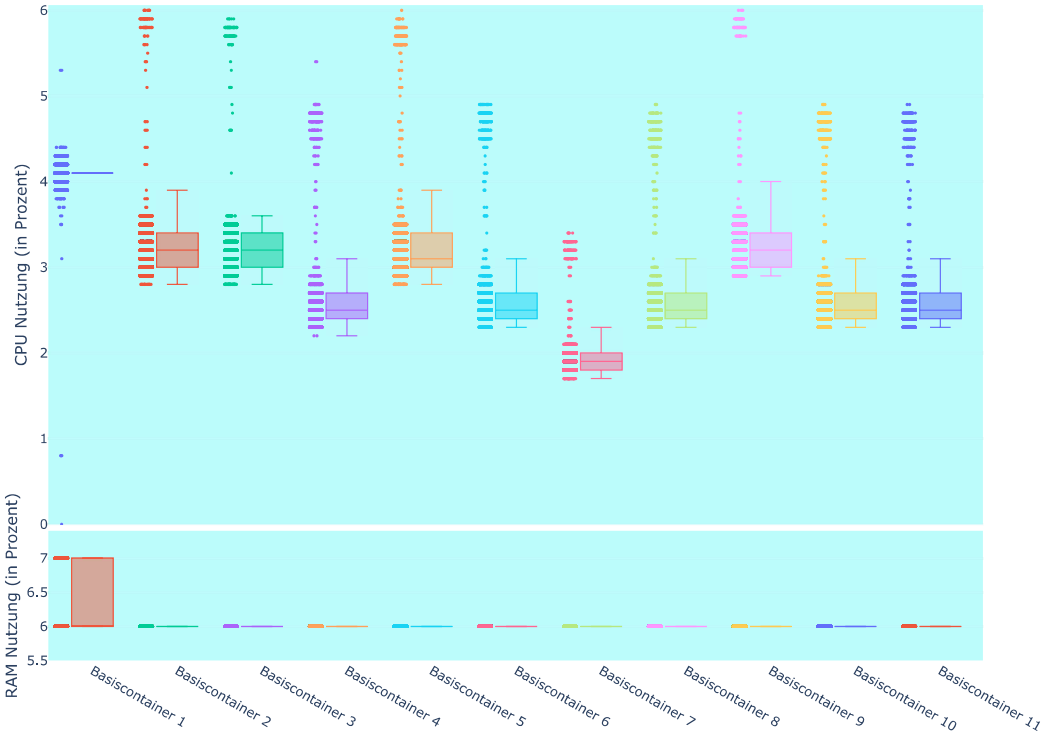

Abb. 6.12.: Ausführung eines Basiscontainer in elf Ortsnetzstationen

Die nächste Betrachtung bezieht sich auf die Ausführung der Netzzustandsschätzung und des Basiscontainers auf einem Stationsgerät. Beide Dienste verwenden die Kommunikation über IEC 61850 MMS und benötigen dafür den Port 102. Dieser Port ist fest im Kommunikationsstack implementiert und lässt sich nicht parametrieren. Stationsgeräte, die keine Virtualisierung unterstützen, nutzen meist einen Kommunikationsstack für alle Anwendungen auf dem Stationsgerät. Dies ist im Fall der Nutzung virtualisierter Dienste nicht möglich, da jeder Dienst logisch getrennt und als eigenständiges IED betrachtet wird. Es gibt verschiedene Möglichkeiten, dieses Problem zu lösen, wobei die Vor- und Nachteile für jeden Anwendungsfall berücksichtigt werden müssen. Eine Möglichkeit ist das Port-Mapping, bei dem ein Dienst innerhalb der Instanz den Port 102 nutzen kann, während der Port außerhalb der Instanz auf einen anderen Port weitergeleitet wird (z.B. 1000), der fortlaufend um eins je Dienst erhöht wird. Der Nachteil dieses Mappings besteht darin, dass andere Dienste, die eine aktive Kommunikation über den Port 102 durchführen möchten, diesen nicht nutzen können, sondern an den geänderten Port ihre Nachrichten senden müssen. Eine Alternative ist die Nutzung sogenannter Overlay-

Kommunikationsnetze, die z.B. durch Kubernetes innerhalb des Clusters oder durch Cloud-Dienst-Anbieter bereitgestellt werden können. Dies ermöglicht jeder virtuellen Instanz, eine eigene interne IP-Adresse zu nutzen, wobei alle Ports über die eigentliche IP-Adresse des Stationsgerätes getunnelt werden. Der Nachteil dieser Lösung besteht darin, dass alle Geräte innerhalb des Clusters oder des Netzwerks des Cloud-Dienst-Anbieters sein müssen. Eine weitere Alternative, die in dieser Arbeit Anwendung findet, ist die Bereitstellung eines festen Subnetzes für jeden Container. In diesem Feldtest wurden jedem Gerät acht IP-Adressen fest zugewiesen. Der Nachteil dieser Lösung liegt in der begrenzten Anzahl ausführbarer Dienste mit eigener IP-Adresse. Unter Berücksichtigung der aktuellen begrenzten Auswahl verfügbarer Dienste für Stationsgeräte und der Limitierung durch die verfügbaren Ressourcen je Gerät ist dies keine direkte Einschränkung. Zusätzlich sind die acht IP-Adressen je Stationsgerät nicht fest vorgegeben, und es könnten auch größere Subnetzbereiche mit 16, 32, 64 oder mehr IP-Adressen je Stationsgerät freigegeben werden.

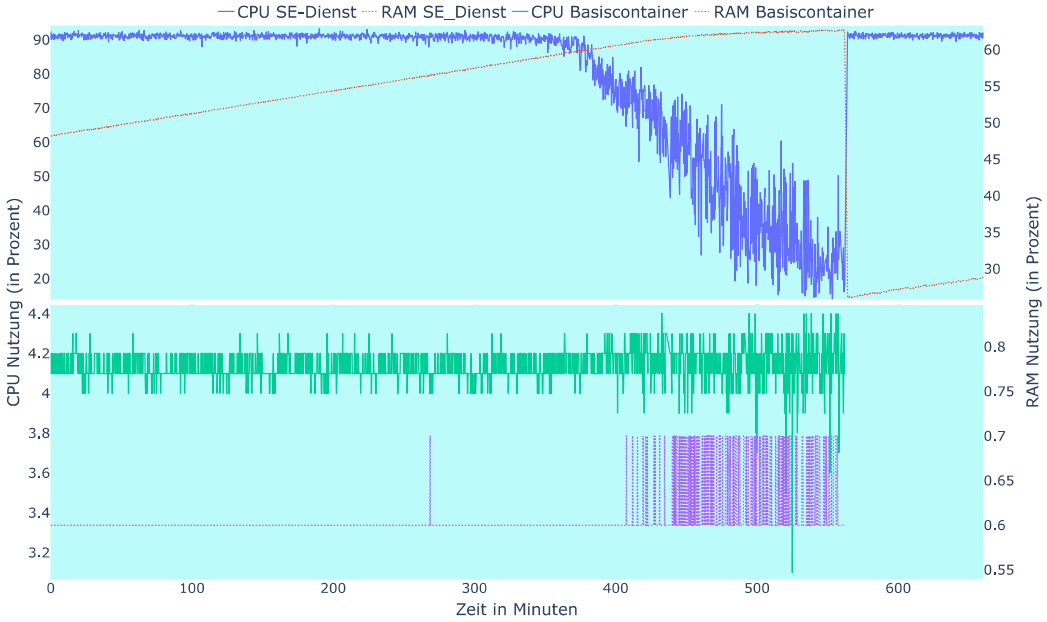

Abb. 6.13.: Absturz durch RAM-Überlauf bei der Netzzustandsschätzung

Nachdem die Ausführung beider Dienste auf einem Stationsgerät ermöglicht wurde, zeigt Abbildung 6.13 die Systemressourcen beider Dienste. Das untere Diagramm zeigt die CPU- und RAM-Nutzung des Basiscontainers, während das obere Diagramm eine Netzzustandsschätzung ohne Limitierung der Ressourcen zeigt. In dieser Abbildung können mehrere Probleme identifiziert werden. Erstens steigt die RAM-Nutzung des SE-Dienstes kontinuierlich an und führt zu einer begrenzten Verfügbarkeit von Arbeitsspeicher. Zweitens nimmt die CPU-Leistung zum Zeitpunkt

