Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Evaluierung und Implementierung einer Verwaltungsschale für Industrie 4.0 Komponenten

Christian Schierle

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Christian Weber, M.Sc., Dipl.-Inf. Frank Steimle, Dipl.-Inf. Pascal Hirmer
Beginn am:	16. Januar 2017
Beendet am:	17. Juli 2017
CR-Nummer:	H.5.3, H.5.m, I.2.1, J.1

Kurzfassung

Aufgrund der enormen Anforderungen an die Flexibilität im produzierenden Gewerbe, steht die Industrie vor neuen Herausforderungen. Möglichst viele Bereiche des Marktes sollen weitläufig bedient werden. Gleichzeitig steht die Umsetzung von Kundenwünschen in der Produktion stark im Vordergrund. Die Vision Industrie 4.0 greift die neuen Anforderungen auf und sucht nach Lösungen, um Wertschöpfungsketten entsprechend zu verbessern. Diese Arbeit liefert einen Einblick in das Themengebiet Industrie 4.0 und die dabei zentralen Technologien. Der Fokus liegt hierbei auf dem Begriff der Verwaltungsschalen. Es wird deren Potential als Datenschnittstelle zwischen physischen Objekten und der Informationswelt untersucht. Im Zuge dessen bietet diese Arbeit auch eine kurze Einführung in OPC Unified Architecture. Ziel der Arbeit ist es, einen Überblick über die Anforderungen im Zusammenhang mit Industrie 4.0, vorhandene Technologien und deren Möglichkeiten zu bieten. Dazu wird eine Beispielimplementierung für Verwaltungsschalen anhand einiger Anforderungen analysiert. Anschließend werden vorbereitende Schritte für die Umsetzung einer eigenen Implementierung erläutert. Diese soll unter einigen Vereinfachungen ebenfalls den Anforderungen entsprechen.

Inhaltsverzeichnis

1. Einleitung **15**

2. Verwandte Arbeiten und theoretische Grundlagen **21**

3. Anforderungen **29**
3.1. Fachliche Anforderungen 29
3.2. Methodische Anforderungen 30
3.3. Technische Anforderungen 30
3.4. Allgemeine Anforderungen an eine Verwaltungsschale 31

4. Evaluierung der open Asset Administration Shell **33**
4.1. Grundlegende Projektstruktur 33
4.2. Zur Analyse verwendete Werkzeuge 35
4.3. Architektur des OPC UA Servers 36
4.4. Beschreibung des Server Codes 41
4.5. Datenzugriff über Endpunkte des OPC UA Servers 45
4.6. Umsetzung der Anforderungen an Verwaltungsschalen laut Plattform I4.0 in der openAAS Demo 46

5. Implementierung **49**
5.1. Anwendungsszenario 49
5.2. Definition eines Informationsmodells 50
5.3. Verfügbare Technologien 53

6. Zusammenfassung und Ausblick **59**

A. Übersicht über Objekte, Variablen und Werte aus der openAAS-Demo **61**

B. Setup eines MQTT Brokers mit ActiveMQ **65**

Literaturverzeichnis **67**

Abbildungsverzeichnis

1.1. Dezentrale Dienste als flexible Alternative zur klassischen Automatisierungspyramide . . . 16
1.2. Kommunikation mittels OPC UA im Fabrikumfeld . . . 18

2.1. RAMI4.0 . . . 22
2.2. I4.0 Komponente als Kombination aus Asset und Verwaltungsschale . . . 24
2.3. Modell einer SPS als I4.0 Komponente . . . 26

4.1. UML-Modell der openAAS . . . 34
4.2. Verschachtelung der openAAS-Verwaltungsschalen . . . 37
4.3. Property Statement Model . . . 39
4.4. Beispiele für Property Value Statements . . . 39
4.5. Allgemeine Struktur von Knoten im OPC UA Server der openAAS-Demo entsprechend der zugehörigen XML-Dateien . . . 40

5.1. Informationsmodell für die Implementierung einer Verwaltungsschale entsprechend des Anwendungsszenarios aus Abschnitt 5.1 . . . 52

A.1. Eigenschaften und Stammdaten des Multi Sensors als übergeordnete Verwaltungsschale . . . 61
A.2. Teilmodelle des Feuchtigkeitssensors . . . 62
A.3. Teilmodelle des Temperatursensors . . . 63
A.4. Teilmodelle der LED . . . 64

Tabellenverzeichnis

2.1. Basis-Sichten der Verwaltungsschale 24

3.1. Grundsätzliche Anforderungen an die Verwaltungsschale nach DIN SPEC 91345 30

4.1. Methoden im Server-Code und ihre Aufgaben 42
4.2. In der Demo-Implementierung umgesetzte Anforderungen 47

Verzeichnis der Listings

4.1. Definition eines Typs zur Speicherung von Messwerten 42
4.2. Ausschnitt aus der *main*-Methode der openAAS Demo 43
4.3. Initialisierung des Knotens mit der Message-Komponente 44
4.4. Zuordnung von Namespaces zur *AssetShell*-Applikation 44
4.5. Überprüfung der zum Umschalten der LED übergebenen Nachricht 46
4.6. Überprüfung der übergebenen Sender- und Empfängeradresse vor dem Umschalten der LED 48

5.1. Aufruf in Python OPC UA zum Einbinden von Knotendefinitionen aus einer XML-Datei 56

B.1. Kommandozeilenbefehle zum Start von ActiveMQ 65

Abkürzungsverzeichnis

BMBF Bundesministerium für Bildung und Forschung. 21

BMWi Bundesministerium für Wirtschaft und Energie. 21

CPS Cyber-physische Systeme. 15

I4.0 Industrie 4.0. 15

IIC Industrial Internet Consortium. 21

IPVS Institut für Parallele und Verteilte Systeme. 30

M2M Maschine-zu-Maschine. 16

MQTT Message Queue Telemetry Transport. 16

OPC UA OPC Unified Architecture. 16

openAAS open Asset Administration Shell. 33

QoS Quality of Service. 23

RAMI4.0 Referenzarchitekturmodell Industrie 4.0. 22

RWTH Aachen Rheinisch-Westfälische Technische Hochschule Aachen. 19

SDK Software Development Kit. 36

SOA serviceorientierte Architektur. 16

SPS speicherprogrammierbare Steuerung. 26

TCP Transmission Control Protocol. 41

TSN Time-sensitive Networking. 27

UML Unified Modeling Language. 33

URI Uniform Resource Identifier. 37

VDE Verband der Elektrotechnik Elektronik und Informationstechnik e.V.. 21

XDK Cross Domain Development Kit. 30

XML Extensible Markup Language. 35

ZVEI Zentralverband Elektrotechnik- und Elektronikindustrie e. V.. 21

1. Einleitung

Digitalisierung ist heutzutage allgegenwärtig und damit steht auch die Industrie vor neuen Aufgaben. Jegliche Art von Gegenstand kann durch Nachrüsten entsprechender Hardware ein gewisses Maß an Intelligenz erhalten. Fügt man einen Prozessor oder Mikrocontroller sowie Speicherkapazität hinzu, können Informationen (z. B. Sensordaten) verwaltet und verarbeitet werden. Produkte haben die Möglichkeit, Daten über ihre Nutzung abzuspeichern und zur Analyse bereitzustellen [MF10]. Dies eröffnet neue Möglichkeiten für IT-Architekturen in Produktionsumgebungen. Smarte Gegenstände, Maschinen oder Produkte benötigen daher kein übergeordnetes IT-System, sondern können selbst Auskunft über sich und ihren aktuellen Zustand geben. Durch aktive Kommunikation untereinander bildet eine Menge an smarten Gegenständen ein intelligentes Netzwerk.

„Der hohe Vernetzungsgrad und die allgegenwärtige Verfügbarkeit von Daten und Diensten lässt für die Automation neue und zukunftsträchtige Perspektiven entstehen. Unter anderem ergibt sich die Vision von adaptiven, sich selbst konfigurierenden und teilweise selbstorganisierenden, flexiblen Produktionsanlagen“ [VDI13, S. 3]. Cyber-physische Systeme (CPS) spielen dabei eine große Rolle: Durch die Kombination von physischen Objekten (z. B. Sensoren und Aktoren) mit informationsverarbeitenden Softwarekomponenten und deren Verbindung über Netzwerke, ergeben sich weitreichende Lösungsansätze in Firmennetzwerken [Gei+11]. CPS können so z. B. selbst Ereignisse melden und auch selbstständig auf Ereignismeldungen anderer Gegenstände reagieren. Daraus ergibt sich die Möglichkeit eines ereignisgesteuerten hochflexiblen Produktionsablaufs, der zur Wandlungsfähigkeit von Fabriken beiträgt. So werden, wie in Abbildung 1.1 zu sehen, die Hierarchieebenen der klassischen Automatisierungspyramide aufgebrochen und die Aufgabe von der Steuerung bis hin zur Unternehmensleitung auf ein gemeinsames Netzwerk verteilt. Wie Bauernhansl [Bau14] beschreibt, herrscht heutzutage beispielsweise in der Automobilindustrie eine enorme Vielfalt an verfügbaren Modellen mit nahezu unbeschränkten Möglichkeiten zur Personalisierung, wodurch nur verhältnismäßig wenige Stückzahlen pro Modell und Variante nachgefragt werden. Die Anforderungen an die Dynamik eines Unternehmens sind also sehr hoch. Durch den Einsatz smarter Objekte und hochgradiger Vernetzung in Fabrikumgebungen ist die Individualisierung von Produkten entsprechend spezieller Kundenwünsche möglich, ohne dabei auf Mechanismen der Massenproduktion verzichten zu müssen. Unter dem Leitwort Industrie 4.0 (I4.0) werden entsprechende Konzepte für moderne Produktionsabläufe entworfen, die sich Entwicklungen aus den Bereichen Datenspeicherung und -analyse sowie der Kommunikation in Echtzeit zunutze machen.

Die tatsächliche Umsetzung dieser Konzepte gestaltet sich derzeit jedoch schwierig, da bestehende Strukturen den neuen Anforderungen nicht gerecht werden. Blank et al. [Bla+16] machen

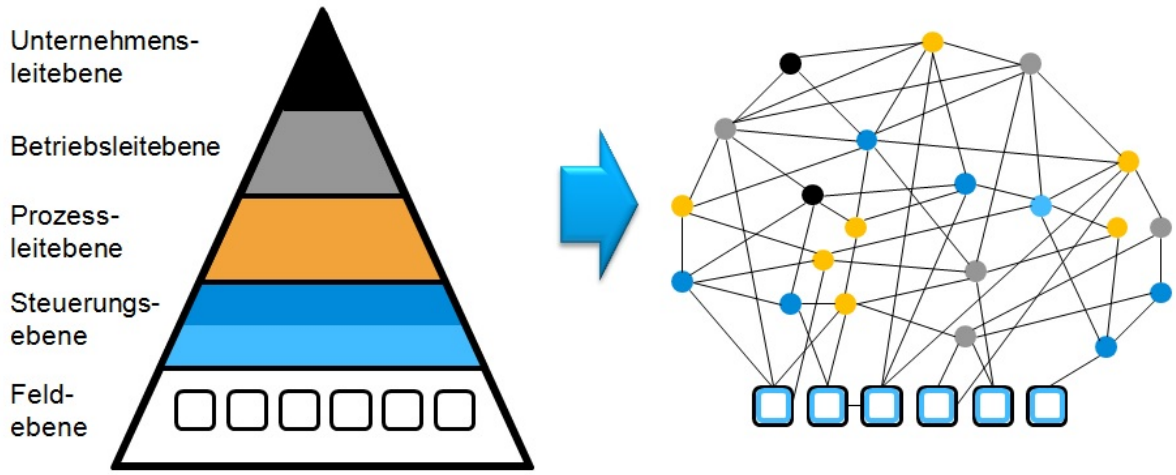

Abbildung 1.1.: Dezentrale Dienste als flexible Alternative zur klassischen Automatisierungspyramide [VDI13]

dafür veraltete und von Produzent zu Produzent verschiedene Kommunikationssysteme verantwortlich, die durch hohen Integrationsaufwand für zeitlichen und finanziellen Mehraufwand sorgen. Zudem verfügen sie aufgrund von eingeschränkten Konfigurationsmöglichkeiten nicht über den benötigten Funktionsumfang für digitale Fabriken: Die Verwirklichung von Industrie 4.0 erfordert ein hohes Maß an Informationsaustausch zwischen allen Bestandteilen einer Fabrik. Ein digitales Abbild jedes involvierten Gegenstandes, das dessen aktuellen Zustand, auszuführende Aufgaben und den vorgegebenen Endzustand beinhaltet, soll den Weg für Entwicklungen auf diesem Gebiet bereiten.

An dieser Stelle sind technische Gegenstände gemeint, die für eine Organisation einen bestimmten Wert besitzen. Sie werden auch Assets genannt und verkörpern sowohl konkrete Dinge, wie Maschinen oder Produkte, als auch abstrakte Dinge, wie z. B. eine Produktidee [DIN16]. Solche Assets können durch die Verbindung mit einer Verwaltungsschale zu einer I4.0 Komponente zusammengefasst werden. Wird dieser Vorgang entlang des kompletten Lebenszyklus eines Produktes durchgeführt, also von der Idee bis zu dessen Betrieb und Wartung, entsteht dadurch ein vollständiges virtuelles Abbild aller zum Produkt gehörigen Daten. Dieser sogenannte digitale Zwilling (oder auch Digital Twin) kann beispielsweise zur Analyse und Fehlererkennung herangezogen werden. Canedo [Can16] beschreibt z. B. die Vorhersage von Nutzungszeiten des Fahrzeuges eines Autofahrers anhand dessen üblichen Fahrgewohnheiten, um ein passendes Zeitfenster für ein Softwareupdate zu finden. Übertragen in ein Fabrikumfeld, können beispielsweise bei der Identifizierung fehlerhafter Produkte präventive Anpassungen in der Produktion vorgenommen werden, um den Wartungsaufwand oder gar verursachte Schäden zu minimieren.

Zur Vernetzung von Objekten insbesondere bei der Maschine-zu-Maschine (M2M) Kommunikation wird häufig Message Queue Telemetry Transport (MQTT) eingesetzt. Hierbei handelt

es sich um ein Netzwerkprotokoll, das den Austausch von Telemetrie-Daten über ein Publish-/Subscribe-Modell abhandelt. Informationsquellen senden Nachrichten an einen Sammelpunkt, den sogenannten „Broker“, wo diese in Warteschlangen („Queues“) eingereiht werden. Dabei wird eine Nachricht an alle Clients (Subscriber) weitergeleitet, die sich für die entsprechende Queue registriert haben. In der Industrie hat sich außerdem OPC Unified Architecture (OPC UA) etabliert. Es handelt sich hierbei um eine serviceorientierte Architektur (SOA), deren Teilnehmer nach dem Client-Server-Prinzip miteinander kommunizieren und ausführbare Dienste bereitstellen bzw. ausführen. Der Datenaustausch erfolgt im Gegensatz zum Publish-/Subscribe-Modell über eine Direktverbindung zwischen Server und Client ohne dazwischen geschaltete Vermittlungskomponenten. Eine detailliertere Einführung in OPC UA findet sich in einem entsprechenden Abschnitt am Ende dieses Kapitels. Abbildung 1.2 zeigt im Kontext einer Fabrik, wie einzelne Objekte bei der Kommunikation mittels OPC UA sowohl als Server, der Informationen und Dienste eines Objektes verwaltet und zugänglich macht, als auch als Client, der auf Informationen und Dienste anderer Objekte zugreift, agieren kann. Der zusätzliche Funktionsumfang hat aber auch eine größere Mächtigkeit zufolge. Für „Praktiker“ und Kleinanwender gestaltet sich die Verwendung von OPC UA deshalb häufig als zu komplex und sie verwenden stattdessen das leichtere MQTT [Spi07].

Die Eigenschaften der Verwaltungsschale konzentrieren sich vor allem auf die Fähigkeit zur Kommunikation zwischen Assets. Durch den Austausch von Informationen über deren Aufbau und Konfiguration sollen Produktionsabläufe schneller geplant, eingerichtet und damit eine Plug-and-Produce-Umgebung geschaffen werden, in der „sich [Maschinen] selbst identifizieren und in das Netzwerk einbinden können und so die erforderlichen Modifikationen an den Linien schneller und effizienter machen“ [Sie11]. Dadurch entsteht ein global verfügbares digitales Abbild der kompletten Fabrikumgebung, das deutliche Verbesserungen bei der Überwachung und Analyse von Prozessabläufen ermöglicht. Außerdem können beispielsweise Planungs oder Entwicklungsabteilungen, selbst standortübergreifend Einblicke in Produktionsabläufe erhalten und dadurch, wie von Kaeser [Kae15] beschrieben, Produkte schneller von der Idee zur Markteinführung bringen, aufkommende Markttrends flexibel aufnehmen und dabei den Einsatz von Energie und Ressourcen effizient gestalten.

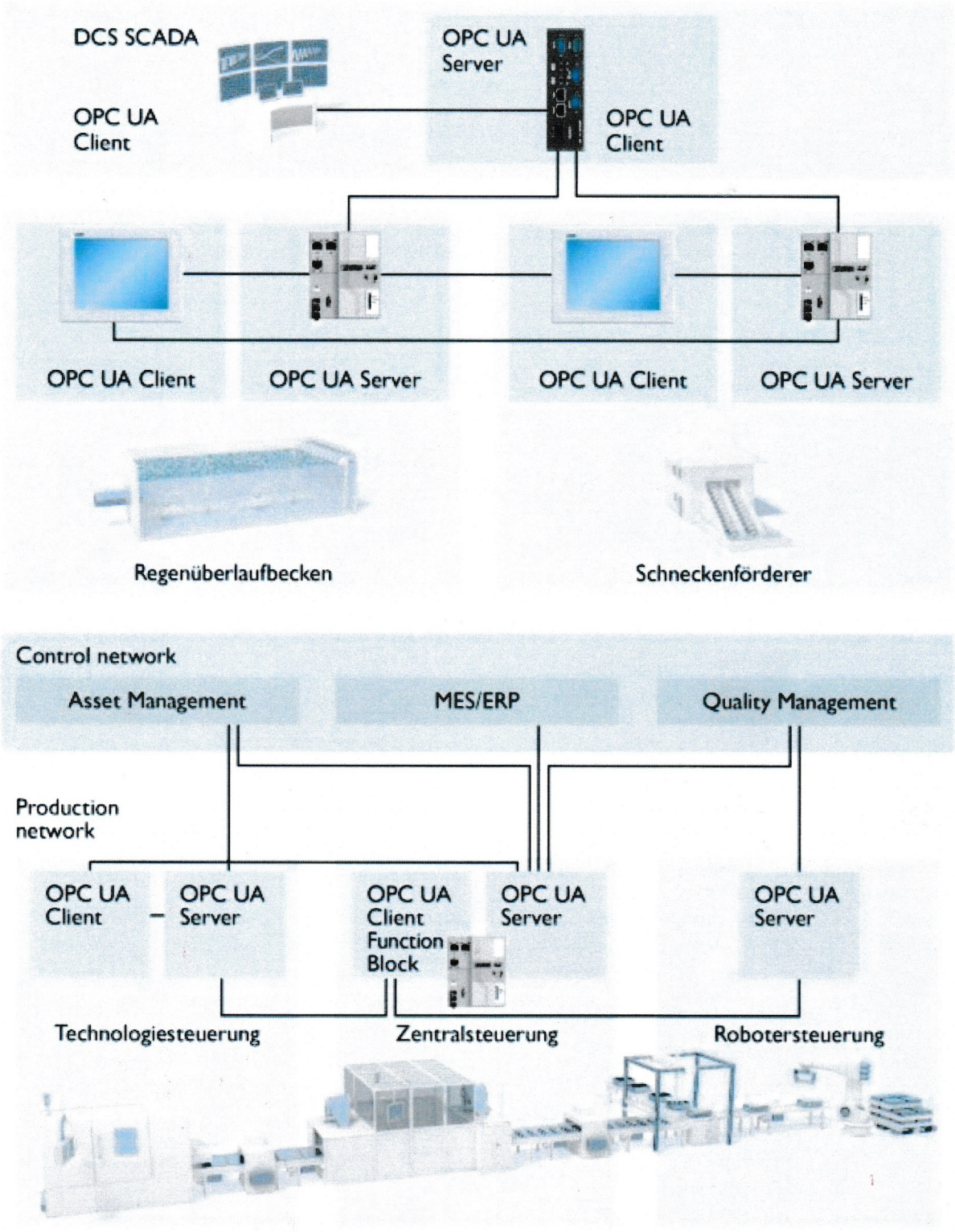

Abbildung 1.2.: Kommunikation mittels OPC UA im Fabrikumfeld

Aufbau der Arbeit

Diese Arbeit ist wie folgt gegliedert:

Kapitel 2 – Verwandte Arbeiten und theoretische Grundlagen: Diese Kapitel bietet eine Einführung in das Themengebiet Industrie 4.0 und führt die Begrifflichkeiten der Verwaltungsschale und des digitalen Zwillings ein. Als zentrale Technologie zur Umsetzung der Vision Industrie 4.0 wird die OPC UA kurz vorgestellt. Außerdem erfolgt die Betrachtung zweier Hardware-Plattformen, die für die Implementierung der Verwaltungsschale in Frage kommen.

Kapitel 3 – Anforderungen umfasst sowohl technische Vorgaben und Einschränkungen als auch die gewählten Arbeits- und Forschungsmethoden.

Kapitel 4 – Evaluierung der open Asset Administration Shell beinhaltet Ausführungen zur Untersuchung der Open Source Implementierung openAAS, wobei deren Aufbau, Funktionsumfang und Verwendung beschrieben werden. Die Rheinisch-Westfälische Technische Hochschule Aachen (RWTH Aachen) als Entwickler des Projekts hat dazu ein Repository auf GitHub erstellt, welches im Zuge dessen untersucht wird.

Kapitel 5 – Implementierung: Als Grundlage für den Implementierungsteil dieser Arbeit wird in diesem Kapitel ein Informationsmodell entworfen. Es bildet die Basis für die Umsetzung einer Verwaltungsschale mit Hilfe von OPC UA. Des Weiteren erfolgt die Auswahl einer Open Source Bibliothek, die bei der Entwicklung den OPC UA Server bereitstellen soll.

Kapitel 6 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit kurz zusammen und stellt Anknüpfungspunkte vor.

2. Verwandte Arbeiten und theoretische Grundlagen

Die zunehmende Informatisierung der Produktion im Lauf der letzten Jahre wird als Beginn einer neuen industriellen Revolution angesehen. Dabei bezeichnet Informatisierung „die Nutzung von Informationen aus unterschiedlichen Quellen, um hieraus weitere Informationen für Entscheidungen im Fertigungsbereich zu generieren“ [MSH16, S. 11]. Nach der Mechanisierung durch den Einsatz von Dampfmaschinen, Massenproduktion mit Hilfe von Fließbändern, sowie der Automatisierung durch Elektronik und IT steht nun also der nächste große Umbruch im produzierenden Gewerbe an. Dessen Ziel ist der Weg zur intelligenten Fabrik oder auch „Smart Factory“. Das bedeutet Interaktion und Kooperation zwischen Maschinen und auch mit dem Menschen auf der Fertigungsebene. Außerdem sollen intelligente Maschinen auch selbstständig auf die Anforderungen des vorliegenden Produktionsprozesses eingehen und beispielsweise beim Ausfall einer Produktionslinie automatisch durch entsprechende abfangende Anpassungen reagieren. Die Kernaspekte von Industrie 4.0 sind also einerseits das Sammeln, Speichern und Verarbeiten von Informationen über den Fertigungsablauf sowie andererseits deren Kommunikation unter den beteiligten technischen oder menschlichen Akteuren.

Da Industrie 4.0 noch ein recht junges Themenfeld ist, wird in vielen Bereichen noch Grundlagenforschung betrieben und erst nach und nach die Standardisierung vorangebracht. Auf nationaler Ebene sind als Vorreiter sowohl der Zentralverband Elektrotechnik- und Elektronikindustrie e. V. (ZVEI) als auch der Verband der Elektrotechnik Elektronik und Informationstechnik e.V. (VDE) zu nennen. Seit 2013 arbeiten diese und einige weitere Verbände in der Plattform Industrie 4.0[1] geleitet vom Bundesministerium für Wirtschaft und Energie (BMWi) und Bundesministerium für Bildung und Forschung (BMBF) gemeinsam an neuen Ideen, Forschungsprojekten und Standards rund um das Thema Industrie 4.0. Die Plattform führt laut Banthien und Senff [BS16] Politik, Wirtschaft, Wissenschaft und Gewerkschaften unter einem Dach zusammen, damit bei der Findung neuer Lösungsansätze die Anforderungen und Bedürfnisse möglichst aller Beteiligter aufgegriffen und berücksichtigt werden können. Weltweit forschen internationale Gruppierungen an neuen Methoden für intelligente Fabrikumgebungen. Die im asiatischen Raum agierende Industrial Value-Chain Initiative[2] und das Industrial Internet Consortium (IIC)[3], welches 2014 von den amerikanischen Unternehmen

[1] http://www.plattform-i40.de
[2] https://www.iv-i.org/
[3] http://www.iiconsortium.org/

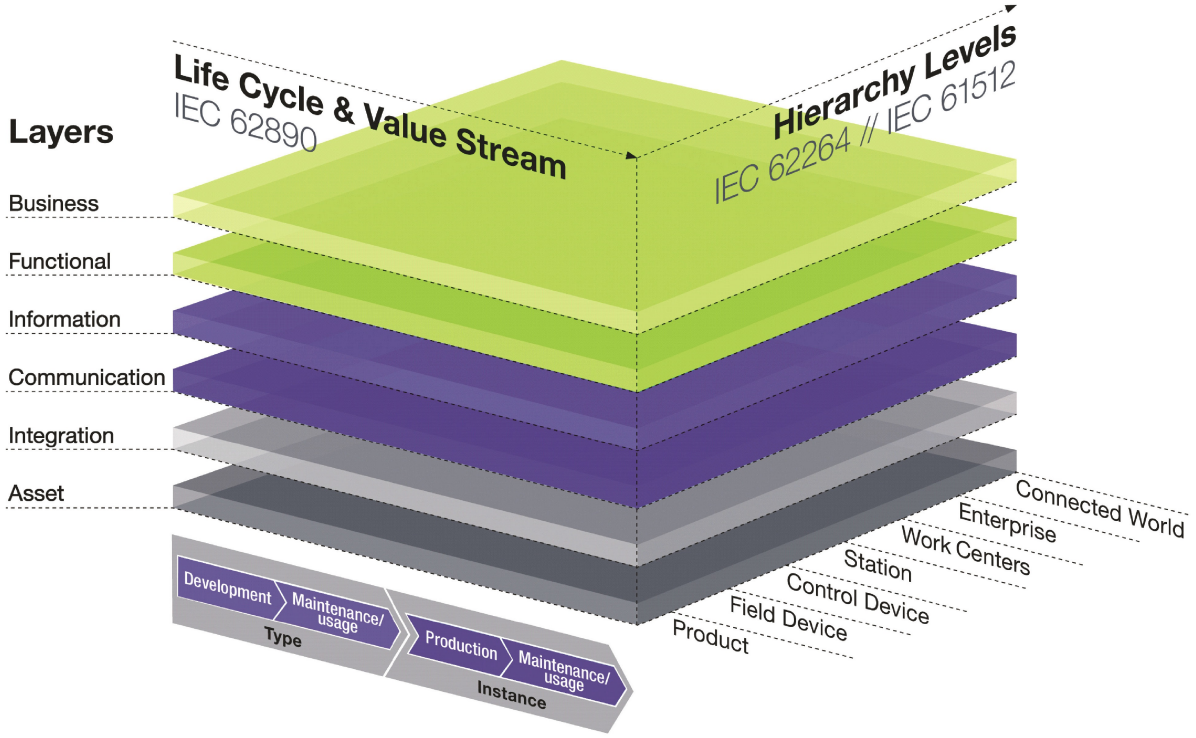

Abbildung 2.1.: RAMI4.0 [DIN16]

AT&T, Cisco, General Electric (GE), IBM und Intel gegründet wurde, werden in diesem Zusammenhang von Manzei, Schleupner und Heinze [MSH16] als Beispiele aufgeführt. Mit Hilfe von Anwendungsszenarien und in Kooperation mit internationalen Partnern will die Plattform I4.0 den Mehrwert digitaler Lösungen vor allem für mittelständische Unternehmen aufzeigen. Damit soll die deutsche Wirtschaft zum einen auf die Veränderungen durch Industrie 4.0 vorbereitet werden und zum anderen auch weiterhin wettbewerbsfähig bleiben.

Zur tatsächlichen Umsetzung der Vision von Industrie 4.0 als ein weltweites, unternehmensübergreifendes System von intelligenten Fabriken, Maschinen, Ideen, Plänen und Produkten bedarf es einiger Standards und Richtlinien. Mit dem Referenzarchitekturmodell Industrie 4.0 (RAMI4.0) [DIN16] legt die Plattform I4.0 die Basis für ein einheitliches Verständnis auf diesem Gebiet. Dem Modell entsprechend werden Informationen zu strukturellen Eigenschaften eines Gegenstandes oder Assets auf verschiedene Schichten abgebildet. Sie werden dabei aus geschäftlicher Sicht, d. h. hinsichtlich finanzieller und organisatorischer Rahmenbedingungen des Geschäftsprozesses, anhand der Funktionen und Kommunikationswege des Assets im Kontext des Industrie 4.0 Systems sowie dabei produzierter und verarbeiteter Daten beleuchtet. Zusammen mit der Beschreibung der Anbindung von realem Gegenstand zur Informationswelt bilden diese Informationen die Schichten der Architektur-Achse in Abbildung 2.1. Die Achse *Life Cycle & Value Stream* beschreibt den Zeitpunkt im Lebenszyklus des Assets, also ob es sich beispielsweise um eine *Typenbeschreibung* durch Entwurfsdaten oder eine konkrete *Instanz*

des Produktes handelt. Eine weitere Achse erlaubt es außerdem, das Asset in die Hierarchie der Fabrikumgebung einzuordnen. Mit Hilfe von Abhängigkeiten und Zuordnungen können Daten zu unterschiedlichen Zeitpunkten im Lebenszyklus eines Gegenstandes, wie z. B. Planungsinformationen und Daten zu unterschiedlichen Variationen eines Produktes, getrennt verwaltet und somit auch komplexe Zusammenhänge übersichtlich beschrieben werden [Ado+16].

Es ist zu beachten, dass sich ein Objekt nur sinnvoll in das Modell einordnen lässt, wenn es den Voraussetzungen einer Industrie 4.0 Umgebung entspricht. Eine sogenannte I4.0 Komponente muss laut DIN SPEC 91345 [DIN16] weltweit eindeutig identifizierbar, im Lebenszyklus entweder Typ oder Instanz und kommunikationsfähig sein. Als zentralen Baustein hat die Plattform I4.0 an dieser Stelle die Verwaltungsschale definiert. Ihre Aufgabe besteht darin, Eigenschaften und Funktionen eines Assets strukturiert aufzulisten und anderen Komponenten zugänglich zu machen [Ado+16]. Die Merkmale eines Assets werden dazu in einem Manifest festgehalten, das als Katalog für alle verfügbaren Funktionen und Daten fungiert. Durch Bereitstellen und Verwalten von Diensten und Zugriffsrechten im Komponentenmanager erfolgt die Anbindung an das Kommunikationsnetz der Fabrik. Die Verwaltungsschale sorgt also dafür, dass die erforderlichen Kriterien für eine I4.0 Komponente erfüllt werden, und kann damit, wie in Abbildung 2.2 veranschaulicht, ein beliebiges Asset in den Kontext einer intelligenten Fabrik einbinden.

Da sich eine I4.0 Komponente möglichst dynamisch in einer Industrie 4.0 Umgebung verhalten soll, ist hierbei eine Möglichkeit zur differenzierten Betrachtung der Daten und Funktionen eines Assets erforderlich. Schließlich sind nicht zu jedem Zeitpunkt alle Informationen über ein Asset von Interesse. Adolphs et al. [Ado+16] beschreiben daher sogenannte Sichten, denen jedes in einer Verwaltungsschale gespeicherte Merkmal zugewiesen werden muss. Dazu gehören beispielsweise die örtliche Sicht, die Informationen zur Position einer Komponente und ihrer Bestandteile beinhaltet, oder die Netzwerksicht, welche die Vernetzung einer Komponente im Produktionsumfeld hinsichtlich der Anbindung sowohl an elektrische Systeme als auch z. B. an die Materialzufuhr beschreibt. Laut Adolphs et al. [Ado+16] kann die Zuweisung der Sichten nur in den wenigsten Fällen automatisiert durchgeführt werden und soll daher bereits während der Implementierung einer Verwaltungsschale erfolgen. Da hierfür ein Experte mit tiefgehendem Wissen über anfallende Daten und typische Anwendungsszenarien benötigt wird, ist dies mit zusätzlichem Personal- und Zeitaufwand verbunden. Bei den in Tabelle 2.1 aufgeführten Definitionen handelt es sich um Basis-Sichten, d. h. sie können je nach Anwendungsfall und Bedarf um weitere Sichten ergänzt werden. Vorstellbar ist beispielsweise die Aggregation von Daten, die für Data Science interessant sind. In Fabrikumgebungen könnten hier unter anderem häufig auftretende Sensorwerte, Ausfallraten sowie Wartungsintervalle betrachtet und auf kausale Zusammenhänge untersucht werden.

Als Grundlage für Interaktionen zwischen I4.0 Komponenten erörtern Graf et al. [Gra+16] die Rolle der Kommunikation. Dabei fokussieren sie auf die Umsetzung von Kommunikationsstrukturen, die sich durch eine möglichst nahtlose Integration auszeichnen. Um den Anforderungen zukünftiger I4.0 Systeme zu genügen, sind zudem neue Strategien für die Migration, Integration und Wartung entlang aller Lebenszyklusphasen sowie ein integriertes

Sicht	Beispiel
Geschäftlich	Daten und Funktionen, welche die geschäftliche Eignung und Leistung einer Komponente zu den Lebenszyklusphasen Beschaffung, Konstruktion, Betrieb und Verwertung erlauben (z. B. Preise, Lieferbedingungen, Bestellcodes).
Konstruktiv	Merkmale zu physischen Dimensionen und zu Eingangs-, Verarbeitungs- und Ausgangsgrößen der Komponente, modulare Sicht auf Teilkomponenten bzw. eine Geräte-Struktur.
Leistung	Leistungs- und Verhaltensmerkmale, kann zu Simulationszwecken verwendet werden.
Funktional	Funktion der Teilkomponenten, Fachliche Funktionalität (z. B. Auslegungs-, Inbetriebnahme-, Berechnungs- oder Diagnosefunktionen).
Örtlich	Positionen und örtliche Zusammenhänge der Komponente oder ihrer Teile.
Security	Markiert ein Merkmal als sicherheitskritisch.
Netzwerksicht	Elektrische, fluidische, Materialfluss-technische- logische Vernetzung der Komponente.
Lebenszyklus	Aktueller Zustand und historische Verwendung der Komponente (z. B. Zuordnung zur Produktion Wartungsprotokolle, vergangene Verwendungszwecke).
Mensch	Aufbereitete Merkmale aus allen Sichten, damit der Mensch einzelne Elemente und Zusammenhänge begreifen kann.

Tabelle 2.1.: Basis-Sichten der Verwaltungsschale nach Adolphs et al. [Ado+16]

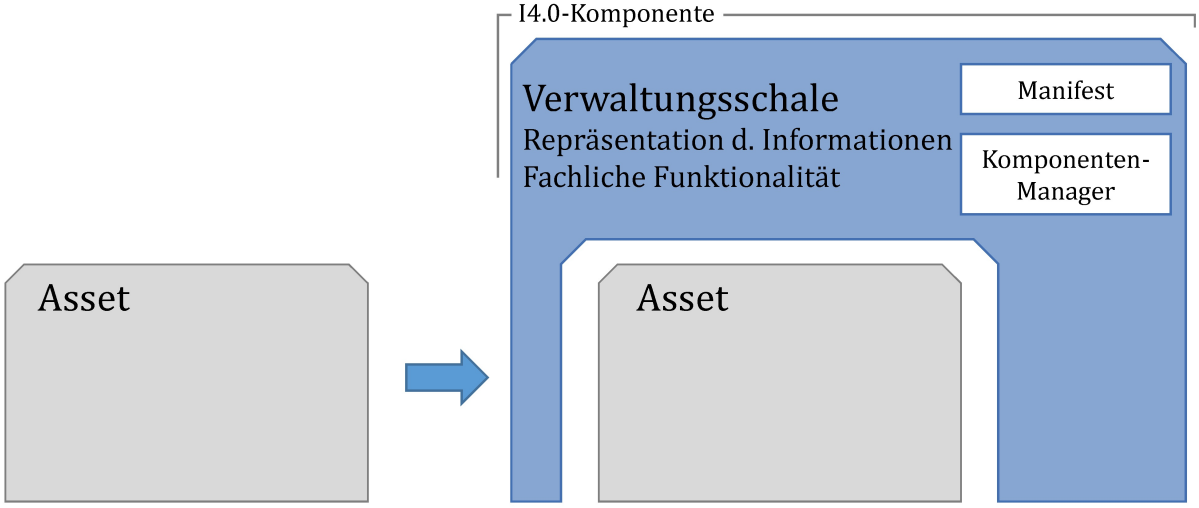

Abbildung 2.2.: I4.0 Komponente als Kombination aus Asset und Verwaltungsschale [DIN16]

Netzwerkmanagement erforderlich. Mit diesen Voraussetzungen soll eine automatische und vor allem selbstständige Konfiguration netzwerkbasierter Kommunikation im Stile einer I4.0 Komponente ermöglicht werden. Außerdem gilt es hierbei, die Einschränkungen aktueller Systeme zu überwinden. Dazu gehört laut Graf et al. [Gra+16] neben mangelhaftem Netzwerkmanagement beispielsweise auch Inkompatibilität zwischen Schnittstellen verschiedener Hersteller. Sie greifen auch die Aufgabe der Verwaltungsschale, nämlich die Bereitstellung notwendiger Eigenschaften und Funktionen, damit Dienste wie erwartet miteinander interagieren können, auf und schlagen eine wichtige Ergänzung vor: Die Interaktion soll dabei funktionieren, ohne dass Details über die Implementierung der Netzwerkumgebung bekannt sein müssen. Damit wird einerseits korrekte Funktionalität in Fabriken mit unterschiedlichen Netzwerkarchitekturen gewährleistet und andererseits auch auf aktuelle Entwicklungstrends hin zu dezentralen und Cloud-basierten Kommunikationssystemen eingegangen. Relevante Daten, Funktionen und Sicherheitsaspekte müssen also standardisiert vorliegen und gewisse Quality of Service (QoS)-Aspekte erfüllen. Dies fällt nach Graf et al. [Gra+16] ebenfalls in den Zuständigkeitsbereich der Verwaltungsschale.