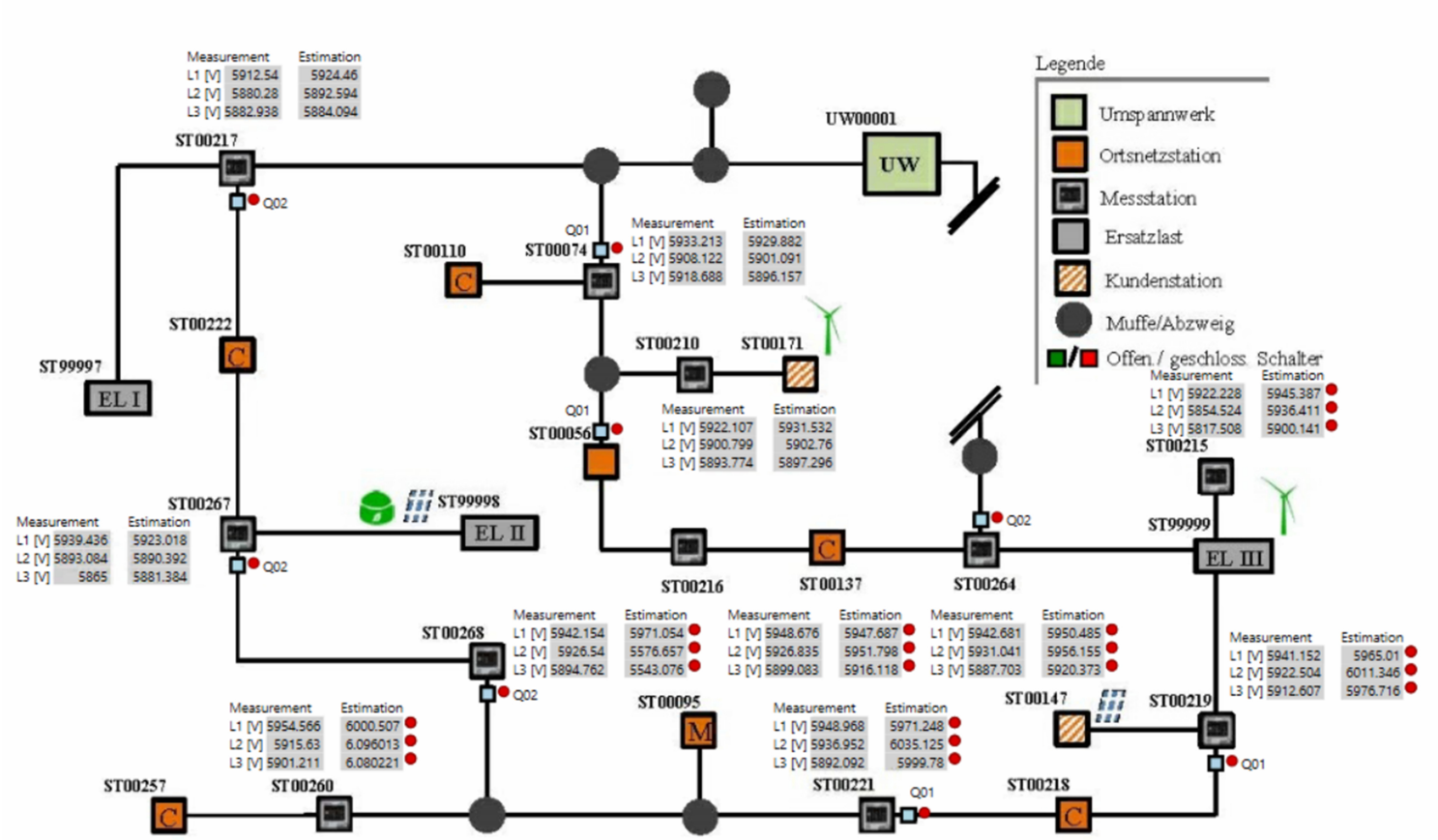

Abb. 6.14.: Darstellung der Netzzustandsdaten in einem Leitsystem [13]

der begrenzten Verfügbarkeit des Arbeitsspeichers ab, was auf eine Einschränkung durch zu wenig verfügbaren Arbeitsspeicher hinweist. Dieser Prozess erfolgt so lange, bis der Dienst abstürzt und gemäß der automatisierten Wiederherstellung durch Kubernetes neu gestartet wird. Drittens fällt der ausgeführte Basiscontainer gleichzeitig mit dem SE-Dienst aus, was darauf schließen lässt, dass auch für diesen Dienst nicht genügend Arbeitsspeicher verfügbar war. Der Dienst wurde nicht neu gestartet, da keine entsprechende Routine konfiguriert wurde. Eine Lösung zur Vermeidung von Seiteneffekten auf andere Prozesse im gesamten System (es könnten auch relevante Betriebssystemprozesse betroffen sein) ist die Limitierung der Ressourcen von Diensten. Beispielsweise könnte die RAM-Nutzung auf 50 Prozent begrenzt werden. Dies würde den fehlerhaften Dienst nicht vor dem Absturz bewahren, jedoch könnten Seiteneffekte dadurch vermieden werden, da andere Prozesse noch genügend Ressourcen zur Verfügung haben.

Die Versuchsdurchführung wird mit der Zusammenführung und Darstellung aller Netzzustandsdaten in einer visuellen Darstellung eines für das Netzgebiet ausgelegten Leitsystems, wie in Abbildung 6.14 zu sehen, abgeschlossen. Diese Abbildung ist im Rahmen des Forschungsprojektes i-Autonomous [13] entstanden und zeigt, dass alle Container-Instanzen ihre Funktionalität erfüllen und über IEC 61850 MMS an eine Leitstelle kommunizieren konnten. Die Ergebnisse der Netzzustandsschätzung sind sowohl für die Messwerte als auch für die berechneten Werte in den grauen Kästen dargestellt. Die Bewertung der Dienste im Hinblick auf die Messwerte und die geschätzten Ergebnisse der Netzzustandsschätzung liegt außerhalb des Rahmens dieser Arbeit. Einige Ergebnisse dazu können in den entsprechenden Abschlussberichten der TIB [62] unter den Forschungsberichten i-Automate und i-Autonomous eingesehen werden.

6.3 Interpretation und Bewertung

In diesem Abschnitt werden die Ergebnisse und Methoden im Bezug auf die funktionalen und nicht-funktionalen Anforderungen (FA) interpretiert und bewertet. Dies beinhaltet die FA 1-8 und die NFA 1-4 sowie den Bezug zu den Artefakten 1-3.

6.3.1 Funktionale Anforderungen - Artefakt 1

Die erste spezifische Forschungsfrage untersucht relevante Hardware- und Softwareanforderungen zur Ausführung isolierter Dienste auf Stationsgeräten. Insgesamt

bilden drei funktionale Anforderungen die Grundlage für ein Virtualisierungskonzept (Artefakt 1).

Die erste FA fordert die simultane Ausführung einer oder mehrerer Dienste auf einem Stationsgerät. Die Untersuchung der notwendigen Hardware- und Software-Infrastruktur zur Umsetzung eines Virtualisierungskonzepts wurde in Abschnitt 3.4 behandelt. Zusätzlich wurden im Szenario 6 die Anforderungen mehrerer simultan ausgeführter Dienste mit insgesamt vier Diensten zur Netzzustandsschätzung gezeigt. Diese Anforderungen beschreiben zudem den Bedarf der isolierten Betrachtung der genutzten Ressourcen sowie eine unabhängige, konstante und sichere Ausführung der Dienste. Abschnitt 3.2.1 beschreibt die Eigenschaften von Virtualisierung, darunter die Abstraktion von Ressourcen und die Fähigkeit zur Isolation und Sicherheit einzelner Instanzen. Die Abstraktion der Ressourcen ist beispielsweise im Szenario 2 (Ressourcenbedarf einer Container-Instanz) dargestellt. Die funktionale Anforderung 1 ist somit vollständig erfüllt.

Die zweite FA beschreibt die Notwendigkeit des Zugriffs der Dienste auf Peripherie- und Netzwerkeigenschaften der Stationsgeräte. Im Abschnitt 3.2.3 wird konzeptionell der Zugriff auf spezielle Hardwareressourcen wie Netzwerke und E/A-Schnittstellen zu Peripheriegeräten geregelt (z.B. über Betriebssystem-Capabilities). Im Abschnitt 6.2.2 wurde der Konflikt bei paralleler Nutzung der E/A-Schnittstellen diskutiert und als Lösung ein Basiscontainer vorgestellt. Dieser Basiscontainer bildet die Schnittstelle zwischen einem oder mehreren Diensten und der verfügbaren E/A-Schnittstelle und wird mit zusätzlichen Rechten gestartet, um den funktionalen Anforderungen zu entsprechen. Ein weiterer Konflikt bezüglich der parallelen Nutzung der Netzwerke wurde im Abschnitt 6.2.4 der Versuchsdurchführung im Verteilnetz diskutiert. Dabei wurden mehrere Lösungsansätze wie Port-Mapping, Netzwerkmanagement und die Bereitstellung mehrerer IP-Adressen pro Stationsgerät erörtert. Diese Anforderung lässt sich sowohl konzeptionell als auch technologisch und anwendungsorientiert umsetzen.