Damit eine Interaktion zwischen I4.0 Komponenten über die vorhandenen Kommunikationswege erfolgen kann, sind also Dienste erforderlich, die den Zugriff auf Daten und Funktionen verwalten. In einem ersten Diskussionspapier befassen sich Bock et al. [Boc+16a] mit der Definition von Interaktionsabläufen zwischen solchen Diensten. Verortet werden sie im Komponenten- oder auch Interaktionsmanager der Verwaltungsschale. Die Abwicklung von Anfragen erfolgt dabei in der Regel über einen Port, der bei Bedarf mit Hilfe eines Sessionmodells auch an andere Ports umgeleitet werden kann. Im Rahmen eines I4.0 Szenarios, d. h. bestimmter Aufgaben entlang der Produktionswege, die durch Kooperation mehrerer I4.0 Komponenten gelöst werden, muss vor der eigentlichen Ausführung zuerst geprüft werden, ob die entsprechende Komponente über die erforderlichen Fähigkeiten verfügt. Bock et al. [Boc+16a] schlagen dazu die Speicherung gewisser Interaktionsmuster vor. Sie sollen beispielsweise beschreiben, ob eine Komponente in der Lage ist, eine Aufgabe zu initiieren, auszuführen oder zu beauftragen. Die Merkmale bzw. die Anforderungen einer entsprechenden Anfrage werden dabei mit den im Manifest der Verwaltungsschale hinterlegten Zusicherungen abgeglichen. „Es entsteht eine Sprache, in der Nachrichteninhalte, deren Abfolgen (Interaktionsmuster) und Regeln zur Steuerung von Alternativen im Ablauf definiert sind." [Boc+16a, S. 6]

Trotz der Bemühungen der Plattform I4.0 existieren noch einige Unklarheiten auf dem Weg zu intelligenten Produktionsumgebungen. So fehlt derzeit eine verlässliche Abgrenzung, in welchen Fällen Daten als Merkmale in der Verwaltungsschale abgelegt oder stattdessen als eigene I4.0 Komponenten verwaltet werden sollen. Dies ist insbesondere für die Realisierung eines digitalen Zwillings, der die vollständige Lebenslaufakte eines Objektes oder Produktes enthält, relevant. Dazu gehören alle nötigen Daten im Lebenszyklus, z. B. der Bearbeitungsstatus bei Produkten, der Betriebszustand von Maschinen und Wartungsintervalle. Im Zuge dieser Arbeit liegt der Fokus hierbei auf der Bereitstellung solcher Daten für Analysezwecke. Ob und welche dieser Daten nun direkt in der Verwaltungsschale des Gegenstandes annotiert oder mittels Verweisen auf andere I4.0 Komponenten referenziert werden sollen, wird in den Ausführungen der Plattform I4.0 nicht eindeutig festgelegt. Die Plattform arbeitet jedoch

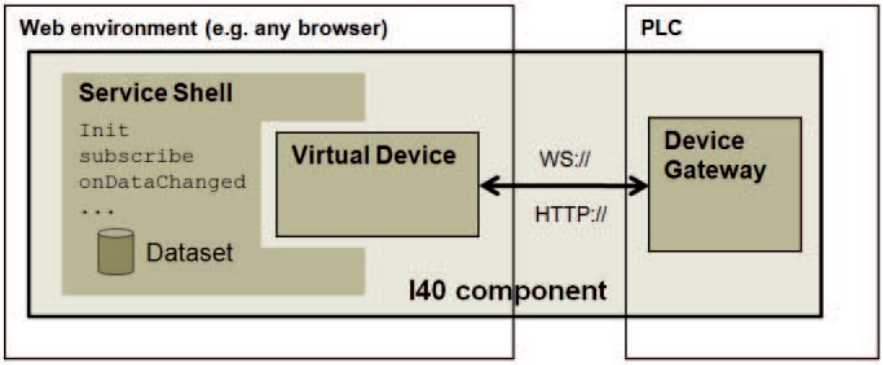

Abbildung 2.3.: Modell einer SPS als I4.0 Komponente [LRP16]. Das „Device-Gateway“ verbindet die digitale SPS mit der physischen und macht deren Funktionalität zugänglich

stetig an neuen Veröffentlichungen und Weiterentwicklungen ihrer Konzepte. Im Zuge dessen können diese und weitere Unklarheiten beseitigt und potentielle neue Erkenntnisse spezifiziert werden.

Bereits heute arbeiten einige Projekte an einer Umsetzung der genannten Lösungsansätze im Kontext von Industrie 4.0. Sowohl wissenschaftliche als auch industrielle Forschungsprojekte untersuchen daher Anwendungsfälle, wie sie in einem solchen Szenario auftreten können, und optimieren dabei Produktions- und Arbeitsabläufe sowie eingesetzte Technologien. Außerdem werden auch durch deren Erkenntnisse mangelhaft definierte Bereiche aufgedeckt und Möglichkeiten zur Verbesserung vorgeschlagen. Langmann und Rojas-Peña [LRP16] beschäftigen sich beispielsweise damit, wie sich eine speicherprogrammierbare Steuerung (SPS) durch Hinzufügen einer Verwaltungsschale als I4.0 Komponente realisieren lässt. Dabei machen sie auch auf Defizite aktueller Service-Lösungen für SPS-Systeme aufmerksam: Aufgrund aufwändiger Kommunikationsprotokolle sind Übertragungszeiten für Daten, die zwischen einem Controller, also einem technischen Gerät, und einem globalen Netzwerk mit Cloud- und Web-Diensten ausgetauscht werden, relativ hoch. Außerdem sind weder standardisierte Netzwerkschnittstellen für die Anbindung und Ansteuerung von SPS Controllern vorhanden noch wurden bisher beim Design solcher Systeme die in Industrie 4.0 gestellten Anforderungen in Betracht gezogen. Um diese Probleme umgehen zu können, führen Langmann und Rojas-Peña [LRP16] ein sogenanntes *Device Gateway* ein. Dabei handelt es sich um ein Modul, das Prozessdaten des SPS Controllers schnell, zuverlässig und sicher im IP-Netzwerk der Fabrikumgebung bereitstellt. Damit besteht bereits eine Möglichkeit zur Überwachung und Analyse der Daten in Echtzeit. Aufgrund des begrenzten Speichers von SPS-Systemen verorten Langmann und Rojas-Peña [LRP16] die Verwaltungsschale nicht auf dem Gerät selbst, sondern dagegen außerhalb des Controllers. Wie in Abbildung 2.3 zu sehen, erfolgt die Zuordnung zwischen der tatsächlichen Steuerungseinheit und der Verwaltungsschale mit Hilfe der sogenannten *Virtual Device*, also einer digitalen Repräsentation des Gerätes. Diese fasst alle Funktionen des Controllers zusammen und macht sie im Netzwerk zugänglich.

OPC Unified Architecture

Die Vernetzung von Verwaltungsschalen erfordert den Einsatz service-orientierter Architekturen. Mit OPC UA steht dafür ein mächtiges Framework der Open Plattform Communications (OPC) Foundation[4] zur Verfügung. Ursprünglich hat die OPC Foundation eine Schnittstelle für den Austausch von Daten zwischen Microsoft Windows-gebundenen Anwendungen, welche lange Zeit die Automatisierungstechnik dominierten, und anderen Plattformen entworfen. Auf dieser Grundlage baut nun die seit 2006 spezifizierte OPC UA[5] auf.

Als SOA und aufgrund der Verwendung von standardisierten Kommunikationstechnologien (derzeit Web Services mit HTTP bzw. HTTPS, oder Binary TCP) ist sie plattformunabhängig und kann damit direkt auf Controllern ausgeführt werden [Leh+11]. Die Spezifikation umfasst ein Metamodell, das Adressraummodell, welches grundlegende Konzepte wie Knotenklassen, Variablen-, Objekt- und Referenztypen beschreibt [OPC15a]. Informationsmodelle basieren auf diesem Metamodell und definieren speziellere Typen oder auch Instanzen. Dadurch ergibt sich ein Netzwerk aus sich gegenseitig referenzierenden Knoten verschiedener Klassen für die Repräsentation von Methoden, Objekten oder Variablen. Jede Knotenklasse verfügt neben den Basisattributen, wie Name und Identifikation, über spezielle Attribute für ihre Zwecke. Der Wert einer Variablen, welche z. B. einen Messwert enthalten kann, wird beispielsweise durch deren Value Attribut repräsentiert [EM11]. Im Adressraum eines OPC UA Servers werden alle Knoten hierarchisch strukturiert. Da Referenzen zwischen den Knoten möglich sind, können die Hierarchieebenen aufgelockert werden und es entsteht ein zusammenhängendes Netzwerk, in dem alle Knoten erreichbar sind. Neben konkreten Instanzen sind dabei auch die zugehörigen Typen zugänglich [OPC14]. Sie tragen zum Verständnis der Beziehungen von Knoten und deren Eigenschaften bei, da sie die Metainformationen entsprechend des verwendeten Informationsmodells enthalten. Gerade die Informationsmodelle, welche alle das in OPC UA spezifizierte Metamodell erweitern, verbessern die Interoperabilität zwischen Client und Server, da die Kommunikation von Daten unter standardisierten Richtlinien abläuft [EM11]. Dadurch ergeben sich umfassende Modellierungsmöglichkeiten für Hersteller und Standardisierungsgremien.

Die ursprünglich 13-teilige OPC UA Spezifikation wurde erst kürzlich um einen weiteren Teil [OPC17] ergänzt, der Publish-/Subscribe-Mechanismen einführt. Mit dieser Erweiterung wird die Notwendigkeit für eine Ende-zu-Ende-Verbindung zwischen OPC UA Client und OPC UA Server aufgelöst. OPC UA findet so in Umgebungen, die mit Cloud-Systemen arbeiten, ein neues Anwendungsgebiet. Derzeit werden die Kommunikationsabläufe in OPC UA überarbeitet und die Kommunikation in Echtzeit unter Verwendung des Time-sensitive Networking (TSN)-Standards vorangetrieben. Somit wird OPC UA den hohen Anforderungen von Industrie 4.0 gerecht und wird sich in deren Zusammenhang sehr wahrscheinlich als allgemeiner Standard durchsetzen [Spi07].

[4] https://opcfoundation.org/
[5] https://opcfoundation.org/developer-tools/specifications-unified-architecture

3. Anforderungen

Im Zuge dieser Arbeit sollen die Ergebnisse der Plattform I4.0 anhand ihres Potenzials zur Realisierung eines digitalen Zwillings analysiert werden. Damit ist die Fähigkeit gemeint, ein vollständiges Abbild eines Gegenstandes von der Idee bis zur Produktion und darüber hinaus in der digitalen Welt darzustellen. Hierfür soll zunächst die Verwaltungsschale[1] der Plattform I4.0 evaluiert werden. Anschließend soll ein Prototyp für einen digitalen Zwilling gebaut werden, der den folgenden Anforderungen entspricht.

3.1. Fachliche Anforderungen

Es sollen Daten zu einem Gegenstand bzw. zu einer Maschine verwaltet werden können. Dabei ist eine Aufteilung in Stammdaten, welche beispielsweise Informationen zum Hersteller oder den Standort des Objektes umfassen, Sensordaten und relevante Dokumente, wie Bedienungsanleitungen, erforderlich. Außerdem soll der gegenseitige Austausch dieser Daten zwischen Maschinen und Anlagen in einem einheitlichen Format ermöglicht werden. Als Grundlage für die Kommunikation über das dafür benötigte Netzwerk dienen die Richtlinien der OPC UA. Hintergrund dafür ist die Ermöglichung von Kooperation zwischen Maschinen, die dadurch selbstständig Abläufe im Produktionsumfeld koordinieren können sollen. Des Weiteren ist eine Anbindung an höherwertige Dienste und übergelagerte Funktionen in der IT-Architektur der Fabrik vorgesehen. Sie sollen Auszüge der gesammelten Daten erhalten, die für Analysezwecke und Optimierungsvorgänge verwendet werden können. Außerdem sollen die Gegenstände in der Lage sein, selbst Informationen über sich an ihrem Standort bereit zu stellen. Damit muss beispielsweise ein Arbeiter an der Fertigungsstraße beim Ausfall einer Maschine nicht einen Wartungsingenieur oder den langen Weg in ein Archiv suchen, sondern kann direkt am Arbeitsplatz selbst im Handbuch Lösungsansätze nachschlagen.

Der Fokus dieser Arbeit liegt aufgrund der zeitlichen Einschränkungen zunächst auf der Verwaltung von Stammdaten und Sensordaten sowie deren Austausch mittels OPC UA. Die übrigen hier aufgeführten Anforderungen können gegebenenfalls in weiteren Arbeiten zum digitalen Zwilling ergänzt werden.

[1] https://github.com/acplt/openAAS

Nr.	Anforderung
1	Die Verwaltungsschale besteht aus Body und Header.
2	Der Body enthält Informationen zum jeweiligen Asset.
3	Der Header enthält Informationen über die Verwendung des Assets.
4	Die Verwaltungsschale besitzt als zentrales Element Manifest und Komponenten-Manager.
5	Die Informationen der Verwaltungsschale sind mittels service-orientierter Architektur zugreifbar (API) und berücksichtigen entsprechende Security-Anforderungen.
6	Die Verwaltungsschale repräsentiert Informationen zu Anwendungsaspekten.
7	Die Verwaltungsschale ist nach Sichten strukturiert.
8	Die Verwaltungsschale besitzt eine eindeutige ID.
9	Das Asset besitzt eine eindeutige ID.
10	Auch eine Fabrik ist ein Asset, das eine Verwaltungsschale besitzt und per ID ansprechbar ist. Das Konzept der Schachtelbarkeit sollte anwendbar sein.
11	Typen und Instanzen müssen als solche gekennzeichnet sein.
12	Die Verwaltungsschale kann Referenzen auf andere Verwaltungsschalen oder I4.0-Informationen enthalten.
13	Zusätzliche Merkmale, z.B. herstellerspezifische, müssen möglich sein.
14	Ein verlässliches Minimum an Merkmalen für jede Verwaltungsschale muss definiert sein.

Tabelle 3.1.: Grundsätzliche Anforderungen an die Verwaltungsschale nach DIN SPEC 91345 [DIN16]

3.2. Methodische Anforderungen

Zunächst soll die Verwaltungsschale der Plattform I4.0 hinsichtlich ihrer vorhandenen Funktionalität analysiert werden. Anschließend soll ein Architekturkonzept für die Implementierung mehrerer Verwaltungsschalen, die auf das Anwendungsszenario in Abschnitt 5.1 abgestimmt sind, geschaffen werden. Dabei wird auf die Entwicklung eines digitalen Zwillings geachtet, der auch aus mehreren Gegenständen und deren zugehörigen Verwaltungsschalen zusammengesetzt werden kann.

3.3. Technische Anforderungen

Während die Eigenschaften der Verwaltungsschale der RWTH Aachen analysiert werden, soll sie auch auf Kompatibilität mit dem Cross Domain Development Kit (XDK) der Firma Bosch überprüft werden. Ursprünglich sollte die Entwicklung eines Prototypen für den digitalen Zwilling ebenfalls unter Verwendung der vom Institut für Parallele und Verteilte Systeme

(IPVS) der Universität Stuttgart zur Verfügung gestellten XDKs erfolgen. Aufgrund der in Abschnitt 5.3.1 auf Seite 53 beschriebenen Limitierungen findet die Implementierung stattdessen auf einem Rasperry Pi statt. Für die Speicherung von Stammdaten, Sensordaten und Dokumenten zu einem Gegenstand, welcher im zu implementierenden Prototyp durch einen XDK bzw.. Raspberry Pi simuliert wird, ist außerdem die Beschaffung und Verwendung mindestens einer MicroSD Karte erforderlich.

3.4. Allgemeine Anforderungen an eine Verwaltungsschale

Zusätzlich zu den Ausführungen im RAMI4.0 (siehe Tabelle 3.1) hat die Plattform Industrie 4.0 in diversen weiteren Veröffentlichungen einige Eigenschaften rund um Assets und deren Verwaltungsschalen im Kontext Industrie 4.0 festgelegt. Im Folgenden werden diese, anhand sich an mehreren Stellen [DIN16][Ado+16][Gra+16] wiederfindender und gegenseitig ergänzender Aussagen, gebündelt aufgelistet:

- Die Verwaltungsschale fasst die wichtigsten Eigenschaften eines Gegenstandes zusammen, damit eine Repräsentation des Assets und seiner fachlichen Funktionalität möglich ist.
- Die Informationen in der Verwaltungsschale beinhalten auch Laufzeitdaten des Assets.
- Informationen werden durch hierarchisch strukturierte Merkmale beschrieben.
- Die Verwaltungsschale enthält eine beliebige Anzahl I4.0-konformer Teilmodelle. Jedes Teilmodell verfügt über hierarchisch organisierte Merkmale, die auf individuelle Daten und Funktionen referenzieren.
- Ein Asset kann mehrere Verwaltungsschalen (zu unterschiedlichen Zeitpunkten im Lebenslauf) besitzen.
- Die Verwaltungsschale soll Merkmale aus unterschiedlichen Domänen in zueinander abgegrenzten Teilmodellen aufnehmen können, welche unabhängig voneinander versioniert und gepflegt werden können.
- Unterstützte Merkmale stammen aus harten Standards (z. B. DIN, ISO, IEC), konsensualen Standards (z. B. eCl@ss[2]) oder aus situationsabhängigen Rahmenbedingungen (z. B. interne Firmenstandards, verwendete Dateiformate).
- Die Verwaltungsschale muss zwischen Merkmalen aus unterschiedlichen Kategorien unterscheiden können.

[2] https://www.eclass.eu/

- Der Komponenten-Manager ...
  - ist ein Dienst, der in der Verwaltungsschale enthaltene Informationen pflegt.
  - bietet die Schnittstelle zur Informationswelt bzw. zum übergeordneten Netzwerk.
  - ist für die Adressierung und Identifikation von Verwaltungsschalen und Assets zuständig.
  - verwaltet die Ressourcen der I4.0 Komponente.
- Das Manifest ...
  - ist ein eindeutig aufzufindendes Inhaltsverzeichnis aller Informationen, Daten und Funktionen der Verwaltungsschale.
  - umfasst Metainformationen über nicht- sowie funktionale Eigenschaften der I4.0 Komponente.
  - beinhaltet charakteristische Merkmale des Assets und deren Beziehungen untereinander.
  - ist grundsätzlich öffentlich zugänglich (im Rahmen der geltenden Sicherheitsbestimmungen).

4. Evaluierung der open Asset Administration Shell

Ein Aufgabenbereich dieser Arbeit ist die Analyse bestehender Lösungen auf dem Gebiet der Verwaltungsschalen im Kontext Industrie 4.0. Die RWTH Aachen bietet mit der open Asset Administration Shell (openAAS) eine quelloffene und daher für akademische Zwecke interessante Implementierung in C an. Um deren Konzepte und Ergebnisse zu präsentieren, haben die Entwickler zusätzlich einen Demonstrator erstellt, dessen Aufbau in Abbildung 4.1 abgebildet ist. Da diese Implementierung als Grundlage für die Ziele dieser Arbeit dient, wird sie in den folgenden Abschnitten genauer beleuchtet.

4.1. Grundlegende Projektstruktur

Hier soll ein Überblick über die einzelnen Bestandteile des openAAS-Projektes geliefert werden. Dabei wird die Aufteilung in verschiedene Teilbereiche, sowie relevante Inhalte der zugehörigen Repositories auf GitHub beschrieben.

4.1.1. Dokumentation

Das openAAS-Repository[1] enthält grundlegende Informationen zum Projekt openAAS. Dazu gehören Dokumentationen zu verwendeten Konzepten, wie dem Property Model, welches von Epple, Palm und Azarmipour [EPA16] beschrieben und in Abschnitt 4.3 erläutert wird. Epple [Epp16] beschreibt außerdem Kommunikationsmuster, die nach einem von der RWTH Aachen eigens definierten Modell ablaufen. Für eine Interaktion im Sinne dieser Arbeit sind diese Muster zunächst nicht weiter von Belang und werden daher auch nicht genauer betrachtet. Ansonsten enthält dieses Repository neben einer Readme-Datei noch Lizenzinformationen und die beiden Unified Modeling Language (UML)-Diagramme aus Abbildung 4.1 und Abbildung 4.3. Eine Dokumentation zur Implementierung bzw. zum Quellcode ist nicht vorhanden.

[1] https://github.com/acplt/openAAS

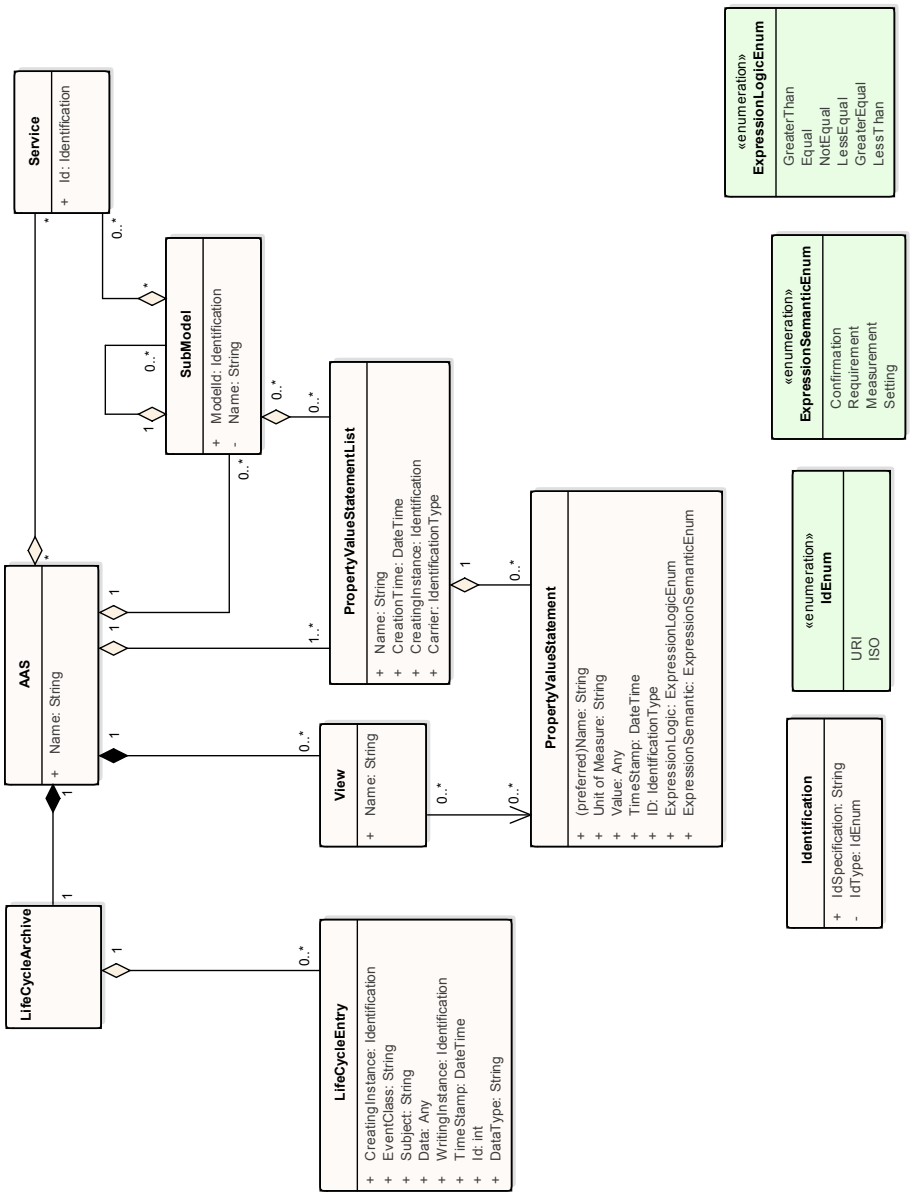

Abbildung 4.1.: UML-Modell der openAAS

4.1.2. Demo-Anwendung

Für dieses Kapitel von besonderem Interesse ist das Repository der Demo-Anwendung[2] zum openAAS-Projekt. Darin enthalten sind zunächst die Darstellung aus Abbildung 4.2 und in einer Readme-Datei gesammelte Informationen zur Einrichtung der Demo unter Verwendung eines Raspberry Pi, sowie zu Eigenschaften der Demo selbst. Des Weiteren sind zwei wichtige Extensible Markup Language (XML)-Dokumente hinterlegt. Eines von ihnen enthält Definitionen zum sogenannten Property Model, welches spezielle Typen und Variablen zur strukturierten Auflistung von Eigenschaften eines Assets bzw. dessen Verwaltungsschale beschreibt. Das andere Dokument entspricht dem späteren Aufbau des OPC UA Servers, über den gespeicherte Daten erreichbar sind. Implementiert wird dieser Server in der C-Datei *dht22.c*, wobei die Bibliotheken *Adafruit_Python_DHT*[3] als Schnittstelle zu angeschlossenen Sensoren und *open62541*[4] als Basis für den OPC UA Server zum Einsatz kommen. Diese Referenzen befinden sich im Ordner *external*. Für das Kompilieren des Quellcodes enthält die Datei *CMakeLists.txt* einige Befehle, welche die Informationen aus den beiden XML-Dateien in Form einer C- und einer Header-Datei kapselt, sodass sie im Server Code korrekt verwendet werden können. Genauere Ausführungen zu Struktur und Inhalt des Servers folgen in Abschnitt 4.3.

4.1.3. Gesonderter Anwendungsfall

Das openAAS-Basisrepository verweist außerdem auf einen Anwendungsfall[5], der speziell für ein Meeting im Rahmen der Plattform I4.0 konstruiert wurde. Mit Hilfe einer Anbindung an LibreOffice will dieser Anwendungsfall das Verwalten und Bearbeiten von Verwaltungsschalen in dem Nutzer bekannten Oberflächen ermöglichen. Allerdings kommen dabei Laufzeitumgebungen zum Einsatz, die an anderer Stelle implementiert sind und deren Verständnis über die Ziele dieser Arbeit hinaus geht. Laut einer Beschreibung seitens der Entwickler dienen sie als Host für Verwaltungsschalen und Schnittstelle für den Zugriff auf darin abgelegte Daten. In den weiteren Ausführungen dieser Arbeit wird dieser Teil des Projektes also nicht weiter betrachtet.

4.2. Zur Analyse verwendete Werkzeuge

Während der Analyse kamen in verschiedenen Aufgabenbereichen die folgenden Programme und Werkzeuge zum Einsatz:

[2] https://github.com/acplt/openAAS_PropertyDemo
[3] https://github.com/adafruit/Adafruit_Python_DHT/tree/310c59b0293354d07d94375f1365f7b9b9110c7d
[4] https://github.com/open62541/open62541/tree/9b66a1d5e927b18a9f39b8dc4aae773c38a223ba
[5] https://github.com/acplt/openAAS_workshop

MS Visual Studio 2017 - Zur Darstellung des C-Quelltextes ist die Entwicklungsumgebung Visual Studio 2017 in der Enterprise-Version[6] verwendbar. Ohne entsprechende Lizenz kann dafür auch die Community-Version verwendet werden.

Prosys OPC UA Java SDK - Die Client-Anwendung der Version 2.3.0 des Software Development Kit (SDK)[7] ist unter der Angabe von Daten zur geplanten Nutzung als Testversion erhältlich. Sie kann dazu verwendet werden, über die Windows-Konsole Verbindungen mit dem OPC UA Server aufzubauen, dessen Knoten zu durchsuchen und verfügbare Methoden aufzurufen und zu testen.

UaExpert - Diese Anwendung[8] bietet die Möglichkeit, einen OPC UA Server mit einer grafischen Benutzeroberfläche zu untersuchen. Damit können mehr Informationen gleichzeitig angezeigt und auch veränderliche (Sensor-) Werte per Subscription überwacht werden. Allerdings ist es nicht möglich, Methoden mit Eingabewerten aufzurufen.

UaModeler - Mit Hilfe des UaModelers[9] lassen sich die Modelle aus den beiden XML-Dateien anschaulich betrachten. Statt etliche Zeilen XML-Code zu lesen, kann hier eine Baumansicht mit Informationen zu den Daten einzelner Knoten angezeigt werden.

Für die beiden letztgenannten Werkzeuge ist eine Registrierung unter https://www.unified-automation.com/ erforderlich.

4.3. Architektur des OPC UA Servers

Die Demo besteht aus einem Raspberry Pi, der mit einem Feuchtigkeits-, einem Temperatursensor und einer LED ausgestattet ist. Sensoren und LED werden vom Raspberry Pi zu einer gemeinsamen Einheit, hier *Multi Sensor* genannt, zusammengefasst. Dies geschieht mit Hilfe von vier teilweise verschachtelten Verwaltungsschalen, die jeweils Informationen zum dazu gehörigen Asset, sowie zur Verwaltungsschale selbst kapseln. Abbildung 4.2 zeigt ein entsprechendes Modell der Schalen des Multi Sensors und wie die zugehörigen Bauteile untereinander in Verbindung stehen. Entsprechend der Abbildung erhält jeder Sensor und auch die LED eine eigene Verwaltungsschale, die grundlegende Informationen enthält. Diese drei werden von einer übergeordneten Verwaltungsschale zu einem gemeinsamen Objekt, dem *Multi Sensor*, zusammengefasst. In der Demo ist dieser Aufbau in Form von sich gegenseitig referenzierenden Knoten in einem OPC UA Server umgesetzt. In ihnen werden beispielsweise Identifikationsnummern, Messwerte und Produkteigenschaften zum zugehörigen Knoten gekapselt. Besitzt dieser zudem aufrufbare Methoden, so werden diese entsprechend in Form eines oder mehrerer Methodenknoten angehängt. OPC UA Server mit einer großen Menge

[6] https://www.visualstudio.com/de/vs/enterprise/
[7] https://www.prosysopc.com/products/opc-ua-java-sdk/
[8] https://www.unified-automation.com/de/produkte/entwicklerwerkzeuge/uaexpert.html
[9] https://www.unified-automation.com/de/produkte/entwicklerwerkzeuge/uamodeler.html

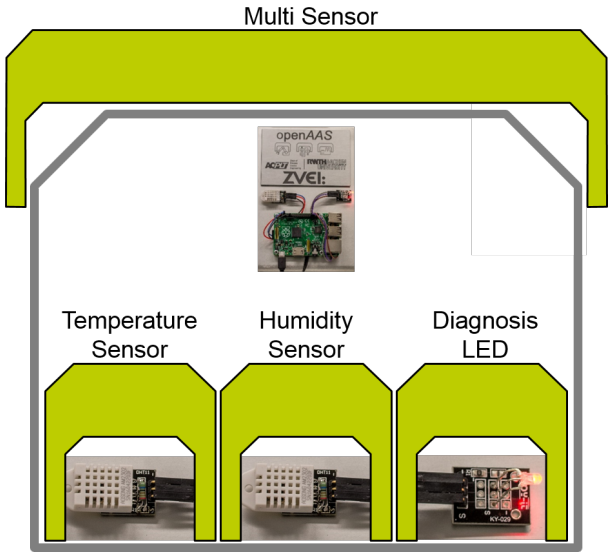

Abbildung 4.2.: Verschachtelung der openAAS-Verwaltungsschalen

an Knoten können leicht unübersichtlich werden, wodurch sich das Auffinden bestimmter Daten schwieriger gestaltet. Da auch die Demo potentiell viele Assets und deren Informationen verwalten will, werden relevante Daten auf unterschiedliche Namespaces aufgeteilt. Namespaces bieten die Möglichkeit, Knoten über ihren Identifikator zu gruppieren. Dies kann auch über unterschiedliche Hierarchieebenen des Informationsmodells hinweg geschehen. Neben dem Basis-Namespace, welcher die Konfiguration und standardmäßig verfügbare Typen des OPC UA Servers beschreibt, werden außerdem die folgenden drei Namespaces angelegt und verwendet:

1: http://openAAS.org/metra Dieser Namespace umfasst alle Knoten, die für Interaktion durch Nachrichtenübermittlung benötigt werden. Im Falle der Demo ist dies genau einer: Der Knoten der *Message*-Komponente besteht aus einem Objektknoten für den Nachrichteneingang (*Inbox*), sowie einem weiteren Objekt, das auf den *Multi Sensor* und seine Bestandteile verweist. So können diese als potentielle Nachrichtenempfänger gehandhabt werden. Tatsächlich kann jedoch nur die LED angesprochen werden, wie es in Listing 4.5 zu sehen ist.

2: http://acplt.org/OPCUA/Models/propertyModel Für die eigentliche Verwendung des OPC UA Servers als Benutzer ist dieser Namespace nicht weiter relevant. Er beinhaltet lediglich Definitionen von Typen, die für Objektknoten in anderen Namespaces verwendet werden. Hierbei ist jedoch das zugrundeliegende Modell durchaus interessant: Bock et al. [Boc+16b] haben bereits festgelegt, dass Eigenschaften von Asset bzw. Verwaltungsschale „in Form von Ausprägungsaussagen zu Merkmalen (Property Value Statements)“[Boc+16b, S. 40] beschrieben werden. Das heißt, es werden Bedingungen aufgestellt, die gewisse Merkmale wie beispielsweise den Wertebereich einer gemessenen Temperatur eingrenzen. Ein Merkmal, bzw.

eine Property, ist dabei immer einem Objekt, dem Merkmalsträger, zugeordnet. Epple, Palm und Azarmipour [EPA16] bezeichnen diesen in ihren Ausführungen als „Property Carrier“ und beschreiben relativ detailliert, wie die Ausprägungsaussagen in Sachen Inhalt und Struktur auszusehen haben. So definiert jedes Statement einen logischen Ausdruck über eine bestimmte Wertemenge eines Merkmals, der aus den folgenden Teilen besteht: 1. Referenzwert, 2. physikalische Einheit des Wertes, 3. Art der logischen Relation, 4. Semantik des Ausdrucks, 5. Zeitpunkt des Erstellens, 6. erstellendes Element. Zu den Typen logischer Relationen zählen neben den Operatoren $<, \leqslant, =, \neq, \geqslant, >$ auch die Mengenoperatoren $\in$ bzw. $\notin$. In der Implementierung der RWTH Aachen sind die letzteren allerdings nicht verfügbar. Dafür zeigt Abbildung 4.3 die Operatoren $<$ (engl. less than (LT)), $>$ (engl. greater than (GT)), usw. in einer Aufzählung, nämlich dem *ExpressionLogicEnum*. Alle Property Value Statements, die zum selben Merkmalsträger gehören, werden in einer eindeutig identifizierbaren Liste (*PropertyValueStatementList*) zusammengefasst. Dabei wird auch der Identifikator selbst spezifiziert, also ob beispielsweise Uniform Resource Identifier (URI), ein ISO-Standard oder eigen zur Identifikation zum Einsatz kommen. Die Aussage eines Ausdrucks hängt von der semantischen Bedeutung seiner Bestandteile ab. Sie muss also ebenfalls festgelegt werden. Hierfür haben Epple, Palm und Azarmipour [EPA16] vier verschiedene Semantiken definiert: *Requirement (R)* – Beschreibt feste Anforderungen, denen der Merkmalsträger entsprechen muss (z. B. „Sensor A darf maximal 0.5A Strom benötigen“). *Assurance (A)* – Versichert, dass der Merkmalsträger bestimmte Eigenschaften erfüllt (z. B. „Jedes Produkt von Typ XY hat vier durchgehende Bohrlöcher“). *Measurement (M)* – Gibt Auskunft über den momentanen Ist-Zustand eines Merkmals (z. B. „Die Temperatur um 13:33:37 beträgt 22.23°C“). *Setting (S)* – Beschreibt Bauart-bedingte oder festgelegte Werte (z. B. „Durchmesser eines Hosenrohrs ist DN 100“). Dass die *Assurance*-Semantik im Diagramm in Abbildung 4.1, welche von der RWTH Aachen wenige Monate nach der Erstellung von Abbildung 4.3 zum GitHub-Repository hinzugefügt wurde, stattdessen „Confirmation“ genannt wird, zeigt erneut, dass sich Definitionen und Konzepte rund um Industrie 4.0 derzeit immer noch in der Entstehung und Wandlung befinden. Außerdem wird in Abbildung 4.3 die Auflistung der *RAMS*-Semantiken fälschlicherweise als *ExpressionLogicEnum* bezeichnet. Dieser Fehler wurde bei der Übernahme in Abbildung 4.1 ebenfalls korrigiert.