Die letzte funktionale Anforderung an das Virtualisierungskonzept beschreibt die Umsetzung einer statischen Ressourcenplanung. D.h. die Systemressourcen eines Stationsgerätes müssen einzelnen Diensten durch Begrenzungen der maximal zugewiesenen und durch minimal verfügbare Ressourcen verwaltet werden. Die konzeptionelle Umsetzung ist in Abschnitt 3.2.3 zur Beschreibung der Verwaltungsschicht beschrieben. Die Auswirkungen nicht regulierter Ressourcennutzung sind in den Szenarien fünf und sechs diskutiert. Im fünften Szenario ist die uneingeschränkte Nutzung der CPU-Leistung durch einen SE-Dienst dargestellt. Konkret wird die CPU-Leistung im sechsten Szenario diskutiert, in dem starke Schwankungen der

CPU-Leistung zwischen vier ausgeführten Diensten dargestellt sind, da jeder Dienst möglichst viel CPU-Leistung anfragt. Als Lösung wird die Einschränkung der CPU-Leistung unter der CFS-Verwaltungsstrategie von Container-Instanzen betrachtet, die im Szenario sieben dargestellt ist. Die CPU-Leistung über alle vier Dienste ist stabil bei etwa 20 Prozent der Leistung eines Kerns. Eine weitere Analyse im Versuchsaufbau des Verteilnetzes zeigte, dass die kontinuierliche Erhöhung der RAM-Nutzung eines Dienstes Seiteneffekte auf andere Dienste projizierte. Nachdem kein Arbeitsspeicher mehr verfügbar war, stürzten bei der Allokation von Arbeitsspeicher andere Dienste ab. Die Einschränkung der maximal verfügbaren RAM-Nutzung des Dienstes hätte verhindern können, dass Seiteneffekte auftreten. Die Darstellung und Durchführung unterschiedlicher Szenarien haben gezeigt, dass das Virtualisierungskonzept den funktionalen Anforderungen entspricht. Damit wurden alle funktionalen Anforderungen des ersten Artefakts erfolgreich umgesetzt.

6.3.2 Funktionale Anforderungen - Artefakt 2

Das zweite Artefakt steht in Verbindung zur spezifischen Forschungsfrage 2 „Welche Konzepte zur Verwaltung, Überwachung und Steuerung von virtualisierten Diensten lassen sich im Kontext von Edge-Computing für Stationsgeräte nutzen“. In dem Zusammenhang wurden die funktionalen Anforderungen vier und fünf spezifiziert.

Die vierte FA stellt die grundlegende Anforderung zur Verteilung von Diensten über Stationsgeräte hinweg. Dazu gehört die Auswahl der Dienste, die innerhalb des Konzepts als Teil der IaC erfolgt. Der Abschnitt 4.2 beschreibt innerhalb des Verwaltungskonzepts die Beschreibung von Diensten und all ihrer Konfigurationen. Die technische Umsetzung dieser Anforderung wird im Abschnitt 4.5 detailliert beschrieben, einschließlich der vollständigen Spezifikation eines Dienstes. Zudem wird in der funktionalen Anforderung die Überprüfung der verfügbaren Ressourcen gefordert, was durch die beschriebenen Überwachungsdienste gewährleistet wird. Diese Dienste ermöglichen beispielsweise die Erstellung von Systemressourcenanalysen für die einzelnen Diagramme. Die Anforderung zur Übertragung der Software-Instanz und die Initialisierung des Dienstes ist technologisch im Abschnitt 4.5.2 dargestellt und fest im Ablauf der Versuchsdurchführung in Abschnitt 6.2.3 verankert. In beiden Evaluationen wurden die Dienste in einer Registry abgelegt und konnten von allen Geräten abgerufen werden. Zusätzlich wurden die Dienste der Szenarien acht und neun sowie der Evaluation im Verteilnetz mittels Kubernetes verteilt.

Die fünfte funktionale Anforderung bezieht sich auf die kontinuierliche Überwachung von Diensten und die Bereitstellung automatisierter Interaktionen. Beide

Punkte sind integraler Bestandteil des Verwaltungskonzept und in den Abschnitten 4.2 und 3.2.3 sowohl konzeptionell als auch technologisch beschrieben. Der Nutzen automatisierter Prozesse wird auch in der Evaluation thematisiert, insbesondere in den Szenarien acht und neun, wo ausgefallene Dienste automatisiert wiederhergestellt oder neu konfiguriert werden. Weitere Anwendungsgebiete für die Durchführung automatisierter Routinen sind im Sequenzdiagramm 5.16 demonstriert. Dazu gehört die Verteilung von Diensten auf andere Stationsgeräte, falls ein Gerät ausfällt oder die Dienste nicht neu gestartet werden können. Die Ergebnisse der Evaluation und der technologischen Beschreibung unterstützen alle geforderten funktionalen Anforderungen an ein Verwaltungskonzept und bestätigen die Fragestellung dieser Forschungsfrage.

6.3.3 Funktionale Anforderungen - Artefakt 3

Ziel des dritten Artefaktes ist die Integration des Virtualisierungs- und Verwaltungskonzepts in die standardisierten Systemautomatisierungsprozesse als Teil eines Engineeringprozesses. Der Engineeringprozess in Artefakt drei muss die folgenden funktionalen Anforderungen erfüllen.

Die sechste FA beschreibt die Unterstützung eines standardisierten Systemautomatisierungsprozesses. Im Abschnitt 5.1 wurden verschiedene bestehende Engineeringprozesse untersucht. Als Basis wurde der Engineeringprozess der IEC 61850 Stationsautomatisierung ausgewählt und entsprechend um die Konzepte der anderen Artefakte erweitert. Grundlegende Schritte im Engineeringprozess sind die Beschreibung, Konfiguration und Parametrisierung der Stationsgeräte mit den auszuführenden Anwendungen. Die in den Szenarien ausgeführte Netzzustandsschätzung wurde im Rahmen des Engineeringprozesses nach IEC 61850 beschrieben und konfiguriert. Dabei handelt es sich um eine einmalige initiale Durchführung im Engineeringprozess und ist grundlegend als initialer Engineeringprozess, wie im Abschnitt 5.2.1 beschrieben, erhalten geblieben.

Ergänzt wird der standardisierte Engineeringprozess um das Virtualisierungs- und Verwaltungskonzept, um der siebten funktionalen Anforderung zu entsprechen. Ergänzt wurden Prozessschritte eines kontinuierlichen Engineeringprozesses (Abschnitt 5.2.2), die sowohl die manuelle Anpassung und Beschreibung von Diensten zur Laufzeit der Stationsgeräte erlauben, als auch einen automatisierten Prozessablauf definieren, der den Ablauf zur Ausführung der IaC im Verwaltungskonzept repräsentiert. Im Abschnitt 5.3 konnte gezeigt werden, dass SCD-Dateien nach IEC 61850 um Dienstbeschreibungen erweiterbar sind und Dienste vollständig automatisiert über

den kontinuierlichen Engineeringprozess ausgeführt werden können. Die einzelnen Prozessschritte wurden durch den Ablauf des Virtualisierungs- und Verwaltungskonzepts beschrieben. Die Beschreibung zum Ablauf der Versuchsdurchführung beschreibt in einem Sequenzdiagramm die Abläufe im gesamten Engineeringprozess. Insgesamt konnte gezeigt werden, dass die Konzepte aus Artefakt eins und zwei in einen kontinuierlichen Engineeringprozess integrierbar sind. Dies beinhaltet vor allem auch die kontinuierliche Überwachung und Durchführung automatisierter Routinen, wie sie in den Szenarien acht, neun und in der Versuchsdurchführung im Verteilnetz erfolgt sind.

Die letzte funktionale Anforderung beschreibt die Notwendigkeit, standardisierte Kommunikation zwischen Diensten innerhalb und außerhalb eines Stationsgeräts zu ermöglichen. Der Engineeringprozess wurde so gewählt, dass ein Dienst als eigenständiges IED konfiguriert wird. Zusammen mit der Beschreibung zum Basiscontainer (Abschnitt 6.2.2) wird die Kommunikation innerhalb eines Geräts gewährleistet. Gleichzeitig wurden verschiedene Lösungsansätze der mehrfachen Nutzung von Netzwerkschnittstellen in den Ergebnissen der Versuchsdurchführung diskutiert und als Lösung die Bereitstellung einer festen Anzahl an IP-Adressen je Stationsgerät gewählt. Dadurch wird eine klare Schnittstelle außerhalb des Stationsgeräts für jeden Dienst ermöglicht, sodass die standardisierte Kommunikation zwischen Station und anderen Geräten im Netzwerk zur Verfügung steht. Ein Beispiel ist in den Ergebnissen der Abbildung 6.14 ersichtlich, in der über IEC 61850 Messdaten und berechnete Werte der Netzzustandsschätzung an eine grafische Oberfläche außerhalb des Kubernetes-Klusters gesendet wurden.

Alle funktionalen Anforderungen der drei Artefakte wurden erfolgreich unterstützt und durch Konzepte, technologische Umsetzungen und Ergebnisse aus Labor- und Verteilnetzevaluation bestätigt.