3: http://acplt.org/OPCUA/Models/assetAdministrationShell Hierunter fallen alle Knoten mit Informationen zu Assets, Verwaltungsschalen oder deren Eigenschaften und Methoden. Im Modell der openAAS verfügt jeder Knoten über eine gewisse Menge an Daten, die im Server verwaltet werden: 1. Informationen zum Asset 2. Beschreibung der Verwaltungsschale und ihrer Teilmodelle 3. Weitere enthaltene Verwaltungsschalen (*SubShells*), wie z. B. die einzelnen Bauteile des *Multi Sensors*. Wie diese im Detail aufgelöst sind, zeigt Abbildung 4.5. Im Eintrag für *SubShells* werden andere Verwaltungsschalen referenziert, die dem aktuellen Knoten zugeordnet sind. Durch diese Art des direkten Verweises wird unnötige Redundanz von Daten verhindert, ohne dabei Informationen zu vernachlässigen. Dabei ist zu beachten, dass der *Multi Sensor* über keine Teilmodelle verfügt. Stattdessen verweist er auf die Verwaltungsschalen für Sensoren und LED. Da diese im Rahmen der Demo keine weiter darunterliegenden Assets besitzen, entfällt hier der Eintrag für *SubShells*. An deren Stelle sind dort Teilmodelle hinterlegt,

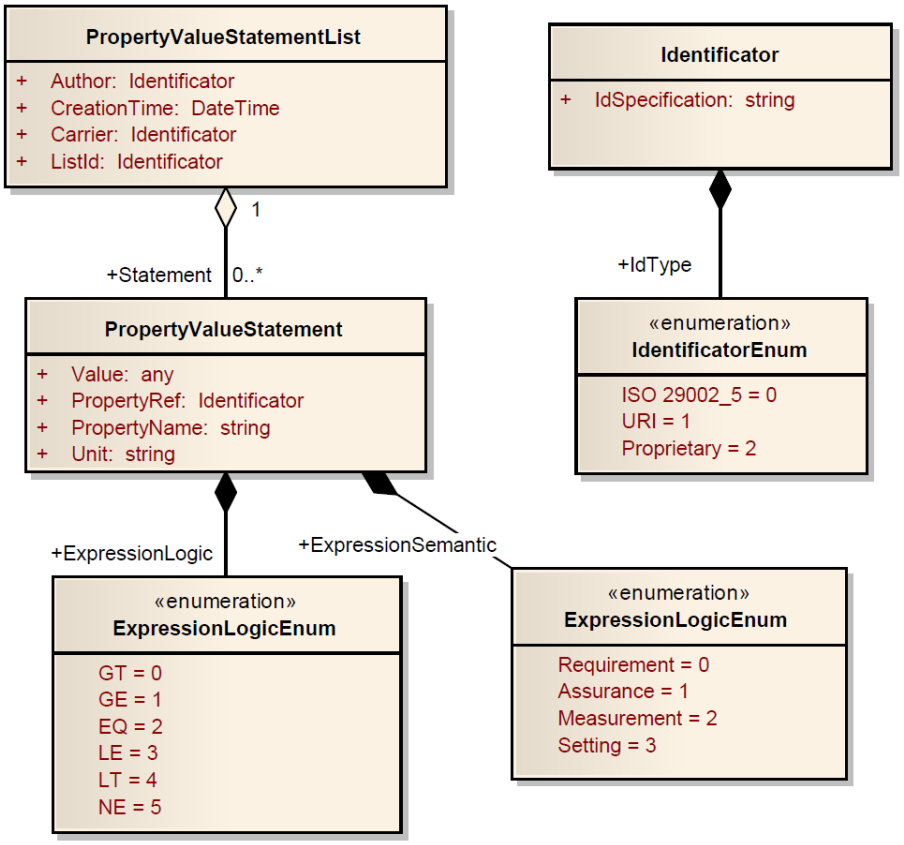

Abbildung 4.3.: Property Statement Model

27-20-04-05 **variable area flowmeter**

0173-1#02-AA0677#001	manufacturer name	Krohne	S	∈
0173-1#02-AA0847#003	description of product type	variable area flowmeter	S	∈
0173-1#02-AA0734#001	manufacturer product description	VA40	S	∈
0173-1#02-BAG982#006	nominal width	DN25	S	∈
0173-1#02-BAB322#005	form	N21.9	S	∈
0173-1#02-BAA048#006	max measurement error	1 %	A	<
0173-1#02-BAA036#008	max. process temperature	100 C	R	<
0173-1#02-BAG943#005	lowest value for volume flow rate	063 l/h	R	>
0173-1#02-BAG944#005	greatest value for volume flow rate	630 l/h	R	<

source:
http://www.eclasscontent.com/index.php?language=en&version=9.1

source:
© KROHNE 09/2013-4000416902 TD VA40/45 R02 en

©Epple2016

Abbildung 4.4.: Beispiele für Property Value Statements aus [EPA16]

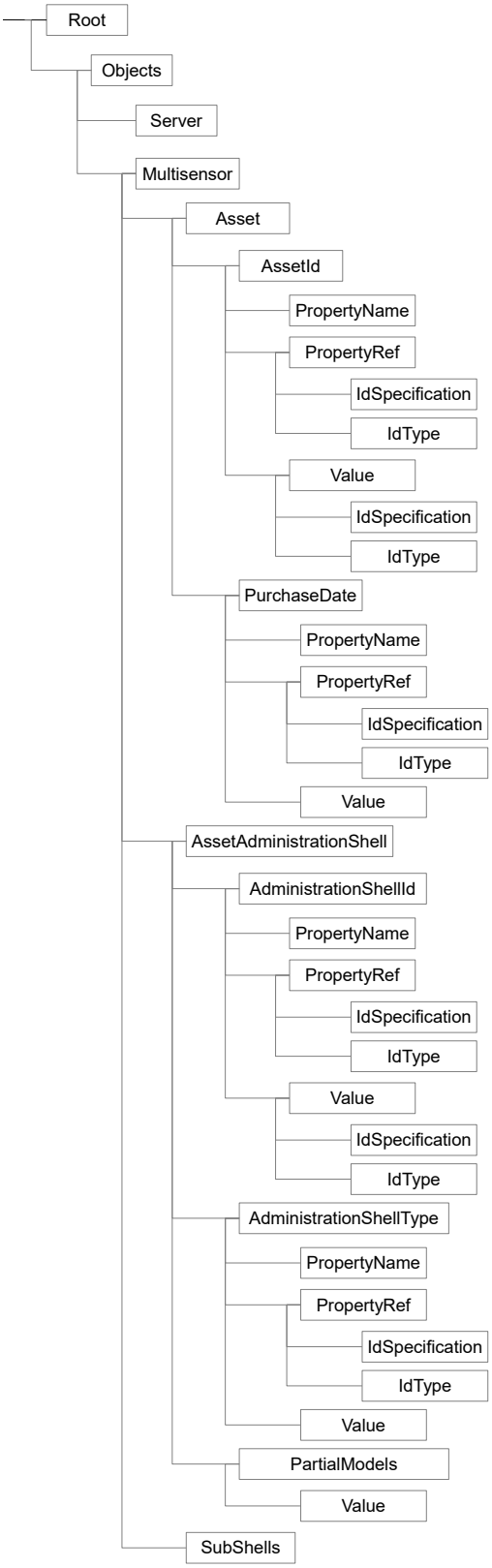

Abbildung 4.5.: Allgemeine Struktur von Knoten im OPC UA Server der openAAS-Demo entsprechend der zugehörigen XML-Dateien

welche einerseits weitere Daten zum zugehörigen Gegenstand und andererseits zu dessen Verwendung enthalten. Mit Hilfe von Property Value Statements werden beispielsweise maximal und minimal mögliche Messwerte festgelegt oder die verwendete Maßeinheit spezifiziert. Wie in den Abbildungen in Anhang A zu erkennen, kommen auch die Definitionen zur Ausdruckslogik und -semantik von Epple, Palm und Azarmipour [EPA16] bzw. aus Abbildung 4.3 zum Einsatz. So werden obere Grenzen für Wertebereiche mit dem $\leqslant$-Operator (*ExpressionLogicEnum(4) = LT*) und der *Assurance*-Semantik (= *ExpressionSemanticEnum(1)*) gekennzeichnet. Bei unteren Grenzen wird entsprechend der $\geqslant$-Operator (*ExpressionLogicEnum(0)*) verwendet. Wieso dabei im Knoten statt der Zahl 0 als Integer, siehe *AdministrationShellType* in Abbildung A.1, „Null“ eingetragen wird, ist unklar und liegt vermutlich an falschen Zuordnungsregeln bei automatisiert generierten Daten und Knoten. Selbiges Problem taucht beispielsweise auch in Abbildung A.2 bei der Anforderungsbeschreibung für die maximale Eingangsspannung am Feuchtigkeitssensor (*MaximumSupplyVoltage*) auf, bei der die *Requirement*-Semantik (= *ExpressionSemanticEnum(0)*) eingesetzt wird. Im „Measurement-Model“ der Sensoren lässt sich des Weiteren über einen speziellen Knoten (*Measurement*) innerhalb des Modells auch der tatsächlich gemessene Wert abfragen. Mit der LED als Aktor im *Multi Sensor* verhält es sich ähnlich, allerdings wird das beschreibende Modell dort „Actuator-Model“ genannt und stellt Informationen zum aktuellen Status (An/Aus) der LED bereit. Außerdem wird hier die besondere Eigenschaft eines Objektknotens genutzt, nämlich die Möglichkeit zum Bereitstellen von Methoden. In diesem Fall ist eine Methode namens *switchLED* (siehe Listing 4.5) verfügbar, die zum Ein- oder Ausschalten der LED dient.

4.4. Beschreibung des Server Codes

Der Programmcode zur Erstellung der Knoten und zur Initialisierung des OPC UA Servers der openAAS Demo befindet sich in der Datei *dht22.c*. Die erste Aufgabe, welche dabei abgehandelt wird, ist das Einbinden von benötigten Bibliotheken in Form von Header-Dateien. Dazu gehören neben dem System-Header *system.h* auch die standardmäßige Konfiguration, sowie grundlegende Funktionen für Logging und Kommunikation über das Transmission Control Protocol (TCP) für den OPC UA Server. Außerdem werden noch zwei Header-Dateien für die Interaktion mit angeschlossenen Sensoren und die dafür benötigte Python-Integration hinzugefügt. Des Weiteren wird eine Datei namens *nodeset.h* eingebunden. Diese wird während der Kompilierung mittels *CMake* erstellt. Sie führt die Informationen aus den beiden XML-Dateien, die das Informationsmodell des Servers beschreiben, zusammen und macht sie zur Verwendung im C-Code verfügbar. Darauf folgen einige Definitionen von Variablen, darunter auch die des Sensor-Typs, der als *DHT22* gesetzt wird. Wie unschwer zu erkennen wurde der Name der C-Datei also entsprechend des Sensortyps gewählt. Als nächstes wird ein neuer Typ definiert, nämlich das *Readings*-Struct aus Listing 4.1. Es fasst Sensormesswerte mit dem Zeitpunkt der Abfrage und einer Status-Variable zusammen, die den Erfolg einer Interaktion mit den Sensoren repräsentiert. Darauf folgen einige Methoden, deren Funktion in Tabelle 4.1

Listing 4.1 Definition eines Typs zur Speicherung von Messwerten

```
typedef struct Readings {
      UA_DateTime readTime;
      UA_Float humidity;
      UA_Float temperature;
      UA_Int32 status;
} Readings;
Readings readings = { .readTime = 0, .humidity = 0.0f, .temperature = 0.0f, .status
    = DHT_ERROR_GPIO };
```

Methode	Beschreibung
stopHandler	Wartet während der Ausführung des Servers auf die Eingabe von „Strg + C“ im Konsolenfenster, um daraufhin den Server zu beenden.
readSensorJob	Wird vom Server alle 2,5 Sekunden ausgeführt und aktualisiert einen Wert in den Readings, falls erneutes Lesen erfolgreich war, oder der alte Wert älter als 60 Sekunden.
readSensor	Liest auf Anfrage eines Clients an einen entsprechenden Knoten den zugehörigen Sensorwert aus.
readLed	Gibt den aktuellen Betriebszustand der LED aus.
switchLed	Überprüft, ob eine Interaktion mit dem GPIO-Pin der LED möglich ist und schaltet diese je nach mitgegebenem Parameter („ON“/„OFF“) ein oder aus.
diagnosisMethod	Gleicht die übergebene Empfängeradresse mit der für die LED vordefinierten Adresse (siehe Listing 4.6) ab und ruft bei Übereinstimmung switchLed mit der erhaltenen Nachricht auf.
getNewNodeId	Gibt die nächsthöhere freie ID im Namespace des übergebenen Indexes zurück.
createComponent	Legt den den Knoten für die Message-Komponente unter der ParentId des Server Objektes (0, 2253).
main	Ist für die Initialisierung des OPC UA Servers zuständig. Dazu gehören Portzuweisung, Anlegen der Knoten mit allen zugehörigen Daten und Methoden, Definition der Namespaces und Endpunkte.

Tabelle 4.1.: Methoden im Server-Code und ihre Aufgaben

jeweils kurz beschrieben ist. Allerdings bedarf die *main*-Methode noch einiger zusätzlicher Erläuterungen:

Sie startet mit der Initialisierung des OPC UA Servers, indem ein Port für die Kommunikation über TCP zugewiesen wird. Hier ist dies der Port 16664 zunächst mit der Standardkonfiguration des OPC UA Servers. Als nächstes wird unter dem Namen *Message* die erste Applikation für einen Endpunkt des Servers angelegt. Damit sie auch für Clients auffindbar ist, weist

Listing 4.2 Ausschnitt aus der *main*-Methode der openAAS Demo

```
1       //default applications "sees" NS 0 and NS1
2       UA_ApplicationDescription* app1 = &config.applicationDescription;
3
4       app1->applicationName.text = UA_STRING_ALLOC("Message");
5       app1->applicationUri = UA_STRING_ALLOC("http://openAAS.org/metra");
6       app1->discoveryUrlsSize = 1;
7       app1->discoveryUrls = UA_Array_new(1, &UA_TYPES[UA_TYPES_STRING]);
8       app1->discoveryUrls[0] = UA_STRING_ALLOC("/app0");
9
10      UA_Server *server = UA_Server_new(config);
11
12      /* create nodes from nodeset */
13      nodeset(server);
14      UA_Server_addNamespace(server, "http://openAAS.org/metra");
15      /* add a sensor sampling job to the server */
16      UA_Job job = { .type = UA_JOBTYPE_METHODCALL, .job.methodCall = { .method =
            readSensorJob, .data = NULL } };
17      UA_Server_addRepeatedJob(server, job, 2500, NULL);
```

die *main*-Methode dem Endpunkt außerdem die Bezeichnung *app0* zu. So können später unter *„opc.tcp://<ServerHost>:16664/app0“* Verbindungen direkt mit der Applikation bzw. dem Endpunkt hergestellt werden. Je nach verwendetem Client ist auch der Zusatz *„/endpoint“* am Ende der URL notwendig. In den Zeilen eins bis acht in Listing 4.2 ist der für die Applikationserstellung zuständige Code abgebildet. Darauf folgt die Initialisierung des OPC UA Servers selbst. Direkt im Anschluss werden die Knoten aus den XML-Dateien bzw. aus der daraus generierten Datei *nodeset.h* eingebunden. In Zeile 14 wird dem Server der Namespace hinzugefügt, der alle für die *Message*-Applikation relevanten Knoten umfasst. Weiterhin wird die *readSensorJob*-Methode dem Server als wiederholt auszuführende Aufgabe hinzugefügt.

Im nächsten Teil der *main*-Methode werden sogenannte *DataSources* zu einigen der zuvor erstellten Knoten zugewiesen. Diese Datenquellen sind Callback-Methoden, die der Server bei einer Anfrage an den zugehörigen Knoten ausführt und dem Client ihren Rückgabewert zusendet. In der Demo behandeln solche Datenquellen beispielsweise die gezielte Abfrage von Sensorwerten, also außerhalb des ständig wiederholten Zyklus.

Nach dem Setzen der Datenquellen wird mit dem in Listing 4.3 dargestellten Code der Knoten für die *Message*-Applikation als Kind des Serverknotens (0, 2253) angelegt. Dazu wird zunächst die *NodeId* (1, 1) festgelegt und eine leere Menge an Attributen für den Knoten initialisiert. Diese wird in der aufgerufenen Methode *createComponent* mit dem Namen *COMCO* sowie einer Beschreibung des Knotens befüllt. Außerdem fügt die Methode dem Knoten ein Objekt namens *Inbox* vom Typ *FolderType* hinzu. Es wird vermutet, dass es eingegangene Nachrichten gebündelt darstellen soll. Die genaue Funktion dieses Objektes konnte jedoch nicht ermittelt werden, da sich im Lauf der Evaluierung hier keine Veränderung von Werten oder sonstige Reaktionen feststellen ließen. Zurück in der *main*-Methode wird dem eben erstellten *COMCO*-Knoten ein Array mit registrierten Komponenten, nämlich dem Feuchtigkeits-, Temperatur-

Listing 4.3 Initialisierung des Knotens mit der Message-Komponente

```
UA_ObjectAttributes objAtrPeerManager;
UA_ObjectAttributes_init(&objAtrPeerManager);
UA_NodeId COMCONodeId = UA_NODEID_NUMERIC(1, 1);
char* peerManager = "COMCO";
createComponent(&objAtrPeerManager, COMCONodeId, peerManager, server,
    UA_NODEID_NUMERIC(0, 2253));

...

UA_Server_addMethodNode(server, UA_NODEID_NUMERIC(1, 2),
    COMCONodeId,UA_NODEID_NUMERIC(0, UA_NS0ID_HASORDEREDCOMPONENT),
    UA_QUALIFIEDNAME(1, "dropMessage"), diagnosisAttr, &diagnosisMethod, server, 3,
    inputArguments, 0, NULL, NULL);
```

Listing 4.4 Zuordnung von Namespaces zur *AssetShell*-Applikation

```
UA_ApplicationDescription app0;
UA_ApplicationDescription_copy(&config.applicationDescription, &app0);
UA_String_deleteMembers(&app0.applicationName.text);
app0.applicationName.text = UA_STRING_ALLOC("AssetShell");
app0.discoveryUrlsSize = 1;
app0.discoveryUrls = UA_Array_new(1, &UA_TYPES[UA_TYPES_STRING]);
app0.discoveryUrls[0] = UA_STRING_ALLOC("/app1");
UA_UInt16 ns[3];
ns[0] = 0;
ns[1] = 2;
ns[2] = 3;
UA_Server_addApplication(server, &app0, ns, 4);
```

und Multi Sensor, hinzugefügt. In Listing 4.3 ist dieser relativ lange Abschnitt durch „...“ symbolisiert. Außerdem wird ein Methodenknoten namens *dropMessage* angelegt. Dieser akzeptiert als Eingabewerte eine Sender-, eine Empfängeradresse und eine Nachricht. Die übergebene Empfängeradresse muss dabei der in Listing 4.6 oben abgebildeten entsprechen. Nach einer Überprüfung der Parameter wird die Methode in Listing 4.5 mit der übergebenen Nachricht aufgerufen. Erlaubte Nachrichteninhalte sind dabei die Befehle „ON“ und „OFF“.

Fernerhin weist die *main*-Methode den Applikationen bzw. den Endpunkten des Servers die jeweils sichtbaren Namespaces zu. Die Applikation zur Verwaltung der Verwaltungsschalen kann beispielsweise auf die Namespaces 0, 2 und 3 zugreifen, welche im folgenden Abschnitt 4.3 genauer erläutert werden. Listing 4.4 enthält den Code für die Zuweisung im Falle der *AssetShell*-Applikation, welche aber für die beiden anderen Endpunkte genauso erfolgt. Letztendlich hat die *Engineering*-Applikation (*„/app2“*) Zugriff auf alle Namespaces (0, 1, 2 und 3), während die *Message*-Applikation nur die beiden Namespaces 0 und 1 zu sehen bekommt.

Danach werden die Knoten des Feuchtigkeits-, des Temperatursensors und der LED mit Hilfe einer *Organizes*-Referenz dem *SubShells*-Knoten des Multi Sensors zugeordnet. So ist gewähr-

leistet, dass dieser die untergeordneten Knoten sauber referenziert und an einer gemeinsamen Stelle zusammenfasst.

Als abschließende Vorbereitung vor dem Start des Servers wird nun ein weiterer Methodenknoten angelegt. Dieser enthält hier eine Methode zum direkten Bedienen der LED und ist dementsprechend auch dem Knoten der LED zugeordnet. Akzeptierte Eingabewerte sind hierbei die Befehle „ON“ und „OFF“, die mit der Methode *switchLED* in Listing 4.5 weiter verarbeitet werden.

Dann wird der Server gestartet und die *main*-Methode vorerst verlassen. Sobald der Befehl zum Stoppen des Servers aktiviert wird, kehrt die Ausführung hierher zurück und beginnt mit dem Aufräumen der erstellten Objekte. Zuerst wird der OPC UA Server mit allen enthaltenen Knoten gelöscht, gefolgt von der Netzwerkschicht und potentieller dort registrierter Kommunikationsteilnehmer. Abschließend erfolgt die Rückgabe eines Integer-Wertes an das Host-System, damit überprüft werden kann, ob die Konsolenanwendung ordnungsgemäß beendet wurde.

4.5. Datenzugriff über Endpunkte des OPC UA Servers

Zum Erreichen von Daten, die in den Variablen einzelner Knoten abgelegt sind, stellt die Demo drei Endpunkte, oder auch *Endpoints*, mit unterschiedlichen Zugriffsbereichen bereit. Der erste Endpoint (App 0), welcher bei einer Verbindung ohne Angabe eines spezifischen Endpoints standardmäßig verwendet wird, beinhaltet eine Schnittstelle für nachrichtenbasierte Kommunikation mit dem Server. Im Rahmen der Demo zählt hierzu allerdings nur das Umschalten der LED per übermittelter Nachricht. Um alle für den Endpoint nicht relevanten Knoten auszublenden, kommen hier die in Abschnitt 4.3 definierten Namespaces zum Einsatz. Die *Message*-App hat beispielsweise nur Zugriff auf den Namespace 1 (http://openAAS.org/metra) und natürlich den Basis-Namespace von OPC UA. Somit hat ein mit dieser Anwendung verbundener Nutzer keine Möglichkeit, Daten aus den Bereichen des Property Models oder der Administration Shells auszulesen.

Des Weiteren bietet der Server einen Endpoint (App 1) an, der lesend auf die Einträge der Administration Shells, sowie auf die vom Property Model definierten Typen zugreifen kann. Ein dritter Endpoint (App2) hat Zugriff auf alle Namespaces des OPC UA Servers und kann für Wartungs- und Engineering-Zwecke verwendet werden.

Listing 4.5 Überprüfung der zum Umschalten der LED übergebenen Nachricht

```
static UA_StatusCode switchLED(void *handle, const UA_NodeId objectId,
            size_t inputSize, const UA_Variant *input, size_t outputSize,
            UA_Variant *output) {
      UA_String* msg = (UA_String*) input[0].data;
      if (pi_mmio_init() == MMIO_SUCCESS) {
            pi_mmio_set_output(DIAGNOSELED_GPIO_PIN);
      }
      UA_String on = UA_STRING("ON");
      if (UA_String_equal(msg, &on)) {
            if (pi_mmio_init() == MMIO_SUCCESS) {
                  pi_mmio_set_high(DIAGNOSELED_GPIO_PIN);
                  ledOn = true;
            }
            return UA_STATUSCODE_GOOD;
      }
      UA_String off = UA_STRING("OFF");
      if (UA_String_equal(msg, &off)) {
            if (pi_mmio_init() == MMIO_SUCCESS) {
                  pi_mmio_set_low(DIAGNOSELED_GPIO_PIN);
                  ledOn = false;
            }
            return UA_STATUSCODE_GOOD;
      }
      UA_LOG_INFO(logger, UA_LOGCATEGORY_SERVER, "Wrong message");
      return UA_STATUSCODE_BADMETHODINVALID;
}
```

4.6. Umsetzung der Anforderungen an Verwaltungsschalen laut Plattform I4.0 in der openAAS Demo

Im Rahmen der Analyse ist es erforderlich, die Verwaltungsschale der RWTH Aachen hinsichtlich der umgesetzten Eigenschaften und Funktionen zu betrachten. Als Orientierung dienen hierzu die in Abschnitt 3.4 auf Seite 31 genannten Anforderungen und Definitionen der Plattform Industrie 4.0 sowie das UML-Diagramm aus Abbildung 4.1. Inwiefern die in dieser Arbeit identifizierten Anforderungen an eine Verwaltungsschale in der openAAS Implementierung umgesetzt sind, ist Tabelle 4.2 zu entnehmen.

Nr.	Umgesetzt?	Anmerkungen
1	Ja	Informationen sind aufgeteilt in einen Bereich, der das Asset an sich betrifft, und einen anderen, der Metainformationen enthält.
2	Ja	Die Verwaltungsschalen (Asset-Knoten) enthalten Informationen (ID, Kaufdatum) über das Asset.
3	Ja	Die Teilmodelle beschreiben Informationen zur Verwendung des Assets (z. B. kleinste Eingangsspannung beim Feuchtigkeitssensor).
4	Bedingt	Komponenten-Manager und Manifest sind nicht direkt in den Verwaltungsschalen der Demo umgesetzt. Vielmehr übernimmt der OPC UA Server die Aufgaben der Informationsbereitstellung (Komponenten-Manager) und die Daten entsprechend des Informationsmodells stellen das Manifest dar. Die Verwaltungsschale besitzt als zentrales Element Manifest und Komponenten-Manager.
5	Ja	Zugriff auf die Daten erfolgt über OPC UA und damit service-orientiert.
6	Nein	In den Verwaltungsschalen der Demo werden keine Anwendungsfälle für die Assets angegeben.
7	Ja	Sichten sind in der Property Demo durch unterschiedliche Endpunkte im OPC UA Server realisiert.
8	Ja	-
9	Ja	-
10	Bedingt	Fabriken als Assets sind der Demo so nicht behandelt, eine Verschachtelung von Verwaltungsschalen ist aber möglich.
11	Nein	In der Demo werden nur die tatsächlichen Instanzen der Sensoren bzw. der LED betrachtet.
12	Ja	Die implementierten Verwaltungsschalen referenzieren sich gegenseitig.
13	Ja	In den Verwaltungsschalen werden bereits Eigenschaften wie Fabrik-, Produkttyp oder Herstellername behandelt.
14	Unklar	Die Merkmale zu den Verwaltungsschalen der Demo werden in den beigefügten XML-Dateien definiert. Dabei ist keine verlässliche generische Merkmalsmenge erkennbar.

Tabelle 4.2.: In der Demo-Implementierung umgesetzte Anforderungen (Nummerierung bezieht sich auf Tabelle 3.1 auf Seite 30)

Listing 4.6 Überprüfung der übergebenen Sender- und Empfängeradresse vor dem Umschalten der LED

```
#define LED_REC_NAME "http://boschrexroth.de/sectorxx/Inventory001_MultiSensor_Lamp"

...

static UA_StatusCode diagnosisMethod(void *handle, const UA_NodeId objectId,
            size_t inputSize, const UA_Variant *input, size_t outputSize,
            UA_Variant *output) {
      if (inputSize != 3) {
            return UA_STATUSCODE_BADMETHODINVALID;
      }

      if (input[0].type != &UA_TYPES[UA_TYPES_STRING]
                  && input[1].type != &UA_TYPES[UA_TYPES_STRING]) {
            return UA_STATUSCODE_BADMETHODINVALID;
      }
      UA_String* receiver = (UA_String*) input[1].data;
      UA_String rec = UA_STRING(LED_REC_NAME);

      if (!UA_String_equal(receiver, &rec)) {
            UA_LOG_INFO(logger, UA_LOGCATEGORY_SERVER, "Wrong receiver");
            return UA_STATUSCODE_BADMETHODINVALID;
      }
      return switchLED(handle, objectId, 1, &input[2], 0, NULL);
}
```

5. Implementierung

Im Folgenden wird zunächst ein Anwendungsfall für den Einsatz einer Verwaltungsschale im Sinne dieser Arbeit definiert. Darauf folgt der Entwurf eines Informationsmodells, das den dabei relevanten Anforderungen entspricht. Dieses Modell dient als Grundlage für den OPC UA Server, der am Ende die Verwaltungsschale repräsentieren soll. Am Ende dieses Kapitels steht dazu eine Gegenüberstellung zweier Bibliotheken für die Implementierung eines OPC UA Servers. Anhand einiger Kriterien wird begründet, welche der beiden im Rahmen dieser Arbeit zum Einsatz kommt.

5.1. Anwendungsszenario

Die Industrie setzt bei der Umsetzung von Industrie 4.0 auf das Kommunikationsprotokoll OPC UA um Maschinendaten semantisch zu beschreiben. Gerade durch die, auf der Hannover Messe 2016 vorgestellte, Erweiterung von OPC UA um Publish-/Subscribe Mechanismen [OPC17] für einen optimierten Nachrichtenfluss in Sensor- und Cloud-Netzwerken gewinnt OPC UA enorm an Bedeutung [Hap04]. Auch die Erweiterung von OPC UA um deterministische Echtzeitkommunikation (TSN) birgt das Potential zur Umsetzung vielfältiger Anwendungsfälle im industriellen Umfeld, die harte Echtzeitanforderungen benötigen. Deswegen soll die Datenschnittstelle zu Maschinendaten in Form einer Verwaltungsschale anhand des OPC UA Stacks prototypisch umgesetzt werden. Die untersuchte Implementierung der RWTH Aachen bietet dabei schon einige Hinweise, wie eine solche Verwaltungsschale aussehen könnte. Den Kern eines OPC UA Servers bildet das Informationsmodell [OPC15b]. Dieses muss vorab definiert werden. Als Basis hierfür dient das Metamodell, welches in der OPC UA Spezifikation beschrieben ist [OPC15a]. Es beschreibt Knotenklassen und Referenztypen, die bei der Erstellung von Informationsmodellen verwendet werden können. Damit stehen auch Werkzeuge bereit, die von der Plattform I4.0 beschriebenen Teilmodelle für Verwaltungsschalen zu modellieren [Bed+17, S. 5]. Da in dieser Arbeit ein mögliches digitales Abbild geschaffen werden soll, das insbesondere Analyseanforderungen zu Forschungszwecken gerecht wird, steht die Anforderung der Repräsentation industrieübergreifender Standards (wie eCl@ss[1] oder Digital Factory[2]) durch Teilmodelle nicht im Vordergrund. Stattdessen soll gezeigt werden, wie eine solche Datenschnittstelle für Maschinendaten anhand von OPC UA umgesetzt wird. Dafür

[1] https://www.eclass.eu/

[2] IEC TS 62832-1:2016 (https://webstore.iec.ch/publication/33023)

reicht ein selbst definiertes minimalistisches Informationsmodell auf Basis der in dieser Arbeit recherchierten und in Abschnitt 3.4 auf Seite 31 zusammengefassten Anforderungen aus. Auch die Bereitstellung verschiedener Datensichten durch Views wird vernachlässigt, da der Fokus dieser Arbeit auf der Bereitstellung möglichst aller Daten eines Systems für Analysezwecke liegt. Ein nachgelagertes Ziel ist es, die implementierte Datenschnittstelle als Datenquelle zum Bau eines echtzeitnahen digitalen Abbilds eines Gegenstands zu nutzen. Dadurch wird es möglich die durch OPC UA maschinenlesbaren und semantisch beschriebenen Daten z. B. für die Datenintegration zu verwenden und Analyseanforderungen im Produktionsbereich zu adressieren. Eine Möglichkeit zur Integration wäre ein Message Bus der bei den Datenformaten auch OPC UA anbietet. Dies macht die Modellierung und Umsetzung von Analysepipelines für die Datenvorverarbeitung möglich, sodass Analysen effizienter durchgeführt werden können.

In einem denkbaren Anwendungsszenario kann eine Maschine, die beispielsweise für das Härten eines Bauteils zuständig ist, in der Produktionslinie einer Fabrik betrachtet werden. Zur Überwachung während des Betriebes verfügt die Maschine über zwei Sensoren. Ein Lichtsensor kontrolliert als vorgeschaltete Lichtschranke die Materialzufuhr der Maschine, um den Eingang eines zu bearbeitenden Werkstücks festzustellen. Außerdem wird die Betriebstemperatur von einem Temperatursensor überwacht. Zur Untersuchung der Produktionsabläufe auf Unregelmäßigkeiten, Fehlerfälle und kausale Zusammenhänge wird eine Schnittstelle benötigt, die die Daten der Maschine und ihrer Sensoren zugänglich macht. Diese Schnittstelle soll zusätzlich den Anforderungen von Industrie 4.0 entsprechen. Es wird also eine Verwaltungsschale für die Maschine benötigt, die Zugriff auf die Daten der Maschine bietet. Dazu gehören einerseits die Überwachungsdaten während des Betriebs, d. h. die Messwerte angeschlossener Sensoren. Andererseits sollen auch Herstellerinformationen in die Analyse mit einbezogen werden. Dafür werden in der Verwaltungsschale die Name des Herstellers, die Modellbezeichnung in Form einer Seriennummer und zusätzlich der Standort abgelegt. So können auch Zusammenhänge zwischen Teilen aus gleichen Baureihen oder Einflüssen in bestimmten Bereichen der Fabrik hergestellt werden. Unter diesen Voraussetzungen bietet sich die Verwaltungsschale auch dazu an, bei einer potentiellen Erstellung eines digitalen Zwillings der gesamten Fabrik mit einbezogen zu werden.

5.2. Definition eines Informationsmodells

Für die Implementierung eines OPC UA Servers ist zunächst die Definition des zugrundeliegenden Informationsmodells erforderlich. Dazu versucht diese Arbeit, sich trotz einiger Vereinfachungen an den Empfehlungen aus Anhang A zur Spezifikation des Address Space Models [OPC15a, S. 75 f.] und an den Definitionen bezüglich Typen, Instanzen und deren Attributen [OPC15b, S. 3 ff.] zu orientieren. Unter Berücksichtigung der zuvor identifizierten Anforderungen an eine Verwaltungsschale im Sinne dieser Arbeit ergibt sich das in Abbildung 5.1 dargestellte Informationsmodell.

Den Einstiegspunkt in den Adressraum des OPC UA Servers bietet der *„Root“*-Knoten. Wie im Anwendungsszenario die Maschine, gruppiert er die Knoten des Temperatur- und des Lichtsensors. Die Sensorobjekte verwenden dabei den von der OPC Foundation definierten Basistyp für Objekte (*BaseObjectType*). Jeder Sensor verfügt über gewisse Stammdaten und Messwerte. Diese werden im Informationsmodell durch Objektknoten vom Typ *FolderType* in die jeweils passende Kategorie eingeordnet. Zu den Stammdaten eines Sensors gehören dessen Hersteller, eine Modellbezeichnung in Form einer Seriennummer sowie der Standort des Sensors. Diese Eigenschaften sind in Form von *Property*-Knoten hinterlegt. Der Name des Herstellers wird dabei als String abgespeichert, genauso wie die Seriennummer, da hier auch Kombinationen aus Zahlen und Buchstaben möglich sind. Die Beschreibung des Standortes erfolgt anhand von GPS-Koordinaten. Es handelt sich dabei nur um eine beispielhafte Auswahl an Stammdaten, die durchaus um weitere Punkte ergänzt werden kann (in Abbildung 5.1 durch „...“ angedeutet). Messwerte eines Sensors werden als Variablen abgelegt. Um Missverständnissen bei der Interpretation des Wertes vorzubeugen, ist die Maßeinheit des Messwertes mit anzugeben. Dies geschieht wiederum unter der Zuhilfenahme eines *Property*-Knotens. Zusammengefasst besteht dieses Modell also aus einer Verwaltungsschale für den Temperatur- und einer weiteren für den Lichtsensor.

Umgesetzte Anforderungen

Mit Hilfe dieses Informationsmodells ist es möglich, alle Bestandteile der Maschine aus dem Anwendungsfall eindeutig zu adressieren. Grund dafür ist die Zuweisung eindeutiger Identifikationsnummern für Knoten innerhalb eines OPC UA Servers. In den Knoten sind Informationen zu den Assets, nämlich Stammdaten und Messwerte der beiden Sensoren, gespeichert und abrufbar. Diese werden durch die Gruppierung in entsprechende Objekte vom Type *FolderType* auch voneinander getrennt. Eine Erweiterung um zusätzliche Merkmale ist im Informationsmodell berücksichtigt. Die dort aufgeführten Eigenschaften werden jedoch als verbindliche Angaben vorausgesetzt. Damit sind die meisten der Anforderungen aus Tabelle 3.1 auf Seite 30 in der implementierten Verwaltungsschale umgesetzt. Eine Verschachtelung von Verwaltungsschalen wird in dieser Implementierung nicht betrachtet, ist aber theoretisch möglich. Dazu könnte eine weitere Ebene über den Sensoren eingefügt werden. Diese enthielte dann ein zusätzliches Objekt für die Maschine, das als Komponenten die Stammdaten und statt der Messwerte z. B. die Objekte Temperatur- bzw. Lichtsensor enthält. Des Weiteren fordert diese Arbeit eine Unterscheidung zwischen Stamm- und Sensordaten eines Assets. Durch die Struktur des definierten Informationsmodells ist die dafür erforderliche Möglichkeit zur Differenzierung gegeben.