6.3.4 Nicht funktionale Anforderung - Skalierbarkeit

Die Anforderungen der Skalierbarkeit untersucht die bereitgestellten Konzepte und Prozesse in Bezug auf die Anzahl ausführbarer Dienste innerhalb eines Stationsgerätes und im Kontext der stationsübergreifenden Verwaltung eines größeren Netzgebiets. Unterschiedliche Szenarien beziehen sich auf die Analyse der Belastung eines Stationsgerätes durch die bereitgestellte Virtualisierung von Diensten. Im zweiten Szenario 6.1.3 wurde die Grundlast der Technologien Dockerd und Containerd im Verhältnis zur Ausführung einer leeren Container-Instanz analysiert und bewertet. Die nachfolgenden Szenarien drei und vier analysierten den

Ressourcenbedarf für zwanzig Container-Instanzen und mögliche Erhöhungen der notwendigen Systemressourcen für die Laufzeitumgebung (Containerd) und die Container-Verwaltung (Dockerd) bei einer kontinuierlich steigenden Anzahl erstellter Container-Instanzen. In allen Szenarien konnte ermittelt werden, dass die Grundlast nur minimal mit der Anzahl ausgeführter Container-Instanzen steigt und die notwendigen Ressourcen für eine Container-Instanz im Vergleich zu einem darin laufenden Dienst minimal sind und selbst bei 20 parallel laufenden Diensten geringfügig ausfallen. Neben der Rechenleistung und dem Arbeitsspeicher wurden mögliche Einschränkungen bei der gleichzeitigen Nutzung des Netzwerks sowie verschiedene Lösungsmöglichkeiten in der Evaluation der Versuchsdurchführung im Verteilnetz diskutiert. Jedem Stationsgerät können beliebig viele physische oder logische IP-Adressen zugewiesen werden, um eine Skalierung der Dienste zu unterstützen. Die verwendeten Technologien sind für skalierbare Systeme ausgelegt und besitzen keine Einschränkungen bei der Skalierung von Diensten auf einem Stationsgerät.

Die Einbindung der Stationen in die Orchestrierung durch Kubernetes ermöglicht die einfachere Verwaltung skalierender Container-Instanzen und ist ein fester Bestandteil des Verwaltungskonzepts. Kubernetes beschreibt einige Begrenzungen in der Skalierung [63]. Das bedeutet, dass auf jedem Knoten (entspricht einem Stationsgerät) maximal 110 Pods gleichzeitig ausgeführt werden können. Betrachtet man die aktuelle Situation eines Smart Grids, so werden meist wenige Dienste im einstelligen Bereich in Ortsnetzstationen ausgeführt. Selbst unter Berücksichtigung zunehmender Digitalisierungen und zukünftiger Szenarien sollte die Obergrenze ausreichen. Zudem müsste die Hardware zunächst für die Unterstützung von bis zu 110 Diensten ausgelegt werden. In Bezug auf die Anzahl eingebundener Knoten gibt Kubernetes eine Obergrenze von 5000 an. Betrachtet man alle Ortsnetzstationen einiger Netzbetreiber, könnte die Obergrenze nicht ausreichen. Bisher ist die Digitalisierung der Ortsnetzstationen noch im Ausbau. Zudem lassen sich parallel mehrere Cluster für unterschiedliche Netzgebiete aufbauen, die jeweils eine Teilmenge der verfügbaren Knoten abdecken. Weiterhin ist eine gesamte Obergrenze von 300.000 Container-Instanzen durch Kubernetes innerhalb eines Clusters vorgegeben. Das würde im Schnitt eine Begrenzung von 60 Instanzen je Knoten bedeuten und lässt sich ebenfalls mit den oben genannten Argumenten relativieren. Aus diesem Grund sind keine Einschränkungen für die Nutzung der vorgestellten Konzepte vorhanden.

6.3.5 Nicht funktionale Anforderung - Flexibilität

Die zweite nicht funktionale Anforderung evaluiert die Nutzung unterschiedlicher Dienste auf einem Stationsgerät und die Anwendbarkeit des Engineeringprozesses in Bezug auf die Stationsunabhängigkeit. Die flexible Nutzung unterschiedlicher Dienste auf einem Stationsgerät ist beispielsweise in der Auflistung 6.2 aller für die Versuchsdurchführung benötigten Dienste abgedeckt. Außerdem zeigt die Abbildung 6.13 die parallele Ausführung des Basiscontainers und der Netzzustandsschätzung auf einem Gerät.

Die Unabhängigkeit des Engineeringprozesses in Bezug auf die verwendete Station und die unterschiedlichen Spannungsebenen gilt in allen Stationen, in denen auch die IEC 61850 Stationsautomatisierung Anwendung findet. Generell ist diese ohnehin für die Übertragungsnetze ausgelegt und wurde nur im Rahmen dieser Arbeit und des Projektes i-Autonomous auf die Ebene der Ortsnetzstationen angewendet. Im Rahmen der Versuchsdurchführung im Verteilnetz wurden ausschließlich Ortsnetzstationen im Spannungsbereich von 10 bis 20 kV verwendet.

6.3.6 Nicht funktionale Anforderung - Interoperabilität

Unter Interoperabilität untersucht diese Arbeit mögliche Einschränkungen bei der Interoperabilität zwischen Diensten innerhalb eines Stationsgerätes und über Stationsgeräte hinweg. Ebenfalls soll die herstellerunabhängige Bereitstellung und Nutzung von Diensten untersucht werden. Die Interoperabilität zwischen Diensten wird generell durch die Repräsentation eines Dienstes als eigenständiges IED in der Modellierung des Engineeringprozesses garantiert. Das bedeutet, jeder Dienst wird im Engineeringprozess behandelt, als wäre es ein eigenständiges IED und besitzt dadurch auch die damit verbundene Interoperabilität. In der technologischen Umsetzung wurde jedem Dienst eine eigene physische IP-Adresse gegeben, sodass eine Kommunikation entsprechend den Netzwerkrichtlinien des Netzbetreibers frei im Kommunikationsnetz erfolgen kann. Ein Beispiel ist in der Darstellung der Messwerte und Netzzustandsdaten der Abbildung 6.14 dargestellt. Dieses Leitsystem vereinigt alle Messdaten und Netzzustandsdaten aller Ortsnetzstationen durch unterschiedliche Dienste.

Im Bezug auf die herstellerunabhängige Nutzung von Diensten im gesamten Prozess gilt, dass die verwendeten Technologien für den kontinuierlichen Engineeringprozess als Open-Source-Lösungen verfügbar sind (beschrieben in den Technologieanalysen 3.3 und 4.3). Dienste unterschiedlicher Hersteller können in den Engineeringprozess

eingebunden werden. Beispielsweise wurden im Rahmen dieser Arbeit Dienste der Technischen Universität Dortmund, öffentlich zugängliche Dienste und eigene Implementierungen genutzt.

6.3.7 Nicht funktionale Anforderung - Wartbarkeit

Die Wartbarkeit als nicht funktionale Anforderung untersucht die Möglichkeiten, um Dienste anzupassen, auszutauschen oder zu verändern. Der bereitgestellte Engineeringprozess und die beteiligten Konzepte zur Virtualisierung und Verwaltung von Diensten ermöglichen den Umgang mit z.B. Updates oder Anpassungen. Aus Sicht der IaC-Beschreibung ist eine gewünschte Änderung nichts anderes als eine abgeänderte Konfiguration, die erfasst und durch die bestehende ausgetauscht wird. Das Verwaltungskonzept hingegen stellt unterschiedliche Möglichkeiten bereit, wie eine Änderung durchgeführt wird. Die einfachste Form ist der Austausch eines Dienstes durch Terminierung des vorherigen Dienstes und das Ersetzen durch einen neuen Dienst. Dieser Fall ähnelt dem Szenario 8, wobei statt des vorherigen Dienstes ein anderer Dienst gestartet wird. Eine weitere Form des Updates ist die Änderung der Parameter der Container-Instanz, wie im Beispiel des Szenarios 9, in dem das Limit der CPU-Nutzung von 20 Prozent auf 60 Prozent angehoben wurde. Darüber hinaus gibt es spezielle „Rolling-Updates“, die es ermöglichen, Dienste unter spezifischen Bedingungen auszutauschen, z.B. mit der Bedingung, dass immer mindestens eine aktive Instanz eines Dienstes vorgehalten wird. Trotz all dieser Verfahren ist die Wartbarkeit abhängig von der gewählten Technologie und sollte im Einzelfall überprüft werden.

7 Fazit und Ausblick

Dieses Kapitel fasst die Arbeit dieser Dissertation zusammen und zieht ein Fazit, indem auf die Forschungsfragen aus Abschnitt 1.6 eingegangen wird. Anschließend werden die Einschränkungen dieser Arbeit erörtert und weiterer Forschungsbedarf abgeleitet.

7.1 Fazit

Die Notwendigkeit der digitalen Aufbereitung in Stationsautomatisierungsprozessen zukünftiger CPES hin zu einem Smart Grid ist ein zentraler Bestandteil dieser Dissertation. Der sichere und stabile Betrieb des Energiesystems wird künftig stärker von ergänzenden Dienstleistungen [4] und damit vom Bedarf neuer Steuerungs- und Überwachungsmechanismen im Verteilnetz abhängen. Entsprechende Dienste müssen in die Struktur bestehender Stationsautomatisierungssysteme integriert werden. Dabei sind Anforderungen wie Skalierbarkeit, Fernwartbarkeit und Standardisierung aufgrund der hohen Anzahl zu unterstützenden Stationen im Verteilnetz und der kritischen Infrastruktur unabdingbar. Diese Arbeit hat verschiedene Konzepte und Paradigmen der Informatik auf bestehende Mechanismen der Stationsautomatisierung angewendet, um den Herausforderungen zukünftiger Smart Grids zu begegnen. Es wurden drei spezifische Ausprägungen der zentralen Forschungsfrage im Rahmen dieser Dissertation untersucht.

Spezifische FF1: *Wie lassen sich relevante Hardware- und Softwareanforderungen für die Virtualisierung von Diensten in Stationsgeräten abbilden?*

Im Kapitel 3 wurde ein Konzept zur virtualisierten Bereitstellung von Diensten entwickelt und auf industrielle Stationsgeräte übertragen. Das daraus entstandene Artefakt, das Virtualisierungskonzept, basiert auf einer BS-Virtualisierung und nutzt Technologien zur Verwaltung der Systemressourcen eines Stationsgeräts. Mit Docker, Containerd und weiteren Technologien werden Dienste als isolierte Container instanziiert und auf dem Gerät verwaltet. Im Kapitel 6 wurde das Artefakt in einer Labor- und Feldtestumgebung erprobt und im Hinblick auf funktionale und

nicht-funktionale Anforderungen untersucht. Mit der Umsetzung des Virtualisierungskonzepts auf industrielle Stationsgeräte und in elf Ortsnetzstationen eines Verteilnetzbetreibers konnte die spezifische Forschungsfrage 1 erfolgreich beantwortet werden.

Spezifische FF2: *Welche Konzepte zur Verwaltung, Überwachung und Steuerung von virtualisierten Diensten lassen sich im Kontext von Edge-Computing für Stationsgeräte nutzen?*

Diese Forschungsfrage adressiert die Verwaltung von Diensten über die lokale Betrachtung einer einzelnen Station hinaus und bezieht sich auf die Verwaltung von Diensten in einem gesamten Netzgebiet. Aufbauend auf dem Virtualisierungskonzept wurde im Kapitel 4 ein weiteres Artefakt, das Verwaltungskonzept, entwickelt. Dieses Konzept basiert auf Paradigmen wie Edge-Computing und ermöglicht die zentrale oder dezentrale Verwaltung mehrerer Stationsgeräte. Umgesetzt wurde das Konzept durch eine umfassende Container-Orchestrierungssoftware namens Kubernetes, die es erlaubt, codebasierte Beschreibungen zu nutzen, um einzelne oder mehrere Dienste über Stationsgrenzen hinweg zu verteilen. Zusätzlich beinhaltet das Verwaltungskonzept die Überwachung und Steuerung der Dienste zur Laufzeit sowie die Möglichkeit, automatisierte Routinen auszulösen. Das Konzept wurde in einen standardisierten Engineeringprozess integriert und in verschiedenen Szenarien der Labor- und Felderprobung evaluiert. Es konnte gezeigt werden, dass alle funktionalen und nicht-funktionalen Anforderungen erfüllt wurden und das Konzept die Fragestellung erfolgreich beantwortet.

Spezifische FF3: *Wie lassen sich Virtualisierungs- und Verwaltungskonzepte in die standardisierten Systemautomatisierungsprozesse von Verteilnetzbetreibern integrieren?*

Die letzte Forschungsfrage ergänzt die technologisch getriebenen Konzepte um die Perspektive des Netzbetreibers und untersucht, wie diese Konzepte in die bisher standardisierten Systemautomatisierungsprozesse eingebettet werden können. Im Rahmen dieser Arbeit wurden die Engineeringprozesse nach IEC 61850 aufbereitet und um kontinuierliche Prozesse erweitert. Im Kapitel 5 wurde die vollständige Integration des Virtualisierungs- und Verwaltungskonzepts in den Prozessablauf nach IEC 61850 ergänzt, von der Ergänzung der Stationsbeschreibung bis hin zur kontinuierlichen Überwachung und Steuerung. Der Engineeringprozess wurde sowohl in der Laborevaluation als auch im Feldtestgebiet des Netzbetreibers umgesetzt und erprobt. Es konnte erfolgreich gezeigt werden, dass eine Integration der Konzepte in bestehende standardisierte Systemautomatisierungsprozesse möglich ist und die Forschungsfrage wurde erfolgreich beantwortet.

Primäre FF: *Wie muss ein Engineeringprozess für virtualisierte Dienste in digitalisierten Verteilnetzen entworfen werden?*

Aufbauend auf die entwickelten Konzepte und die Erweiterung des Engineeringprozesses nach IEC 61850 konnte erfolgreich gezeigt werden, wie ein Engineeringprozess für virtualisierte Dienste in digitalisierten Verteilnetzen aussehen kann.

7.2 Einschränkungen und weiterer Forschungsbedarf

Die Komplexität und der große Umfang eines Forschungsgebiets sind wesentliche Bestandteile einer Dissertation. Dazu gehört auch die gezielte Eingrenzung der Untersuchungen und der Forschungsfragen, um eine ausreichende Tiefe in einem Themenfeld zu erreichen. Neben einer Vielzahl potenziell interessanter Untersuchungen werden im Folgenden einige wichtige Einschränkungen dieser Dissertation und der daraus abzuleitende Forschungsbedarf beschrieben.

Echtzeitfähigkeit der Dienste
Wie bei der Analyse verwandter Arbeiten im Bezug auf Dienste der Stationsgeräte im Abschnitt 3.1.1 beschrieben, stellen einige Dienste besondere Anforderungen an die echtzeitfähige Ausführung. Die meisten dieser Dienste fallen unter die Kategorie der Schutzfunktionen und werden oft auf speziellen echtzeitfähigen Prozessoren ausgeführt. Im Rahmen dieser Arbeit wurden die Konzepte nur auf Prozessoren und Betriebssystemen ohne besondere Echtzeitanforderungen umgesetzt. In zukünftigen Arbeiten könnte untersucht werden, wie die Konzepte im Hinblick auf die besonderen Anforderungen der Echtzeitfähigkeit von Diensten ausgelegt werden müssen und ob die gewählten Technologien auch auf speziellen Betriebssystemen und CPUs ausführbar sind.

Standardisierung und Normung
Die Einbettung der Konzepte in die standardisierten Systemautomatisierungsprozesse nach IEC 61850 basierte unter anderem auf der Erweiterung der SCD-Datei. Diese Datei wurde im Spezifikationsfeld um die Kubernetes-spezifische Beschreibung erweitert. Auch wenn diese Beschreibung bereits an das SCL-Format angelehnt ist, fehlt eine entsprechende Normung der genauen Parameter und die Überführung in die Hauptbeschreibung der SCD-Datei. Aus den bisherigen Erkenntnissen besteht jedoch zunächst ein Forschungsbedarf in der standardisierten und interoperablen Beschreibung der Container-Orchestrierung, die aktuell noch abhängig von der jeweils genutzten Technologie ist. Im letzten Schritt müssten Technologien so weit angepasst

werden, dass sie Beschreibungen basierend auf dem IEC 61850 interpretieren und ausführen könnten.

Plug-and-Play Installation
In dieser Arbeit wurde gezeigt, dass Dienste einfach über codebasierte Beschreibungen automatisiert in einem Plug-and-Play-Ansatz auf Stationsgeräten bereitgestellt werden können. Wird der Plug-and-Play-Ansatz im Hinblick auf die Stationsgeräte, Sensoren, Netzwerke und andere Komponenten in einer Station untersucht, dann wurde in dieser Arbeit nur geprüft, ob eine Vorkonfiguration genutzt werden kann, um die Erreichbarkeit über einen Fernzugriff zu ermöglichen. Es bleibt also ein offener Forschungsbedarf, inwieweit sich Konzepte und Technologien finden lassen, die eine einfachere Form der Plug-and-Play-Installation ohne Vorkonfiguration in einer Station ermöglichen.

Integration in Engineeringwerkzeuge
Der entworfene Engineeringprozess aus Kapitel 5 erweitert bestehende Engineeringprozesse, die üblicherweise durch Engineeringwerkzeuge unterstützt werden. Diese Werkzeuge helfen einem Netzbetreiber bei der Konfiguration und Abwicklung der Prozessschritte. Auch wenn ein Großteil der vorgestellten Prozessschritte automatisiert durch das Verwaltungskonzept umgesetzt werden, sollten bestehende Engineeringwerkzeuge erweitert werden. Dies umfasst insbesondere die Berücksichtigung der erweiterten SCD-Beschreibung sowie die Einbindung der notwendigen Netzwerkkonfigurationen.