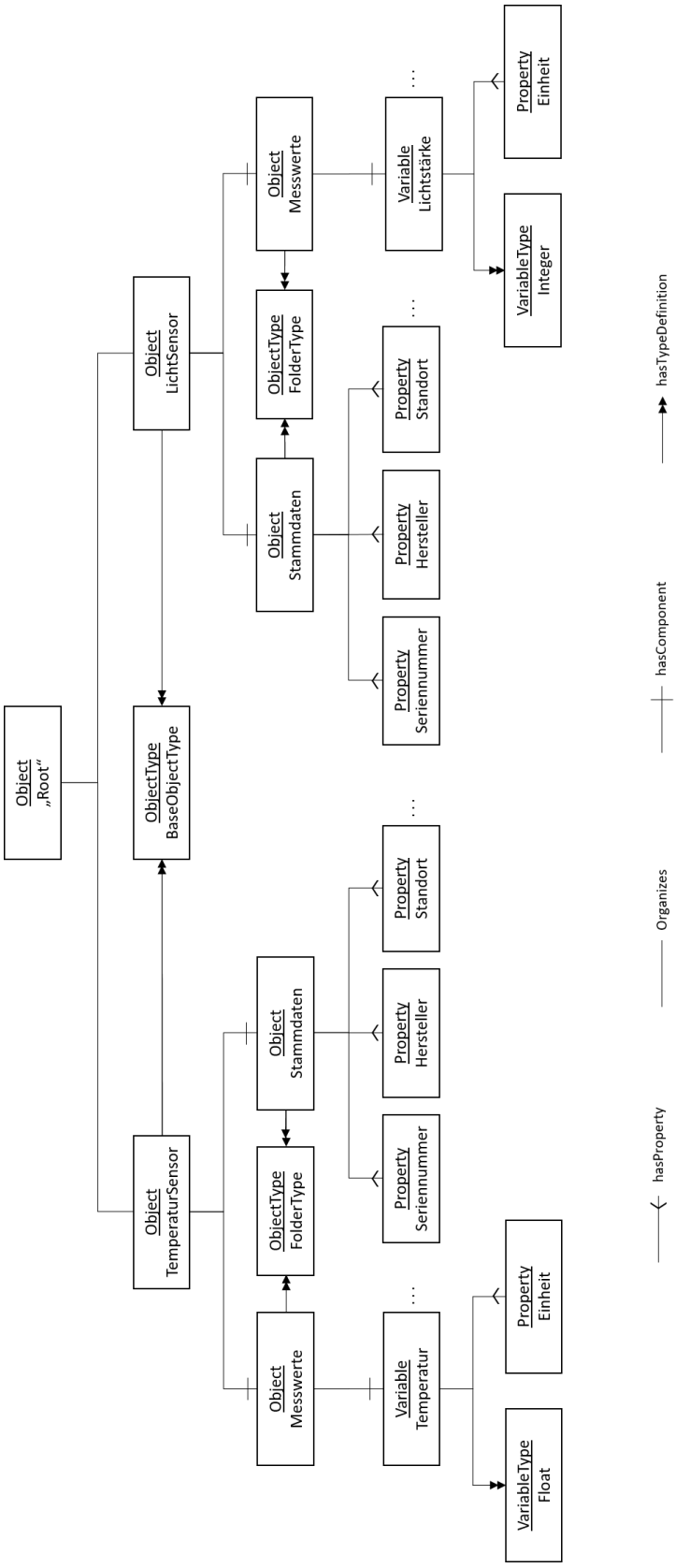

Abbildung 5.1: Informationsmodell für die Implementierung einer Verwaltungsschale entsprechend des Anwendungsszenarios aus Abschnitt 5.1

5.3. Verfügbare Technologien

Dieser Abschnitt bietet einen Überblick über die Auswahl der für die Implementierung verwendeten Technologien. Dazu gehört die Hardware, auf der das fertige Programm ausführbar sein soll, und die Bibliothek, welche für die Umsetzung eines OPC UA Servers eingesetzt wird. Zu den betrachteten Technologien werden jeweils einige Eigenschaften vorgestellt. Auf deren Grundlage wird daraufhin eine begründete Auswahl getroffen.

5.3.1. Hardwareplattformen

Hier werden mit dem Bosch XDK und dem Raspberry Pi zwei Hardware Plattformen vorgestellt, die sich für den Einsatz in der Industrie eignen. Es werden deren technische Eigenschaften untersucht und einige Funktionalitäten gegenübergestellt.

Bosch XDK

Mit dem XDK110[3] bietet die Firma Bosch Connected Devices and Solutions eine Entwicklungsplattform für Anwendungen im Bereich Internet of Things (IoT). Damit sind qualifiziert sich das Gerät auch für den Einsatz in intelligenten Fabrikumgebungen. Es ist neben Modulen für Bluetooth- und WLAN-Verbindungen auch Sensoren für Bewegung, räumliche Lage (Gyroskop), magnetische und akustische Signale, Feuchtigkeit, Druck, Temperatur und Licht ausgestattet. Außerdem verfügt der XDK über eine eingebaute Lithium-Ionen-Batterie, zwei programmierbare Knöpfe sowie drei steuerbare LEDs. Zusammen mit dem verbauten 32-Bit Mikrocontroller (ARM Cortex M3), 1 MB an internem Flash-Speicher und einem Arbeitsspeicher (RAM) von 128 kB sind diese Komponenten in einem recht kompakten Gehäuse ($60 * 40 * 22$ mm) untergebracht. Zum Aufladen der Batterie oder für den Betrieb am Netz liegt dem XDK ein microUSB-Kabel bei. Außerdem verfügt er über einen Steckplatz für eine microSD-Karte, um den Speicher zu erweitern. Die hier aufgeführten Angaben entstammen der Hardware-Spezifikation, welche auf der Homepage zum XDK zu finden ist.

Durch die kompakten Abmessungen, die integrierte Batterie und die üppige Ausstattung eignet sich der XDK beispielsweise auch für die Überwachung von Gegenständen und Maschinen einer Fabrik. Anwendungen und Programme zur Ausführung auf dem XDK werden unter Verwendung einer angepassten Version der Entwicklungsumgebung Eclipse, der XDK Workbench, in C und C++ codiert. Es werden einige Beispielanwendungen sowie Treiber für die Komponenten des XDK mitgeliefert. Mit Hilfe der Workbench können diese kompiliert und per USB auf den XDK überspielt werden. Dort werden sie unter einem Echtzeitbetriebssystem ausgeführt, das auf FreeRTOS[4] basiert. Da diese Arbeit als Zielplattform ursprünglich

[3] https://xdk.bosch-connectivity.com/
[4] http://www.freertos.org/

den XDK vorgesehen hatte, wurden damit einige Tests durchgeführt. Zur Simulation einer M2M-Kommunikation wurde folgendes Szenario zwischen zwei XDKs entworfen:

Per Knopfdruck auf einem XDK soll beim jeweiligen Gegenstück eine der LEDs umgeschaltet werden. Der dazu benötigte Nachrichtenaustausch findet mittels MQTT statt. Über ein lokales WLAN-Netzwerk verbinden sich die XDKs mit einem MQTT Broker, der entsprechend Anhang B auf Seite 65 auf einem Windows-System im selben Netzwerk eingerichtet ist. In Folge eines Knopfdrucks sendet der betroffene XDK eine Nachricht mit diesem Ereignis an den Broker, der sie an den anderen XDK, welcher als Subscriber für die zugehörige Warteschlange registriert ist, weiterleitet. Der Empfänger verarbeitet die erhaltene Nachricht und schaltet als Ergebnis eine LED ein bzw. aus.

Des Weiteren sollte die Verwaltungsschale der RWTH Aachen auf Kompatibilität mit dem XDK geprüft werden. Allerdings ist die dort eingesetzte OPC UA Implementierung open62541[5] auf dem XDK nicht lauffähig. Durch Versuche, das obige Szenario unter Verwendung von open62541 anstatt MQTT zu realisieren, ließ sich das Problem genauer identifizieren. Für die Kommunikation über TCP, wie sie bei OPC UA vorgesehen ist, werden in open62541 Bibliotheken verwendet, die nicht vom Betriebssystem unterstützt werden. Daher war eine Evaluierung der openAAS auf dem XDK nicht möglich.

Raspberry Pi

Der Raspberry Pi[6] ist ein aus einer Platine bestehender Computer. Er ist in verschiedenen Ausführungen und Modellen erhältlich. Der in dieser Arbeit betrachtete Raspberry Pi 3 (Model B) stammt aus der dritten Produktgeneration und ist mit einem 64-Bit Prozessor mit vier Kernen sowie 1 GB Arbeitsspeicher (RAM) ausgestattet. Außerdem verfügt er über ein Bluetooth- und ein WLAN-Modul. Es sind keinerlei Sensoren, steuerbare LEDs oder ähnliche Komponenten fest verbaut. Stattdessen bietet der Raspberry Pi die Möglichkeit, solche Bauteile über GPIO (general purpose input/output) Pins an das Gerät anzuschließen. Die entsprechenden Bauteile können separat erworben werden. Bei der Wahl des Betriebssystems kann der Anwender je nach Anwendungsfall aus einer Vielzahl an Angeboten das jeweils passende auswählen. Die meist auf Linux basierenden Systeme werden einfach auf einer microSD-Karte mit ausreichend Speicherplatz installiert und diese in den entsprechenden Steckplatz des Rasperry Pi eingelegt. Im Rahmen dieser Arbeit kommt das von der Raspberry Pi Foundation empfohlene Betriebssystem Raspbian, eine angepasste Version der Linux-Distribution Debian, zum Einsatz. Über die vier USB 2.0- und den HDMI-Anschluss können Monitore und Peripheriegeräte angeschlossen werden. Zudem verfügt der Raspberry Pi über eine LAN-Schnittstelle sowie über einen Audioausgang. Die Stromzufuhr erfolgt mit einem microUSB-Netzteil.

[5] https://open62541.org
[6] http://www.raspberrypi.org

Mit dem Linux-System werden bereits einige Softwarepakete mitgeliefert. Zusätzliche Software kann mit Hilfe des Advanced Packaging Tools (APT), dem Paketmanager in Debian bzw. Raspbian, installiert werden. Dadurch steht für Implementierungszwecke eine große Auswahl an Programmiersprachen bereit, mit denen zum Teil auch direkt auf dem Raspberry Pi entwickelt werden kann.

Auswahl und Begründung

Für die Implementierung einer Verwaltungsschale wird hier als Hardwarebasis der Raspberry Pi gewählt. Grund dafür ist die größere Flexibilität in Sachen anschließbarer Sensoren und anderer Komponenten. Die fehlende Möglichkeit zum Batteriebetrieb und die größeren Abmessungen (ca. $93 * 66 * 35$ mm) schränken zwar die Anwendung in manchen Bereichen ein, allerdings sind diese Aspekte für die Ziele der Arbeit nicht relevant. Außerdem bietet das umfangreichere Betriebssystem des Raspberry Pi entscheidende Vorteile. So können für die Implementierung beliebige Programmiersprachen verwendet werden. Dadurch ist auch der Einsatz von leichtgewichtigen Skriptsprachen möglich und eine Vorkompilierung des Quellcodes ist nicht weiter nötig.

5.3.2. Bibliotheken für OPC UA

Für die Implementierung im Rahmen dieser Arbeit ist es erforderlich, kostenfrei verfügbare OpenSource-Lösungen zu verwenden. Da die Entwicklung einer kleineren Anwendung mit Hilfe von Rapid Prototyping vorgesehen ist, empfiehlt sich beim Programmieren der Einsatz von Skriptsprachen. Gerade bei der Entwicklung von Prototypen ist es von Vorteil, dass auf die explizite Deklaration von Variablen verzichtet wird. So können wenig komplexe Funktionen schnell implementiert werden. Ein weiterer Vorteil ist, dass Skripte ohne vorherige Kompilierung ausführbar sind. Durch die interpretative Ausführung von Skripten entfällt nämlich dieser Zwischenschritt [RP06, S. 554 f.]. Dieser Aspekt ist in Anbetracht der eingeschränkten Rechenleistung der in dieser Arbeit verwendeten Hardware von besonderem Interesse. Aufgrund der oben genannten Aspekte wurde die üppige Auswahl an verfügbaren Bibliotheken für OPC UA auf die folgenden beiden eingegrenzt:

Node OPC UA

Node OPC UA[7] ist ein OPC UA Stack, der mit JavaScript implementiert wurde. Dabei kommt die Laufzeitumgebung NodeJS als Erweiterung von JavaScript zum Einsatz. Sie stellt hier die Grundlage für die Kommunikation über ein Netzwerk. Auf der Homepage von Node OPC UA steht je ein Beispiel für die Implementierung eines OPC UA Clients und eines OPC UA

[7] https://node-opcua.github.io/

Listing 5.1 Aufruf in Python OPC UA zum Einbinden von Knotendefinitionen aus einer XML-Datei

```
self.server = Server()

self.server.import_xml(<model_filepath>)
```

Servers bereit. Die Verwendung der Code-Beispiele ist zusätzlich in Form von kurzen Tutorials erläutert. Außerdem wird auf eine Schnittstellendokumentation verwiesen, die Informationen zu implementierten Klassen und deren Funktionen enthält.

Python OPC UA

Bei Python OPC UA[8] handelt es sich um einen OPC UA Stack in der Programmiersprache Python geschrieben ist. Unterstützt werden laut Angaben der Entwickler sowohl Python 2 als auch Python 3. Im GitHub Repository von Python OPC UA werden einige Beispiele für die Implementierung von OPC UA Clients und OPC UA Servern bereitgestellt, die verschiedene Funktionalitäten hervorheben. Dabei ist außer den Kommentaren im Quelltext keine zusätzliche Beschreibung vorhanden. Eine Dokumentation der implementierten Klassen und ihrer Funktionen steht jedoch zur Verfügung. Außerdem werden Werkzeuge für die Kommandozeile mitgeliefert, die einige Client-Funktionen bereitstellen und einen Demo OPC UA Server starten. Des Weiteren wird auf einen OPC UA Client mit grafischer Benutzeroberfläche verwiesen, der zum übergeordneten Projekt FreeOpcUa gehört und ebenfalls auf GitHub verfügbar ist.

Auswahl und Begründung

Die genannten Bibliotheken erlauben es, die Knoten des Adressraums für OPC UA Server im Programmcode zu initialisieren. Python OPC UA bietet außerdem die Möglichkeit, mit Hilfe der in Listing 5.1 dargestellten Codezeile XML-Dateien einzubinden, die der vorgegebenen Struktur für ein Informationsmodell[9] entsprechen. Durch diesen Aufruf wird ein Parser für die importierte XML-Datei gestartet, der die dort hinterlegten Informationen einliest und zur weiteren Verwendung im Programmcode in Form eines Arrays bereitstellt. Besonders vorteilhaft ist hierbei, dass sich Änderungen am Informationsmodell durch entsprechende Anpassungen der XML-Datei einpflegen lassen. Der eigentliche Programmcode muss dafür nicht verändert werden. Dadurch ergibt sich nicht nur eine größere Flexibilität, zudem sind so definierte Informationsmodelle auch in anderen Serverinstanzen wiederverwendbar.

[8] https://github.com/FreeOpcUa/python-opcua
[9] https://opcfoundation.org/UA/2011/03/UANodeSet.xsd

Für die Implementierung eines OPC UA Servers wurde aufgrund der oben beschriebenen Eigenschaften die Bibliothek Python OPC UA gewählt. Das in Abschnitt 5.2 definierte Informationsmodell wird dabei in Form einer XML-Datei umgesetzt. Somit ist es nicht nur leicht einzubinden, sondern kann bei Bedarf um zusätzliche Knoten ergänzt werden, ohne dass Änderungen am Programmcode erforderlich sind. Als Entwicklungsumgebung für Python wird hier PyCharm (JetBrains) für Windows benutzt.

6. Zusammenfassung und Ausblick

In dieser Arbeit sollte eine Verwaltungsschale für Industrie 4.0 Komponenten evaluiert und anschließend implementiert werden. Dazu wurde zunächst eine Einführung in das Thema Industrie 4.0 vor allem im Hinblick auf Definitionen und wichtige Begrifflichkeiten im Zusammenhang mit Verwaltungsschalen geliefert. Außerdem wurde die dabei zentrale Technologie OPC UA vorgestellt.

In den Anforderungen wurden einige Rahmenbedingungen für die Evaluierungs- und Implementierungsphase festgelegt und die wichtigsten Aspekte bezüglich Verwaltungsschalen im Sinne dieser Arbeit zusammengefasst.

Darauf folgte mit der Analyse einer von der RWTH Aachen implementierten Demoanwendung für Verwaltungsschalen der Hauptteil dieser Arbeit. Es wurden deren zugrundeliegende Modelle vorgestellt sowie deren Umsetzung im zugehörigen OPC UA Server untersucht. Anschließend wurden Möglichkeiten beschrieben, auf die gespeicherten Daten zuzugreifen. In einer Gegenüberstellung der Implementierung mit den gegebenen Anforderungen ließ sich erkennen, dass ein Großteil der gewünschten Aspekte erfüllt wird. Allerdings leidet dadurch das Beispiel an einer recht hohen Komplexität, was die nur spärlich vorhandene Dokumentation kaum ausgleichen kann.

Zum Ende der Arbeit wurde ein Modell entworfen, das als Grundlage für die Umsetzung einer eigenen Implementierung der Verwaltungsschale dient. Dabei wurden die gestellten Rahmenbedingungen und Basismodelle der OPC UA Architektur aufgegriffen. Anschließend wurden zwei verwendbare Hardwareplattformen genannt, von denen eine für die Implementierung ausgewählt wurde. Außerdem erfolgte die Auswahl einer aus zwei vorgestellten Bibliotheken für die Realisierung des Modells in Form eines OPC UA Servers.

Ausblick

Es bleibt abzuwarten, in welcher Form sich Konzepte wie die Verwaltungsschale in Industrie 4.0 letztendlich durchsetzen werden. Die Plattform I4.0 versucht in Zusammenarbeit mit weiteren internationalen Gremien, die ähnliche Standards entwickelt haben, Gemeinsamkeiten zu identifizieren und einen Konsens zu finden. So werden beispielsweise das Referenzarchitekturmodell für Industrie 4.0 (RAMI4.0) und das Pendant, die Industrial Internet Referenzarchitektur (IIRA), des Industrial Internet Consortiums miteinander abgeglichen. Als treibende Kraft wird sich

dabei voraussichtlich OPC UA durchsetzen. Vor allem aufgrund der jüngsten Ergänzungen um Kommunikation in Echtzeit und bessere Unterstützung für Cloud-basierte Systeme.

Das mögliche Aufgabenspektrum für Verwaltungsschalen reicht dabei von der reinen Speicherung von Daten bis hin zur Umsetzung von Steuerungs- und Konfigurationsabläufen in Plug-and-Produce-Umgebungen. In dieser Arbeit wurde das Spektrum auf die Funktion als Schnittstelle zur Bereitstellung von Informationen zu Stammdaten sowie Messwerten von Sensoren für Analysezwecke fokussiert. Eine Erweiterung um zusätzliche Datensichten in anschließenden Entwicklungsarbeiten ist aber durchaus denkbar. Ebenso verhält es sich mit der Anbindung von Teilmodellen, die gewisse Merkmalsmengen für Eigenschaften z. B. bezüglich Sicherheitsaspekten der Verwaltungsschale definieren können.

A. Übersicht über Objekte, Variablen und Werte aus der openAAS-Demo

Die folgenden Abbildungen sollen einen Überblick über die gespeicherten Informationen und Werte aus der openAAS-Demo liefern. In ähnlicher Form wiederkehrende Einträge, wie z. B. „PropertyName“ und „PropertyRef“, wurden zur besseren Übersichtlichkeit größtenteils weggelassen. Da es sich bei den URIs des „PropertyRef“-Wertes meist um simulierte Adressen handelt, die bei einem Aufruf im Browser ins Nichts führen, werden dadurch auch keine für die Verwendung des Servers relevanten Informationen vernachlässigt. Abbildung A.1 zeigt jedoch beispielhaft alle Eigenschaften in voller Ausführung, da der Multi Sensor selbst nur wenige untergeordnete Knoten besitzt und dessen Darstellung sich deshalb nicht so umfangreich gestaltet wie die der Sensoren bzw. der LED.