Bereitstellung der Dienste durch Hersteller
Die Bereitstellung von Diensten wurde bisher immer über spezifische Hardware mit vorkonfigurierten Diensten erfüllt. Die in dieser Arbeit umgesetzten Konzepte ermöglichen eine herstellerunabhängige Bereitstellung von Diensten auf ein Hardware-Gerät. Das bedeutet, es muss den Herstellern möglich sein, Dienste und passende ICD-Beschreibungen den Netzbetreibern zur Verfügung zu stellen, ohne darauf abgestimmte Hardware zu liefern. Die Umsetzung des aktuellen Konzepts nutzt eine lokale Instanz einer Docker-Registry als Repository zur Ablage von Diensten. Eine passende Forschungsfrage wäre, wie eine geeignete Infrastruktur zur Bereitstellung von Software-Instanzen in die Systemlandschaft integriert werden kann. Diese Forschungsfrage kann durch unterschiedliche Paradigmen wie Cloud-Computing oder DevOps-Strategien gelöst werden.

Wartbarkeit des Betriebssystems

Die Wartbarkeit einzelner Dienste als Container-Instanz wurde in dieser Arbeit detailliert aufgezeigt. Jedoch wurde nicht untersucht, wie sich das Betriebssystem

des Stationsgeräts oder darunterliegende BIOS- oder Boot-Systeme warten lassen. Dieser Prozess sollte sicherstellen, dass ein System im Fehlerfall auf den vorherigen Zustand zurückgesetzt werden kann. Es ist zudem wichtig, dass anschließend alle Eigenschaften der Fernwartbarkeit wieder verfügbar sind. Möglicherweise muss das Stationsgerät zusätzliche Hardware bereitstellen, um diese Prozesse ausführen zu können.

Aus der jetzigen Perspektive gibt es viele ungeklärte Forschungsfragen, Prozesse und Entwicklungen im Bereich eines vollständig automatisierten Stationsautomatisierungssystems, die in zukünftigen Arbeiten betrachtet werden können.

Abkürzungsverzeichnis

Akronyme

B

BS Betriebssystem.

C

CPES Cyber-Physischem Energiesystems.

CPU Central Processing Unit.

CRI Container Runtime Interface.

CVC Coordinated Voltage Control.

D

DER Distributed Energy Resource.

DERs Distributed Energy Resources.

DevOps Development and Operations.

DSRM Design Science Research Methodology.

E

E/A Ein- und Ausgabe.

G

GFV Grid Function Virtualization.

I

IaC Infrastructure as Code.

ICD IED Capability Description.

IED Intelligent Electronic Device.

IEDs Intelligent Electronic Devices.

IKT Informations- und Kommunikationstechnologie.

IoT Internet of Things.

IP Internet Protocol.

K

KPIs Key Performance Indicators.

L

LXC Linux Containers.

R

RAM Random Access Memory.

RPI Raspberry Pi.

RPIs Raspberry Pis.

RTU Remote Terminal Unit.

RTUs Remote Terminal Units.

S

SCD Substation Configuration Description.

SCL Substation Configuration Language.

SE State Estimation.

SGAM Smart Grid Architecture Model.

SSD System Specification Description.

V

VGF Virtual Grid Function.

VGFs Virtual Grid Functions.

VM Virtuelle Maschine.

VMs Virtuelle Maschinen.

Y

YAML YAML Ain't Markup Language.

Abbildungsverzeichnis

Tabellenverzeichnis

Skriptverzeichnis

Eigene Veröffentlichungen

[3] **Krüger, Carsten**, Marcel Otte, Stefanie Holly et al. “Redispatch 3.0–Congestion Management for German Power Grids–Considering Controllable Resources in Low-Voltage Grids”. In: *ETG Congress 2023*. VDE. 2023, S. 1–7 (zitiert auf den Seiten 1, 2).

[4] Anand Narayan, Batoul Hage Hassan, Shadi Attarha et al. “Grid function virtualization for reliable provision of services in cyber-physical energy systems”. In: *2020 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE. 2020, S. 1–5 (zitiert auf den Seiten 1, 2, 12, 31, 32, 57, 64, 151).

[7] Jorge Velasquez, **Krüger, Carsten**, Davood Babazadeh et al. “Flexible and reconfigurable automation architecture for electrical power systems”. In: *2019 IEEE Milan PowerTech*. IEEE. 2019, S. 1–6 (zitiert auf den Seiten 2, 12, 119).

[8] **Krüger, Carsten**, Anand Narayan, Felipe Castro et al. “Real-time test platform for enabling grid service virtualisation in cyber physical energy system”. In: *2020 25th IEEE international conference on emerging technologies and factory automation (ETFA)*. Bd. 1. IEEE. 2020, S. 109–116 (zitiert auf den Seiten 2, 12, 24, 59–64).

[11] **Krüger, Carsten**, Jirapa Kamsamrong und Sebastian Lehnhoff. “Virtualisation and management technologies of smart substations”. In: *CIRED* (2023) (zitiert auf den Seiten 3, 13, 93, 95).

[12] **Krüger, Carsten**, Shadi Attarha, Davood Babazadeh und Sebastian Lehnhoff. *i-Automate: Modular konfigurier- und prüfbare Automatisierungsarchitektur für zukünftige aktive elektrische Energienetze : Schlussbericht zum Forschungsprojekt : 6. Energieforschungsprogramm der Bundesregierung : Netze für die Stromversorgung der Zukunft*. Oldenburg, 2020 (zitiert auf den Seiten 6, 13, 27, 29, 48, 119).

[13] **Krüger, Carsten** und Sebastian Lehnhoff. *i-Autonomous: Standardisierung und Integration modular-autonomer Automatisierungskomponenten in neuartige, intelligente Ortsnetzstationen Teilvorhaben: Standardisiertes IKT-Monitoring und sicheres Systemupdaten von Automatisierungskomponenten in intelligenten Ortsnetzstationen*. Oldenburg, 2024 (zitiert auf den Seiten 6, 13, 26, 27, 29, 48, 92, 109, 119, 120, 132, 141, 142).

[17] Anand Narayan, **Krüger, Carsten**, Andre Goering et al. “Towards future SCADA systems for ICT-reliant energy systems”. In: *International ETG-Congress 2019; ETG Symposium*. VDE. 2019, S. 1–7 (zitiert auf den Seiten 12, 24).

[18] **Krüger, Carsten**, Jorge Velasquez, Davood Babazadeh und Sebastian Lehnhoff. “Flexible and reconfigurable data sharing for smart grid functions”. In: *Energy Informatics* 1 (2018), S. 361–366 (zitiert auf Seite 12).

[19] Shadi Attarha, Anand Narayan, Batoul Hage Hassan et al. “Virtualization management concept for flexible and fault-tolerant smart grid service provision”. In: *Energies* 13.9 (2020), S. 2196 (zitiert auf den Seiten 12, 24, 31–33, 57, 58, 64).

[20] S Raczka, B Bauernschmitt, D Hilbrich et al. “A novel software applications rollout and monitoring strategy for enabling the transition to electromobility in future smart grids”. In: *CIRED* (2022) (zitiert auf den Seiten 13, 91, 92).

[21] Sebastian Raczka, Frederik Puhe, **Krüger, Carsten**, Jan Arph und Christian Rehtanz. “Automated integration process of future automation and monitoring systems in distribution grids”. In: *ETG Congress 2023*. VDE. 2023, S. 1–7 (zitiert auf den Seiten 13, 92, 94).

Veröffentlichungen

[1] Karlo Hainsch, Konstantin Löffler, Thorsten Burandt et al. “Energy transition scenarios: What policies, societal attitudes, and technology developments will realize the EU Green Deal?” In: *Energy* 239 (2022), S. 122067 (zitiert auf Seite 1).

[2] Moses Jeremiah Barasa Kabeyi und Oludolapo Akanni Olanrewaju. “Sustainable Energy Transition for Renewable and Low Carbon Grid Electricity Generation and Supply”. In: *Frontiers in Energy Research* 9 (2022) (zitiert auf Seite 1).

[5] Theis B. Rasmussen, Guangya Yang, Arne H. Nielsen und Zhaoyang Dong. “A review of cyber-physical energy system security assessment”. In: *2017 IEEE Manchester PowerTech*. 2017, S. 1–6 (zitiert auf Seite 2).

[6] Xinghuo Yu und Yusheng Xue. “Smart Grids: A Cyber–Physical Systems Perspective”. In: *Proceedings of the IEEE* 104.5 (2016), S. 1058–1070 (zitiert auf Seite 2).

[9] DKE AK 952.0.1. *Description of the Engineering Process*. `https://www.dke.de/resource/blob/1622858/e1974896c7bb7d8b32cfbcb5e80386f9/e-ak952-0-1-engineering-process-v10-data.pdf`. [Accessed 04-01-2024] (zitiert auf den Seiten 3, 16, 30, 95, 96).

[10] Sunil Kumar Mohanty, Gopika Premsankar und Mario Di Francesco. “An evaluation of open source serverless computing frameworks”. English. In: *Proceedings - IEEE 10th International Conference on Cloud Computing Technology and Science, CloudCom 2018*. IEEE International Conference on Cloud Computing Technology and Science, CloudCom ; Conference date: 10-12-2018 Through 13-12-2018. United States: IEEE, Dez. 2018, S. 115–120 (zitiert auf Seite 3).

[14] *VDE definiert Autonomiestufen für Stromnetzbetrieb — vde.com*. `https://www.vde.com/de/etg/arbeitsgebiete/v2/vde-definiert-autonomiestufen-fuer-stromnetzbetrieb`. [Aufgerufen 25-06-2024] (zitiert auf Seite 7).

[15] Marten Bunnemann, Hendrik Paul und Florian Hintz. “Digitale Verteilnetze als Rückgrat der Energiewende”. In: *Digitale Daseinsvorsorge: Stadtwerke als Treiber der digitalen Transformation für Kommunen, Land und Bund*. Hrsg. von Jens Meier, Tobias Brosze, Ulf Papenfuß und Manuel Wiesche. Wiesbaden: Springer Fachmedien Wiesbaden, 2024, S. 319–335 (zitiert auf Seite 7).

[16] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger und S. Chatterjee. “A design science research methodology for information systems research”. In: *Journal of Management Information Systems* 24 (Jan. 2007), S. 45–77 (zitiert auf den Seiten 10, 11).

[22] DKE AK 952.0.1 TG3. *Anforderungen an IEC 61850 Engineeringwerkzeuge*. `https://www.dke.de/resource/blob/1622834/762050db66c0406098a48b08913bba2e/ak952-0-1-tg3-engineeringwerkzeuge-1-0-data.pdf`. [Accessed 04-01-2024] (zitiert auf den Seiten 16, 30, 90, 95, 105).

[23] Christian Baun. *Betriebssysteme kompakt*. Springer, 2017 (zitiert auf den Seiten 16, 17).

[24] Miguel Masmano, Ismael Ripoll, Alfons Crespo und J Metge. “Xtratum: a hypervisor for safety critical embedded systems”. In: *11th Real-Time Linux Workshop*. Citeseer. 2009, S. 263–272 (zitiert auf Seite 18).

[25] Stefan Reinheimer. *Cloud Computing: Die Infrastruktur der Digitalisierung*. Springer-Verlag, 2018 (zitiert auf Seite 19).

[26] Peter Mell, Tim Grance et al. “The NIST definition of cloud computing”. In: (2011) (zitiert auf Seite 19).

[27] Keyan Cao, Yefan Liu, Gongjie Meng und Qimeng Sun. “An Overview on Edge Computing Research”. In: *IEEE Access* 8 (2020), S. 85714–85728 (zitiert auf Seite 20).

[28] Z. Zhao, F. Liu, Z. Cai und N. Xiao. “Edge Computing: Platforms, Applications and Challenges”. In: *Jisuanji Yanjiu yu Fazhan/Computer Research and Development* 55 (Feb. 2018), S. 327–337 (zitiert auf Seite 20).

[29] Björn Bauernschmitt. “Modulare Smart-Grid-Automatisierungsarchitektur mit integriertem Konfigurationsprozess auf Basis der IEC 61850-6”. Diss. Dortmund, Technische Universität, 2023 (zitiert auf den Seiten 24–26, 119).

[30] Rajkumar Palaniappan, Björn Bauernschmitt, Dominik Hilbrich und Christian Rehtanz. “An intelligent measurement and control device for active distribution grids”. In: *2020 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)*. IEEE. 2020, S. 975–979 (zitiert auf Seite 25).

[31] Bundesamt für Sicherheit in der Informationstechnik. *IT-Grundschutz-Bausteine — bsi.bund.de*. https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/IT-Grundschutz-Kompendium/IT-Grundschutz-Bausteine/Bausteine_Download_Edition_node.html. [Aufgerufen 04-01-2024]. 2023 (zitiert auf Seite 30).

[32] Boas Betzler. "Virtualisierung". In: *IT: Technologien Lösungen Innovationen*. Hrsg. von Herbert Kircher. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, S. 100–110 (zitiert auf Seite 30).

[33] Docker inc. *Network drivers overview — docs.docker.com*. https://docs.docker.com/network/drivers/. [Accessed 27-01-2024]. 2024 (zitiert auf Seite 35).

[34] Docker inc. *Running containers — docs.docker.com*. https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities. [Accessed 27-01-2024] (zitiert auf den Seiten 35, 100).

[35] Michael Kerrisk. *capabilities(7) - Linux manual page — man7.org*. https://man7.org/linux/man-pages/man7/capabilities.7.html. [Accessed 27-01-2024]. 2024 (zitiert auf Seite 35).

[36] Ross D. Arnold und Jon P. Wade. "A Definition of Systems Thinking: A Systems Approach". In: *Procedia Computer Science* 44 (2015). 2015 Conference on Systems Engineering Research, S. 669–678 (zitiert auf Seite 36).

[37] Peter Marwedel. *Eingebettete Systeme: Grundlagen Eingebetteter Systeme in Cyber-Physikalischen Systemen*. Springer Nature, 2021 (zitiert auf Seite 36).

[38] Inc. GitHub. *GitHub - rkt/rkt: [Project ended] rkt is a pod-native container engine for Linux. It is composable, secure, and built on standards. — github.com*. https://github.com/rkt/rkt. [Accessed 05-02-2024] (zitiert auf Seite 40).

[39] Linux Containers. *Linux Containers — linuxcontainers.org*. https://linuxcontainers.org/. [Accessed 05-02-2024] (zitiert auf Seite 40).

[40] The Linux Foundation. *Open Container Initiative - Open Container Initiative — opencontainers.org*. https://opencontainers.org/. [Accessed 05-02-2024] (zitiert auf Seite 40).

[41] Inc. GitHub. *GitHub - opencontainers/runc: CLI tool for spawning and running containers according to the OCI specification — github.com*. https://github.com/opencontainers/runc. [Accessed 05-02-2024] (zitiert auf Seite 40).

[42] The Linux Foundation. *Cloud Native Computing Foundation — cncf.io*. https://www.cncf.io/. [Accessed 05-02-2024] (zitiert auf Seite 41).

[43] The Linux Foundation. *containerd — containerd.io*. https://containerd.io/. [Accessed 05-02-2024] (zitiert auf den Seiten 41, 44, 45).

[44] he Linux Foundation. *cri-o — cri-o.io*. https://cri-o.io/. [Accessed 05-02-2024] (zitiert auf Seite 41).

[45] *Docker overview — docs.docker.com*. https://docs.docker.com/get-started/overview/#docker-architecture. [Accessed 05-02-2024] (zitiert auf den Seiten 42, 46).

[46] Docker Inc. *Manage data in Docker — docs.docker.com*. https://docs.docker.com/storage/. [Aufgerufen 25-06-2024] (zitiert auf Seite 43).

[47] KoCoS Messtechnik AG. *KoCoS, Hersteller für Mess- und Prüftechnik*. https://www.kocos.com/de/startseite. [Accessed 17-02-2024] (zitiert auf den Seiten 49, 50).

[48] Docker Inc. *docker — docs.docker.com*. https://docs.docker.com/reference/cli/docker/. [Accessed 02-03-2024] (zitiert auf Seite 53).

[49] Rajkumar Palaniappan, Dominik Hilbrich, Björn Bauernschmitt und Christian Rehtanz. "Co-ordinated voltage regulation using distributed measurement acquisition devices with a real-time model of the Cigré low-voltage benchmark grid". In: *IET Generation, Transmission & Distribution* 13.5 (2019), S. 710–716 (zitiert auf Seite 62).

[50] Michele Guerriero, Martin Garriga, Damian A Tamburri und Fabio Palomba. "Adoption, support, and challenges of infrastructure-as-code: Insights from industry". In: *2019 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE. 2019, S. 580–589 (zitiert auf den Seiten 64–66, 135).

[51] Shapna Muralidharan, Gyuwon Song und Heedong Ko. "Monitoring and managing iot applications in smart cities using kubernetes". In: *Cloud Computing* 11 (2019) (zitiert auf den Seiten 65, 66).

[52] The Kubernetes Authors. *Production-Grade Container Orchestration — kubernetes.io*. https://kubernetes.io/. [Aufgerufen 13-03-2024] (zitiert auf Seite 76).

[53] The Kubernetes Authors. *Kubernetes API Reference Docs — kubernetes.io*. https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.30/. [Aufgerufen 26-06-2024] (zitiert auf Seite 82).

[54] Marion Gottschalk, Mathias Uslar und Christina Delfs. *The use case and smart grid architecture model approach: the IEC 62559-2 use case template and the SGAM applied in various domains*. Springer, 2017 (zitiert auf den Seiten 85, 86).

[55] Weit e.V. *Das deutsche Referenzmodell für Systementwicklungsprojekte*. https://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.3/V-Modell-XT-Gesamt.pdf. [Aufgerufen 25-03-2024] (zitiert auf Seite 87).

[56] ArbeitsKreis 952.0.1. *dke.de*. https://www.dke.de/resource/blob/1622858/e1974896c7bb7d8b32cfbcb5e80386f9/e-ak952-0-1-engineering-process-v10-data.pdf. [Aufgerufen 14-03-2024] (zitiert auf den Seiten 87, 88, 104).

[57] ArbeitsKreis 952.0.1. *dke.de*. https://www.dke.de/resource/blob/1622834/762050db66c0406098a48b08913bba2e/ak952-0-1-tg3-engineeringwerkzeuge-1-0-data.pdf. [Aufgerufen 27-03-2024] (zitiert auf Seite 87).

[58] he Linux Foundation. *OpenSCD — openscd.github.io*. https://openscd.github.io/. [Aufgerufen 02-06-2024] (zitiert auf Seite 104).

[59] Inc. GitHub. *GitHub - flannel-io/flannel: flannel is a network fabric for containers, designed for Kubernetes — github.com*. https://github.com/flannel-io/flannel. [Aufgerufen 11-06-2024] (zitiert auf Seite 120).

[60] *GitHub - kubernetes-sigs/metrics-server: Scalable and efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines. — github.com*. https://github.com/kubernetes-sigs/metrics-server. [Aufgerufen 11-06-2024] (zitiert auf Seite 120).

[61] *CFS Scheduler 2014; The Linux Kernal Documentation — docs.kernel.org*. https://docs.kernel.org/scheduler/sched-design-CFS.html. [Aufgerufen 23-06-2024] (zitiert auf Seite 129).

[62] *Forschungsberichte - TIB — tib.eu*. https://www.tib.eu/de/publizieren-archivieren/forschungsberichte. [Aufgerufen 24-06-2024] (zitiert auf Seite 142).

[63] *Considerations for large clusters — kubernetes.io*. https://kubernetes.io/docs/setup/best-practices/cluster-large/. [Aufgerufen 24-06-2024] (zitiert auf Seite 147).

Anhang

A Ergänzungen zum Engineeringprozess

A.1 Python Programm zur Umwandlung von ICD-Dateien

```
1
2  import re
3  import sys
4  import xml.etree.ElementTree as ET
5
6
7  def convertYaml(pathToXML: str, yamlOutputPath: str) -> str:
8      """
9      Converts an XML to a yaml
10     :param pathToXML:
11     :param yamlOutputPath:
12     :return: Metadataname of generated yaml
13     """
14     print(pathToXML)
15
16     toBeParsedXMLFile = ET.parse(str(pathToXML))
17     root = toBeParsedXMLFile.getroot()
18
19     # find uri
20     uri: str = ""
21     if root.tag[0] == "{":
22         uri, ignore, tag = root.tag[1:].partition("}")
23         uri = "{" + uri + "}"
24     else:
25         uri = ""
26         tag = root.tag
27
28     for ied in root.findall(uri + "IED"):
29         for private in ied.findall(uri + "Private"):
30             for docker in private.findall(uri + "Docker"):
```

```
                for kubeconfig in docker:
                    try:
                        metadata = kubeconfig.find(uri + "
                            ↪ metadata")
                        metaDataName: str = metadata.attrib.
                            ↪ get("name")
                        labelName: str = metadata.find(uri +
                            ↪ "labels").attrib.get("app")

                        spec = kubeconfig.find(uri + "Spec")

                        containers = spec.find(uri + "
                            ↪ containers")

                        containerName: str = containers.
                            ↪ attrib.get("name")
                        imageName: str = containers.attrib.
                            ↪ get("image")

                        envStr: str = ""
                        # check if environment is given
                        try:
                            env = containers.find(uri + "env"
                                ↪ )
                            envStr = """\n      env:"""

                            for child in env:
                                envName: str = child.attrib.
                                    ↪ get("key")
                                envValue: str = child.attrib.
                                    ↪ get("value")
                                childStr: str = """\n
                                    ↪ - name: """ + envName +
                                    ↪  """
            value: """ + envValue + ""
                                envStr += childStr
                        except AttributeError:
                            pass

                        # check if security Context is given
                        securityContextStr: str = ""
                        try:
                            securityContext = containers.find
                                ↪ (uri + "securityContext")
```

```
63                              capabilitiesAsString =
                                    ↪ securityContext.find(uri +
                                    ↪ "capabilities").attrib.get(
                                    ↪ "add")
64                              capabilities =
                                    ↪ capabilitiesAsString.split(
                                    ↪ sep=", ")
65                              securityContextStr = """\n
                                    ↪ securityContext:
66                  capabilities:
67                    add: ["""
68                              for capability in capabilities:
69                                  securityContextStr += "\"" +
                                        ↪ capability + "\""
70                                  if capability == capabilities
                                        ↪ [-1]:
71                                      securityContextStr += ",
                                            ↪ "
72                              securityContextStr += "]"
73                          except AttributeError:
74                              pass
75
76                          mountPath: str = containers.find(uri
                                ↪ + "VolumeMounts").find(uri + "
                                ↪ mountPath").attrib.get(
77                               "path")
78                          nameOfMountPath: str = containers.
                                ↪ find(uri + "VolumeMounts").find
                                ↪ (uri + "mountPath").attrib.get(
79                               "name")
80
81                          # resources
82                          resourcesStr: str = ""
83                          try:
84                              resources = containers.find(uri +
                                    ↪  "resources")
85                              requests = resources.find(uri + "
                                    ↪ requests")
86                              requestMemory: str = requests.
                                    ↪ attrib.get("memory")
87                              requestCpu: str = requests.attrib
                                    ↪ .get("cpu")
88                              limits = resources.find(uri + "
                                    ↪ limits")
```

```
                            limitMemory = limits.attrib.get("
                                ↪ memory")
                            limitCpu = limits.attrib.get("cpu
                                ↪ ")
                            resourcesStr = """\n
                                ↪ resources:
            requests:
              memory: \""" + requestMemory + """\"
              cpu: \""" + requestCpu + """\"
            limits:
              memory: \""" + limitMemory + """\"
              cpu: \""" + limitCpu + """\"""
                        except AttributeError:
                            pass

                        containerPortStr: str = ""
                        try:
                            containerPort = containers.find(
                                ↪ uri + "ports").attrib.get("
                                ↪ containerPort")
                            containerPortStr = """\n
                                ↪ ports:
        - containerPort: """ + containerPort
                        except AttributeError:
                            pass

                        nodeName: str = spec.find(uri + "
                            ↪ NodeName").attrib.get("name")
                        pullSecret: str = spec.find(uri + "
                            ↪ imagePullSecrets").attrib.get("
                            ↪ name")

                        volume: str = ""
                        try:
                            volume: str = """\n  volumes:
    - name: """
                            volume += spec.find(uri + "
                                ↪ Volumes").attrib.get("name
                                ↪ ") + "\n"
                            volume += """      emptyDir: """
                            volume += spec.find(uri + "
                                ↪ Volumes").attrib.get("
                                ↪ emptyDir")
                        except AttributeError:
```

```
                            pass
                        # print(kubeconfig.tag)
                        yamlString: str = """---
    apiVersion: v1
    kind: Pod
    metadata:
      name: """ + metaDataName + """
      labels:
        app: """ + labelName + """
    spec:
      containers:
        - name: """ + containerName + """
          image: """ + imageName + \
                                            envStr + \
                                            securityContextStr
                                               ↪ + """
          volumeMounts:
          -  mountPath: """ + mountPath + """
             name: """ + nameOfMountPath + \
                                            resourcesStr + \
                                            containerPortStr +
                                               ↪ """
      nodeName: """ + nodeName + """
      imagePullSecrets:
        - name: """ + pullSecret + """
      hostNetwork: true""" + volume
                    except AttributeError as error:
                        print(
                            "Kubeconfig of Metadataname \"" +
                               ↪  metaDataName + "\" is
                               ↪ missing critical Attribute
                               ↪ " + error.name + " and will
                               ↪  is being skipped")
                        continue  # skip this loop iteration

                    print(yamlString)
                    finalYaml = open(yamlOutputPath + "/" +
                       ↪ metaDataName + ".yaml", "w")
                    finalYaml.write(yamlString)
                    finalYaml.close()
                    return metaDataName
```

Skript A.1: Beispiel Python Code zur Umwandlung einer ICD-Datein in ein Kubernetes YAML.File