Multisensor				
Asset	AssetId	PropertyName	SerialNumber	
		PropertyRef	IdSpecification	http://acplt.org/propertyTypes/AssetId
			IdType	1
		Value	IdSpecification	1604220DF_A
			IdType	2
	PurchaseDate	PropertyName	Puchasedate	
		PropertyRef	IdSpecification	http://acplt.org/propertyTypes/PurchaseDate
			IdType	1
		Value	20.07.2016	
AssetAdministrationShell	(AS)Id	PropertyName	Null	
		PropertyRef	IdSpecification	http://acplt.org/openAAS/AdministrationShellId
			IdType	1
		Value	IdSpecification	http://boschrexroth.de/multisensor001
			IdType	1
	(AS)Type	PropertyName	Administration Shell Type	
		PropertyRef	IdSpecification	http://acplt.org/propertyTypes/AdministrationShellType
			IdType	1
		Value	0	
	PartialModels		-	
Subshells	• MultiSensor_Humidity • MultiSensor_Lamp • MultiSensor_Temperature			

Abbildung A.1.: Eigenschaften und Stammdaten des Multi Sensors als übergeordnete Verwaltungsschale

Multisensor_Humidity				
...				
http://acplt.org/models/Humidity (Partial Model)	MeasurementModel	Accuracy		
		ExpressionLogic	4	Unit	Percent
		ExpressionSemantic	1	Value	2
		Accuracy		
		ExpressionLogic	Null	Unit	Percent
		ExpressionSemantic	1	Value	-2
		Drift		
		ExpressionLogic	2	Unit	Percent RH Per Year
		ExpressionSemantic	1	Value	0.5
		Measurement		
		PropertyName	Humidity	
		Unit	Percent RH	
		Value	<measured value goes here>	
		MeasurementRange		
		ExpressionLogic	Null	Unit	Percent RH
		ExpressionSemantic	1	Value	0
		MeasurementRange		
		ExpressionLogic	4	Unit	Percent RH
		ExpressionSemantic	1	Value	99.9
		Repeatability		
		ExpressionLogic	4	Unit	Percent RH
		ExpressionSemantic	1	Value	0.3
		Repeatability		
		ExpressionLogic	Null	Unit	Percent RH
		ExpressionSemantic	1	Value	-0.3
		Resolution		
		ExpressionLogic	2	Unit	Percent RH
		ExpressionSemantic	1	Value	0.1
		Response		
		ExpressionLogic	4	Unit	S
		ExpressionSemantic	1	Value	5
		Sluggish		
		ExpressionLogic	4	Unit	Percent RH
		ExpressionSemantic	1	Value	0.3
	TypeDescription	CountryOfOrigin		
		Value	China	
		Height		
		ExpressionLogic	2	Unit	mm
		ExpressionSemantic	1	Value	7
		LowInputVoltage		
		ExpressionLogic	4	Unit	VDD
		ExpressionSemantic	1	Value	30%
		LowInputVoltage		
		ExpressionLogic	Null	Unit	VDD
		ExpressionSemantic	1	Value	Null
		ManufacturerName		
		Value	Aosong	
		MaximumSupplyVoltage		
		ExpressionLogic	2	Unit	V
		ExpressionSemantic	Null	Value	5.5
		ProductName		
		Value	HumiditySensor	
		Test12		
		Value	Null	

Abbildung A.2.: Teilmodelle des Feuchtigkeitssensors

Multisensor_Temperature				
...				
http://acplt.org/models/Temperature (Partial Model)	MeasurementModel	Accuracy		
		ExpressionLogic	4	Unit	C
		ExpressionSemantic	1	Value	1
		Accuracy		
		ExpressionLogic	Null	Unit	C
		ExpressionSemantic	1	Value	-1
		Drift		
		ExpressionLogic	2	Unit	C/y
		ExpressionSemantic	1	Value	0.3
		Measurement		
		PropertyName	Temperature		
		Unit	C		
		Value	<measured value goes here>		
		GreatestValueOfTemperatureRange		
		ExpressionLogic	2	Unit	C
		ExpressionSemantic	1	Value	80
		Resolution		
		ExpressionLogic	2	Unit	C
		ExpressionSemantic	1	Value	0.1
		Response		
		ExpressionLogic	4	Unit	S
		ExpressionSemantic	1	Value	10
		SamplingPeriod		
		ExpressionLogic	Null	Unit	S
		ExpressionSemantic	1	Value	2
	TypeDescription	CountryOfOrigin		
		Value	China		
		Height		
		ExpressionLogic	2	Unit	mm
		ExpressionSemantic	1	Value	7
		Length		
		ExpressionLogic	2	Unit	mm
		ExpressionSemantic	1	Value	25
		ManufacturerName		
		Value	Aosong		
		MaximumSupplyVoltage		
		ExpressionLogic	2	Unit	V
		ExpressionSemantic	Null	Value	5.5
		MeasurementRange		
		ExpressionLogic	Null	Unit	C
		ExpressionSemantic	1	Value	-40
		MeasurementRange		
		ExpressionLogic	4	Unit	C
		ExpressionSemantic	1	Value	80
		ProductName		
		Value	Temperature Sensor		
		RecommendedSupplyVoltage		
		ExpressionLogic	2	Unit	V
		ExpressionSemantic	Null	Value	5
		SupplyVoltageRange		
		ExpressionLogic	4	Unit	V
		ExpressionSemantic	Null	Value	5.5
		SupplyVoltageRange		
		ExpressionLogic	Null	Unit	V
		ExpressionSemantic	Null	Value	3.3

Abbildung A.3.: Teilmodelle des Temperatursensors

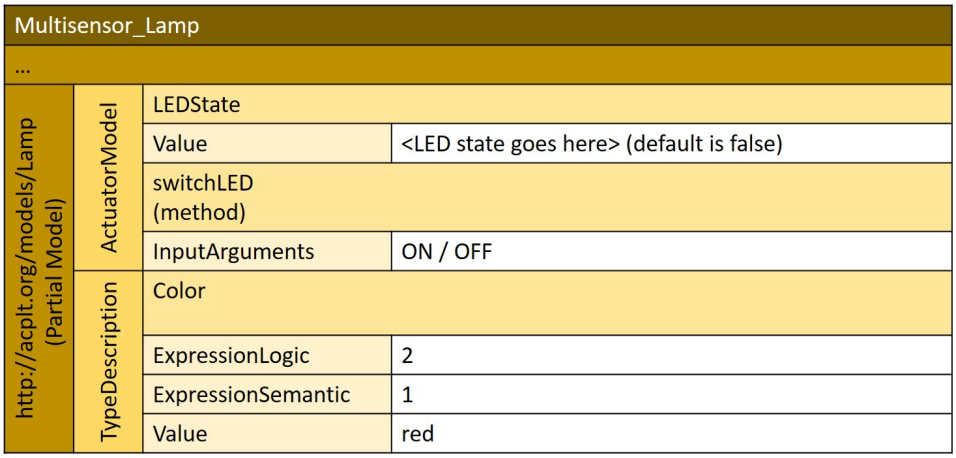

Abbildung A.4.: Teilmodelle der LED

B. Setup eines MQTT Brokers mit ActiveMQ

Im Rahmen der in Abschnitt 5.3.1 beschriebenen Tests zur Entwicklung mit einem Bosch XDK war die Einrichtung eines MQTT Brokers erforderlich. Mit Hilfe von ActiveMQ[1] lässt sich dieser recht schnell bereitstellen. Die folgenden Schritte sind nötig, um unter Microsoft Windows einen verwendbaren Broker einzurichten:

1. Auf der Homepage von Apache die aktuellste Version („apache-activemq-x.x.x-source-release.zip“) für Windows herunterladen.
2. Das heruntergeladene Archiv in ein beliebiges Verzeichnis entpacken.
3. Von einer Kommandozeile aus in das Zielverzeichnis navigieren und den ActiveMQ-Broker starten (siehe Listing B.1).
4. Zum Testen der Installation im Browser die URL http://127.0.0.1:8161/admin/ öffnen.
5. Dort mit den standardmäßigen Anmeldedaten („admin“, „admin“) einloggen.
6. Unter dem Menüpunkt „Queues“ kann nun eine neue Warteschleife angelegt und anschließend mit einem Klick auf „Send to“ getestet werden.
7. Ist die Installation funktionsfähig, so können sich Publisher und Subscriber beim Broker registrieren und selbstständig Queues anlegen.

Zum Stoppen des ActiveMQ Brokers genügt es, im zugehörigen Konsolenfenster die Tastenkombination „Strg + C“ zu drücken.

Listing B.1 Kommandozeilenbefehle zum Start von ActiveMQ

```
cd [activemq_verzeichnis]
bin\activemq start
```

[1] siehe auch http://activemq.apache.org/getting-started.html

Literaturverzeichnis

[Ado+16] P. Adolphs, S. Auer, H. Bedenbender, M. Billmann, M. Hankel, R. Heidel, M. Hoffmeister, H. Huhle, M. Jochem, M. Kiele-Dunsche, G. Koschnick, H. Koziolek, L. Linke, R. Pichler, F. Schewe, K. Schneider, B. Waser. *Struktur der Verwaltungsschale: Fortentwicklung des Referenzmodells für die Industrie 4.0-Komponente*. Hrsg. von BMWi Öffentlichkeitsarbeit. Berlin, Apr. 2016. URL: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/struktur-der-verwaltungsschale.pdf?__blob=publicationFile&v=8 (zitiert auf S. 23, 24, 31).

[BS16] H. Banthien, D. Senff. „Plattform Idustrie 4.0: Ein Schulterschluss von Politik, Wirtschaft, Gewerkschaften und Wissenschaft“. In: *Industrie 4.0 im internationalen Kontext*. Hrsg. von C. Manzei, L. Schleupner, R. Heinze. Beuth Innovation. Berlin u. a.: VDE Verlag GmbH und Beuth Verlag GmbH, 2016, S. 134–137. ISBN: 978-3-410-26049-3, 978-3-8007-3671-3 (zitiert auf S. 21).

[Bau14] T. Bauernhansl. „Die Vierte Industrielle Revolution – Der Weg in ein wertschaffendes Produktionsparadigma“. In: *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Hrsg. von T. Bauernhansl, M. ten Hompel, B. Vogel-Heuser. SpringerLink. Wiesbaden: Springer Vieweg, 2014, S. 5–35. ISBN: 978-3-658-04681-1. DOI: 10.1007/978-3-658-04682-8_1. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-658-04682-8_1.pdf (zitiert auf S. 15).

[Bed+17] H. Bedenbender, A. Bentkus, U. Epple, T. Hadlich, M. Hankel, R. Heidel, O. Hillermeier, M. Hoffmeister, H. Huhle, M. Kiele-Dunsche, H. Kozoliek, S. Lohmann, M. Mendes, J. Neidig, F. Palm, S. Pollmeier, B. Rauscher, F. Schewe, B. Waser, I. Weber, M. Wollschlaeger. *Beziehungen zwischen I4.0-Komponenten – Verbundkomponenten und intelligente Produktion*. Techn. Ber. Berlin, Juni 2017. URL: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/beziehungen-%20i40-komponenten.pdf?__blob=publicationFile&v=3 (zitiert auf S. 49).

[Bla+16] A. Blank, H. Fleischmann, J. Franke, J. Fuchs, J. Kohl, M. Schacht. „Semantische Kommunikationsschnittstellen zur Zustandsüberwachung im Karosseriebau: Serviceorientierte Architekturen für die Maschinendiagnose in vernetzten Produktionssystemen“. In: *wt Werkstattechnik online* 106.10 (2016), S. 699–704. URL: http://www.werkstattstechnik.de/wt/get_article.php?data[article_id]=86526 (zitiert auf S. 15).

[Boc+16a] J. Bock, C. Diedrich, A. Gössling, O. Graeser, R. Hänisch, A. Kraft, O. Niggemann, F. Pethig, J. Reich, F. Vollmar, J. Wende. *Interaktionsmodell für Industrie 4.0 Komponenten - Diskussionspapier*. Hrsg. von BMWi Öffentlichkeitsarbeit. Berlin, März 2016. URL: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/interaktionsmodell-I40-komponenten.pdf?__blob=publicationFile&v=13 (zitiert auf S. 25).

[Boc+16b] J. Bock, C. Diedrich, R. Hänisch, A. Kraft, J. Neidig, O. Niggemann, F. Pethig, J. Reich, T. Schulz, J. Vialkowitsch, F. Vollmar. *Weiterentwicklung des Interaktionsmodells für Industrie 4.0-Komponenten – Diskussionspapier*. Hrsg. von BMWi Öffentlichkeitsarbeit. Berlin, Nov. 2016. URL: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/interaktionsmodell-i40-komponenten-it-gipfel.pdf?__blob=publicationFile&v=10 (zitiert auf S. 37).

[Can16] A. Canedo. „Industrial IoT Lifecycle via Digital Twins“. In: *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES '16. Pittsburgh, Pennsylvania: ACM, 2016, 29:1–29:1. ISBN: 978-1-4503-4483-8. DOI: 10.1145/2968456.2974007 (zitiert auf S. 16).

[EM11] U. Enste, W. Mahnke. „OPC Unified Architecture“. In: *at - Automatisierungstechnik* 59.7 (2011), S. 397–404. ISSN: 0178-2312. DOI: 10.1524/auto.2011.0934 (zitiert auf S. 27).

[EPA16] U. Epple, F. Palm, M. Azarmipour. *PRoMo Property Meta Model: Basic Concepts for openAAS*. Hrsg. von RWTH Aachen Lehrstuhl für Prozessleittechnik. Aachen, Okt. 2016. URL: https://github.com/acplt/openAAS/blob/master/Doc/PropertyMetaModel.pdf (zitiert auf S. 33, 38, 39, 41).

[Epp16] U. Epple. *Com4.0-Basic: Basic Models of Communication*. Hrsg. von RWTH Aachen Lehrstuhl für Prozessleittechnik. Aachen, Dez. 2016. URL: https://github.com/acplt/openAAS/blob/master/Doc/ComBasic.pdf (zitiert auf S. 33).

[Gei+11] E. Geisberger, M. V. Cengarle, P. Keil, J. Niehaus, C. Thiel, H.-J. Thönnißen-Fries. *Cyber-Physical Systems: Innovationsmotor für Mobilität, Gesundheit, Energie und Produktion*. Hrsg. von acatech - Deutsche Akademie der Technikwissenschaften. Berlin und München, Dez. 2011. URL: http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Publikationen/Stellungnahmen/POSITION_CPS_NEU_WEB_120130_final.pdf (zitiert auf S. 15).

[Gra+16] U. Graf, R. Heidel, G. Kadel, B. Kärcher, F. Mildner, L. Rauchhaupt, F. Schewe, D. Schulz. *Network-based Communication for Industrie 4.0 – Proposal for an Administration Shell*. Hrsg. von BMWi Öffentlichkeitsarbeit. Berlin, Nov. 2016. URL: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/network-based-communication-for-i40.pdf?__blob=publicationFile&v=5 (zitiert auf S. 23, 25, 31).

[Hap04] M. Happacher. *Vom Sensor bis in die Cloud*. 27.04.2016. URL: http://www.computer-automation.de/feldebene/vernetzung/artikel/129604/ (zitiert auf S. 49).

[Kae15] J. Kaeser. „From Data to Business: Neue Geschäftsmodelle deutscher Industrieunternehmen“. In: *Digitales Neuland*. Hrsg. von T. Becker. Wiesbaden: Springer Gabler, 2015, S. 23–35. ISBN: 978-3-658-09691-5. DOI: 10.1007/978-3-658-09692-2_2 (zitiert auf S. 17).

[LRP16] R. Langmann, L. F. Rojas-Peña. „A PLC as an Industry 4.0 component“. In: *Proceedings of 2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)*. Piscataway, NJ: IEEE, 2016, S. 10–15. ISBN: 978-1-4673-8246-5. DOI: 10.1109/REV.2016.7444433 (zitiert auf S. 26).

[Leh+11] S. Lehnhoff, W. Mahnke, S. Rohjans, M. Uslar. „IEC 61850 based OPC UA Communication - The Future of Smart Grid Automation“. In: *17th Power Systems Computation Conference*. PSCC ’11. Stockholm, 2011. URL: https://www.researchgate.net/publication/267561083_IEC_61850_based_OPC_UA_communication_-_The_future_of_smart_grid_automation (zitiert auf S. 27).

[MF10] F. Mattern, C. Flörkemeier. „Vom Internet der Computer zum Internet der Dinge“. In: *Informatik-Spektrum* 33.2 (2010), S. 107–121. ISSN: 0170-6012. DOI: 10.1007/s00287-010-0417-7 (zitiert auf S. 15).

[MSH16] C. Manzei, L. Schleupner, R. Heinze, Hrsg. *Industrie 4.0 im internationalen Kontext: Kernkonzepte, Ergebnisse, Trends*. Beuth Innovation. Berlin u. a.: VDE Verlag GmbH und Beuth Verlag GmbH, 2016. ISBN: 978-3-410-26049-3, 978-3-8007-3671-3 (zitiert auf S. 21, 22).

[RP06] P. Rechenberg, G. Pomberger. *Informatik-Handbuch*. 4., aktualisierte und erw. Aufl. München u.a.: Hanser, 2006. ISBN: 978-3-446-40185-3 (zitiert auf S. 55).

[Sie11] Siemens. *Der digitale Zwilling*. 17.11.2015. URL: https://www.siemens.com/customer-magazine/de/home/industrie/digitalisierung-im-maschinenbau/der-digitale-zwilling.html (zitiert auf S. 17).

[Spi07] S. Spinnarke. *OPC UA wird (neben anderen) Industrie 4.0 Standard*. 13.07.2016. URL: https://www.produktion.de/trends-innovationen/opc-ua-wird-neben-anderen-industrie-4-0-standard-334.html?page=1 (zitiert auf S. 17, 27).

[DIN16] DIN Deutsches Institut für Normung e. V. (SPEC 91345). *Referenzarchitekturmodell Industrie 4.0 (RAMI4.0)*. Berlin, Apr. 2016 (zitiert auf S. 16, 22–24, 30, 31).

[OPC14] OPC Foundation. *OPC Unified Architecture: Wegbereiter der 4. industriellen (R)Evolution*. März 2014. URL: https://opcfoundation.org/wp-content/uploads/2014/03/OPC_UA_I_4.0_Wegbereiter_DE_v2.pdf (zitiert auf S. 27).

[OPC15a] OPC Foundation. *Part 3 - Address Space Model (Version 1.03)*. Juli 2015. URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model (zitiert auf S. 27, 49, 50).

[OPC15b] OPC Foundation. *Part 5 - Information Model (Version 1.03)*. Juli 2015. URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model (zitiert auf S. 49, 50).

[OPC17] OPC Foundation. *Part 14: PubSub (Release Candidate 1.04)*. Feb. 2017. URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub/ (zitiert auf S. 27, 49).

[VDI13] VDI Verein Deutscher Ingenieure e.V. *Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. 2013. URL: https://www.vdi.de/uploads/media/Stellungnahme_Cyber-Physical_Systems.pdf (zitiert auf S. 15, 16).

Alle URLs in Fußnoten und Literaturverzeichnis wurden zuletzt am 13. 07. 2017 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift