Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung

Alexander M. Sowitzki

Eine Infrastruktur für Cyber Physical Systems

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander M. Sowitzki

Eine Infrastruktur für Cyber Physical Systems

Ausarbeitung eingereicht im Rahmen der Arbeit zum Grundprojekt

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Herr Professor. Dr. Kai von Luck

Eingereicht am: ???

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Zur Durchführung von Versuchen in einem Labor mit Internet of Things (IoT) ist es nötig, eine Infrastruktur aufzubauen, die einfaches Testen und Bereitstellen Softwareagenten ermöglicht. Sollen diesem Umfeld Multiagentensysteme mit Cyber Physical System (CPS) Komponenten ein gesetzt werden, ergibt sich daraus eine Vielzahl von Problemen, die die Umsetzung eines kurzen Entwicklungszyklusses erschweren. In diesem Dokument sollen sogenannte Showstopper identifiziert und Lösungsmöglichkeiten für den Versuchsaufbau angegeben werden.

1.2 Motivation

Während das IoT zunehmend an Verbreitung gewinnt, wird bei vielen Konzepten streng zwischen Anwendern und Entwicklern getrennt. Anwender verfügen hier über wenig bis kein Technikverständnis, weswegen für sie vereinfachte Mensch-Maschine-Schnittstellen entwickelt werden müssen. Über diese kann die Systemkonfiguration bis zu einem bestimmten Maße erfolgen. Entwickler hingegen sind für die Installation und Konfiguration des Systems zuständig und besitzen die dazu nötigen Fähigkeiten. [1, S. 112]

Vor allem im Endkonsumentenbereich wird davon ausgegangen, dass im Einsatzumfeld des Systems keine dedizierten Entwickler präsent sind sondern nur Anwender, weswegen Hersteller vom IoT und CPS Lösungen nur Anwenderschnittstellen bereitstellen. Sollte es unter den Endkonsumenten jedoch Entwickler geben, die komplexere Installationen umsetzen möchten, so müssen diese aus einer deutlich kleineren Auswahl von Lösungen wählen. Für die Zielgruppe von Personen mit Programmierkenntnissen existieren keine vollwertige und umfassende Lösung, die einen agentenorientierten Betrieb zulässt.

1.3 Abgrenzung

Die Entwicklung eines passenden Frameworks, der Systeminfrastruktur und der Erstellung eines Companion für diesen Versuchsaufbau soll sich in nachfolgenden Arbeiten beschäftigt werden. Dieses Projekt hat das Ziel, den DevOps Teil des Aufbaus darzustellen.

Es existieren verschiedene Projekte, unter anderem *Node-RED* und *OpenHAB*, die es Benutzern ohne Programmierkenntnisse ermöglichen sollen, Smart Home Module miteinander zu koppeln. Diese vereint jedoch, dass diese die Entwicklung auf verschiedenen Hosts nicht unterstützen und oft nur REST integriert haben, welche für die Einbindung von Embeddedsystemen unvorteilhaft ist. [2] [3]

2 Analyse

2.1 Grundbegriffe

Für die Aufstellung der Anforderungen ist es zunächst notwendig, nötige Grundbegriffe, vorrangig CPS, zu identifizieren und zu beschreiben.

2.1.1 Definition

CPS beschreiben eine Variante von Komponenten, die in einem IoT System eingesetzt werden können. Komponenten, die von sich aus nicht über eine Anbindung an ein Netzwerk verfügen und materiell (physikalisch) in der Umgebung vorhanden sind, erhalten eine IT-Komponente, die sie virtuell repräsentiert. In diese Kategorie fallen viele Systeme wie Drucker, Sensoren und Aktoren für Smart Environments und neuere Modelle von Automobilen. Das Konzept beschreibt, wie diese virtuelle Repräsentation umgesetzt und wie sie verwendet werden kann, um Mehrwert zu schaffen. CPS können eigenständig ohne Netzanbindung operieren, entfalten aber ihren eigentlichen Zweck erst im Zusammenspiel miteinander. [4] [5]

Grundsätzlich lassen sich CPS mit folgende Schwerpunkten beschreiben: [6, S. 22]

Sensorik & Aktorik

Die grundsätzliche Eigenschaft, die virtuelle Welt mit der realen zu koppeln, erfordert Einflussnahme auf die Umgebung. Dies geschieht durch den Einsatz von Sensoren und Aktoren. Sensoren dienen dazu, den Zustand der Welt zu messen, also z.B. die Temperatur des Raumes zu erfassen oder Sprachkommandos eines Benutzers zu erkennen. Aktoren verändern die Umgebung indem z.B. Fensterrollos oder Lampen gesteuert werden. [7]

Datenverarbeitung

Daten werden innerhalb des CPSs direkt verarbeitet und optional gespeichert. Auf der Grundlage der so gewonnenen Informationen kann dann mit der realen Umgebung interagiert werden.

Zu verarbeitende Daten sind heterogen: So können Bilddaten von Kameras auf Bewegung untersucht werden, Magnetkontakte mit ihrem Standort korreliert werden oder verschiedene Einzelinformationen mittels Complex Event Processing (CEP) zu einer neuen Informationen verknüpft werden.

Netzzusammenschluss

CPS bestehen aus mehreren Komponenten, die in einem Netzwerkverbund integriert sind. Kommunikation zwischen den Teilnehmern kann dabei über einen Dienst wie einen Messagebroker erfolgen, oder auf rudimentärere Art über Uni-, Multi- oder Broadcasts im Netzwerk erfolgen. Da unterschiedliche Komponenten im CPS wie Agenten in einem Multiagentensystem agieren, dient das Austauschen der Informationen der Erfüllung eines Zwecks, den die einzelnen Agenten alleine nicht erreichen könnten. Damit dies möglich ist, müssen mehrere Voraussetzungen erfüllt werden. [8]

So müssen alle CPS Komponenten auf die verwendendete Kommunikationsweise zugreifen können. Sollte das Verfahren zu aufwändig sein, könnte es Embeddedsysteme ausschließen. Zudem ist es für den Betrieb von Diensten teilweise zwingend, ein Mindestmaß an QoS bereitzustellen, also bestimmten Garantien für die Übertragung von Daten im Netzwerk, auch Dienstgüte genannt. [6, S. 162, 232]

Während im einfachen Falle das komplette CPS in eine Trust Boundary fällt, also von einem Anwender oder einer Anwendergruppe mit den selben Zugehörigkeiten betrieben wird, können für komplizierte Fälle ansonsten unabhängige CPS in einer Föderation zusammen arbeiten, um bestimmte Aufgaben zu erfüllen. Dies erfordert besondere Beachtung der Systemgrenzen und Isolation.

Ein weiterer relevanter Aspekt ist die Selbstheilung des Systems. Falls Hardware- oder Softwarekomponenten ausfallen, sollte das System in der Lage sein, dies zu einem gewissen Maß zu überbrücken. So können standortunabhängige Programme auf andere Ausführungsumgebungen ausgelagert werden oder die Komponenten stellen bei Netzwerkausfall ihre Basisfunktionalität lokal bereit. [6, S. 233]

Mensch-Maschine-Interaktion

Ubiquitous Computing ist ein Modell welches beschreibt, dass Computer überall und unsichtbar in normalen Umgebungen present sind und mit dem Funktionalitätsbereich von nicht vernetzen Systemen verschmelzen. Während diese in normalen Arbeitsumgebungen eindeutig in Form

von Desktop-PCs, Laptops oder Smartphones erkennbar sind, werden sie hier versteckt und interagieren mit dem Benutzer mithilfe von Stellvertreterobjekten. Eine Form davon sind Companionsysteme, welche für eine Umgebung ein zentralisiertes Objekt mit Persönlichkeit bereitzustellen. Ubiquitous Computing ist u.a. in Smart Homes üblich. Dieser Trend ist einer der treibenden Kräfte für CPS und findet sich in Fahrzeugen und Sensorumgebungen wieder. [6, S. 19 f.]

Teilautonomie und -automation

Die Aufgabenerfüllung in einem CPS soll automatisiert erfolgen, d.h. nach der Konfiguration durch den Benutzer soll das System ohne Zutun des Nutzers vorgegebene Aktionen in Abhängigkeit von Sensordaten ausführen. Während dies nur auf Grundlage von einfachen Parametern erfolgen kann, ist in vielen Fällen die Bewertung der Systemsituation erforderlich. [9, S. 7] [10]

Context Awareness

Für die Ermöglichung der Autonomie muss das System über einen Entscheidungsmechanismus verfügen, der die gegebene Systemsituation erkennen und einordnen kann. Dies kann in drei Abstufungen durchgeführt werden. Das System kann minimal über eine *Physical Awareness*, also Sensormesswerte und deren isolierte Bedeutung verfügen, um rudimentäre Entscheidungen zu treffen. Dies könnte ein frontaler Abstandssensor eines Roboters sein. Unterschreitet der gemessene Wert die eingestellte Schwelle, so bremst der Roboter.

Die Steigerung ist die *Situation Awareness*. Das System verwendet die Daten der *Physical Awareness* zu einer Situationsbewertung. Der Roboter kann aus einem abnehmenden Frontalabstand schließen, dass er sich auf ein Ziel zubewegt oder aber sich ein Objekt auf ihn zubewegt, sollte sich der Roboter nicht bewegen.

Zur vollständigen Situationsbewertung ist eine *Context Awareness* notwendig. Dies bedeutet, dass das System zur Entscheidung zusätzliche Informationen einbeziehen kann. So können Karteninformationen zur Umgebung von anderen Robotern angefordert werden um eine mögliche Ausweichroute um ein Hindernis zu finden oder es kann Kontakt mit anderem Robotern aufgenommen werden, um eventuelle Kollisionen zu vermeiden. [6, S. 83] [11]

Anpassungsfähigkeit

CPS sind eine Grundlage für anpassungsfähige Systeme und erfordern daher selbst eine hohe Flexibilität bei ihrem Einsatz. Auch wenn ein CPS für einen konkreten Anwendungsfall entwickelt

wird, so ist davon auszugehen, dass dieser sich während des Betriebs radikal verändern kann. Auf diese Änderung sollte keine Neuentwicklung des Systems erfolgen müssen. Indem einzelne Komponenten konfigurierbar gestaltet werden und deren Interaktion lose gekoppelt wird, ist eine adaptive Anpassung an neue, potentiell initial unbeachtete Einsatzbedingen möglich. [12, S. 2] [13]

Sicherheit

Die Sicherheit von CPS stellt eine besondere Herausforderung da. Die Trust Boundary eines Dienstes umfasst oft eine Vielzahl von Agenten, die gesichert werden müssen und führt zu einem verteilten System mit hoher Komplexität, was Sicherheitsoptimierungen erschwert. Dabei kann es sich um Embeddedkomponenten handeln, die nur über wenige Hardwareressourcen verfügen, um dies zu realisieren. So kann die Übertragung von Nachrichten mit einer starken Transportverschlüsselung kleinere Systeme signifikant auslasten. Zudem befinden sich diese Komponenten teilweise in ungesicherten Umgebungen, bsw. eine intelligente Türklingel. Auch können Systeme verschiedener Anwender kooperieren, um ein gemeinsames Ziel zu erreichen. Dies setzt eine gemeinsame Strategie voraus. [6, S. 79-83]

Eine zusätzliche Problematik ist der Datenschutz selbst. In einem Multiagentensysteme ist die Ausbreitung von Daten aufgrund seiner Verteiltheit nur schwer auf einzelne Komponenten beschränkbar, ohne Erweiterbarkeit und Funktion einzuschränken. [6, S. 120] [14]

2.1.2 Abgrenzung

IoT beschreibt die Integration von IT-Komponenten in Geräte und deren Vernetzung untereinander zur Erfüllung eines höheren Zieles. Ein IoT System ist also ein Multiagentensystem. Solche Systeme können bsw. in der Industrie in Produktionsmaschinen und Ähnliches. integriert werden, um eine bessere Übersicht über die Produktivität der Anlage zu erhalten oder Kapazitätsüberschüsse zu erkennen. Im häuslichen Umfeld können Lampen, Lichtschalter, Schaltsteckdosen etc. zusammengeschlossen werden, um den Bewohnern eine bessere Wohnerfahrung zu bieten. Diese Komponenten kommunizieren über verschiedene Transportprotokolle und -medien mit einem lokalen Hub, welchen anfallende Daten bündelt. In vielen kommerziellen Lösungen werden diese Daten dann an die Infrastruktur eines Dienstleisters weitergeleitet und dort verarbeitet. Steuerkommandos werden an die Komponenten auf dem selben Weg zurückgesendet. Bricht die Internetanbindung ab oder versagt der Anbieter, so verfügen die Komponenten einiger Hersteller über einen manuellen Schalter direkt an der Endkomponente

um die einzelne Funktionalität zu steuern. Eine Steuerung des Gesamtsystems ist in diesem Fall jedoch selten möglich.

Insofern beschreibt der Begriff IoT das gleiche Konzept wie CPS, nimmt jedoch stärkeren Bezug auf die Vernetzung der Geräte und wird häufig im Open Source Umfeld verwendet. Zudem kann ein CPS auch komplett ohne Netzanbindung arbeiten.

CPS beschreibt einen umfassenden Bereich, der sich mit der Integration einer realen Umgebung mit eine virtuelle Repräsentation beschäftigt. Die Verbindung solcher Komponenten in einem Netzwerk sowie die Beschreibung der Systemlogik überschneidet sich dabei mit dem vertikalen Rahmens dieses Projekts. [6, S. 17]

2.2 Herausforderungen

Das IoT ist ein Multiagentensystem, in welchem eine Sammlung von heterogenen Komponenten zusammenarbeitet. Diese Heterogenität bezieht sich dabei auf verschiedene Eigenschaften: [6, S. 17]

- *Funktion*: Agenten erfüllen stark voneinander abgegrenzte Funktionen.
- *Ressourcen*: Agenten verfügen über einen unterschiedlichen Umfang von Ressourcen wie Rechnerleistung, Speicher, Laufzeit etc.
- *Iteroperabilität*: Während Agenten, die auf einem vollwertigen Betriebssystem laufen über IP-Netzwerke kommunizieren zu können, haben Kleinstgeräte teilweise nur bedarfsgetriebene Protokolle wie Bluetooth Low Energy zur Verfügung, was potentiell Protokollbrücken erfordert.
- *Komplexität*: Je nach Funktionsumfang ist die Software, wie auch die Hardware, von unterschiedlicher Komplexität. Bsw. ist das Erkennen einer geschlossenen Tür über einen Kontaktschalter potenziell einfacher als die Gesichtserkennung in einer laufenden Videoaufnahme.
- *Kapselung*: Während einige Agenten über eine sehr lokalisierte Funktionalität wie das Schalten einer einzelnen Lampe verfügen, ist die Funktionalität von bsw. der Positionserkennung für Personen potenziell über mehrere Maschinen aufgeteilt. Zweiteres ergibt zusätzliche Anforderungen wie bsw. ein eigenes Medium für die interne Kommunikation oder eine Integrationsfähigkeit in bestehende Infrastrukturen.

- *Physikalische Zugänglichkeit*: Komponenten, die direkt mit dem Benutzer interagieren, sind für Installation und Wartung zugänglicher als bsw. in Decken verbaute Ventilatonsanlagen.
- *Sicherheitsanforderungen*: Während die bisher genannten Beispiele zwar sicher zu Betreiben sein müssen, ein Funktionsaufall jedoch potentiell mehr eine Umständlichkeit darstellt als eine Gefahr, so müssen Türschlösser, Alarmmelder etc. über eine hohe Verlässlichkeit verfügen. Dies schließt die verwendete Infrastruktur ein.

2.3 Verteilte Funktionalitäten

Im Gegensatz zu vielen kommerziellen Produkten, die ihre Funktionalität abgeschlossen bereitstellen, soll neben einer losen Kopplung in dieser Infrastruktur auch eine starke Kopplung möglich sein.

Die Anforderung, stark gekoppelte Agenten umsetzen zu können, ergibt sich aus der Heterogenität der einzelnen Komponenten. Unterschiedliche Agenten verfügen über unterschiedliche Fähigkeiten und haben dadurch eine unterschiedliche Effizienz bei der Ausführung von Teilaufgaben. Zudem gibt es einzigartige Agenten, die für die Erfüllung einer bestimmten Aufgabe kritisch sind, wie z.B. die Lampe beim Einschalten der Beleuchtung. Es ist daher nötig, dieses Modell zu spezifizieren. In Abbildung 2.1 ist ein Beispielsystem gezeigt. Es ist also zwischen standortagnostischen Agenten zu unterscheiden, welche nur auf generische Ressourcen wie Rechenzeit und Speicher angewiesen sind, und standortspezifischen Agenten, welche spezielle lokale Resourcen benötigen.

2.4 Datenverarbeitung

Agenten kommunizieren über den Austausch von Nachrichten. Für diese Kommunikationart wurden folgende Anforderungen ermittelt:

Beschränkung Während die Verbreitung der Nachrichten im gesamten Netzwerk als komfortabel und einfach erscheint, bringt dieses Vorgehen mehrere Nachteile mit sich. Da sich das Multiagentensystem über mehrere Standorte erstrecken kann, müssten größere Datenmengen über die Internetanbindung übertragen werden, was bei geringen Bandbreiten Kosten und

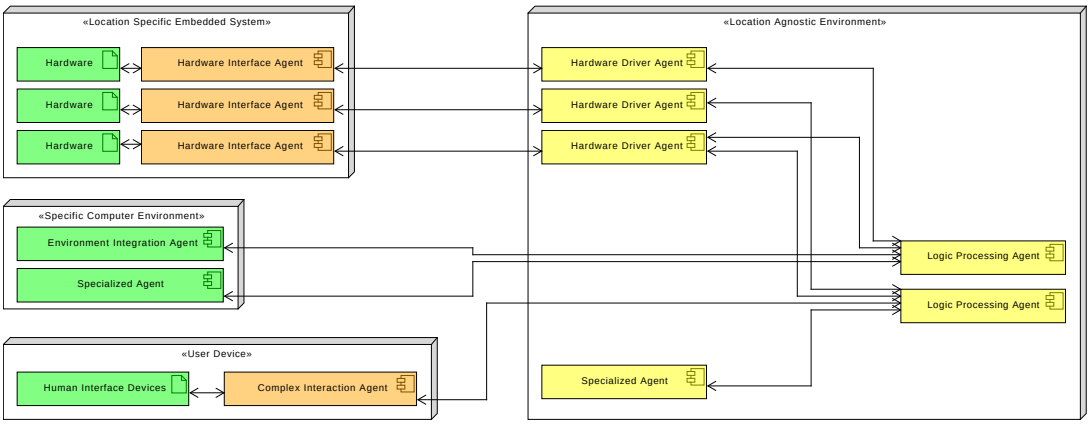

Abbildung 2.1: Architektur für verteilte Funktionalitäten

Störungen bei anderen Diensten verursachen kann. Aus diesem Grund soll zwischen standortspezifischen und globalen Daten unterschieden werden, welche im Netzwerk verteilt werden.

Beschreibung Der Formatierung der Nachrichten sollte in einem Format geschehen, das auch simple Agenten nicht mit unnötiger Serialisierungsarbeit belastet. Deshalb wird von platzineffizienten Codierungen wie JSON und XML abgesehen. Vielmehr werden Nachrichten soweit wie möglich zerlegt. Verbleibende Informationen werden im Big Endian Format direkt binär übertragen. In vielen Programmiersprachen erfordert dies nur geringen Aufwand.

Speicherung Grundsätzlich soll im Versuchsaufbau Event Sourcing betrieben werden. Der Messagebroker sollte sich dabei um die Persistenz kümmern. Dies bedeutet, dass nur für spezielle Lösung eine Datenpersistenz notwendig ist.

2.4.1 Hardwaresupport

Es ist mit den begrenzten Ressourcen dieses Projektes nicht möglich, alle verfügbaren Hardwareplattformen zu unterstützen. Aus diesem Grund sollte ein Weg gewählt werden, der eine möglichst hohe Abdeckung an Anwendungsfällen ermöglicht. Um dies zu erreichen wurden drei Plattformgattungen identifiziert:

Workstation & Server Vermutlich der Großteil der Agenten wird auf normalen Arbeitsstationen und Serversystemen laufen. Während vor einigen Jahren noch 32 Bit Systeme noch

dominant vertreten waren, ist heute die Migration auf die x86_64/amd64 Architektur fast vollständig umgesetzt. Neben der x86 Architektur sind andere Prozessorarchitekturen in diesem Bereich rar. Aus diesem Grund soll für Workstations und Server nur die 64 Bit x86 Variante unterstützt werden. Diese Systeme verfügen über keine spezielle Peripherie die generell gesondert beachtet werden muss.

Systems on a Chip Zu dieser Gattung zählen Embeddedplattformen wie der *Raspberry PI* oder das *Beaglebone*. Diese Hardwareplattformen werden über ein Betriebssystem angesprochen und verfügen über eine schwächere Leistung als Systeme der vorherigen Kategorie, sind aber deutlich stromsparender und verfügen meistens über speziellere Peripherie zum Ansteuern von elektrotechnischen Komponenten. Hier ist vorwiegend dir Prozessorarchitektur ARM zu finden. Während neuere ARM Prozessoren 64 Bit unterstützen ist die Softwareunterstützung noch nicht weitläufig vorhanden. [15] [16]

Microcontroller Für besondere Stromsparanforderungen, verbesserte Echtzeitfähigkeit oder Unterstützung komplexerer Elektronik kann es nötig sein, Microchips zu verwenden. Diese verfügen über kein Betriebssystem, betreiben eine einzelne Programmeinheit und ermöglichen weitreichende Kontrolle über sich selbst. Die Architekturen sind hier stark heterogen. Viele Hersteller liefern für ihre Produkte jeweils einen eigenen C-Compiler mit. Während die Unterstützung dieser Hardwaregattung wichtig ist, kann sie nur eingeschränkt in eine Orchestrierungslösung eingebunden werden. Gründe dafür sind u.a., dass das Programmieren und Updaten dieser Systeme über Hilfssysteme erfolgen muss und Monitoringfähigkeiten anwendungsspezifisch umgesetzt werden müssten.

2.4.2 Programmiersprachenfreiheit

Die von Programmierern bevorzugten Sprachen hängen u.a. von persönlichen Präferenzen ab und sind deswegen stark gestreut. Programmiersprachen auszuschließen ist deswegen für ein Plattformprojekt nicht sinnvoll. Die zeitliche Beschränkung des Projekts macht es zudem nicht ratsam, die Plattformbibliothek in mehreren Sprachen zu entwickeln. Jedoch kann mit verschiedenen Designstrategien darauf hingearbeitet werden, dass die Integration von Programmen in anderen Sprachen möglichst einfach ist.

Bei der Begrenzung der Programmiersprachen sollen keine Sprachen ausgeschlossenen werden, die nur auf einer Teilmenge der Plattformgruppen laufen. Laut aktuellen Analysen sind

folgende Sprachen führend (Alphabetisch sortiert, vorkommend in den Top 20): Java, C, C#, Python, JavaScript, PHP, .NET, Assembly, Ruby, R, Go, Swift. Der Tiobe Index führt Ässemblyäls Programmiersprache auf, wobei nicht definiert wird, um welchem Dialekt es sich handelt. Da Assembly sich auf eine große Menge von Maschinensprachen bezieht, kann eine Unterstützung nicht gewährleistet werden. Abgesehen davon sollen alle übrigen Sprachen ohne Agenten, die als Gateways fungieren, angebunden werden können.

3 Design

3.1 Umgebung & Funktionalität

Als Testumgebung wird das Labor 10.81 am Berliner Tor 7 auf dem Gelände der HAW mit dem Kosenamen *catspace* verwendet. In diesem Raum werden mehrere Computersysteme installiert, um die Funktionalität eines Smart Homes bereitzustellen. Siehe dazu Abbildung 3.1.

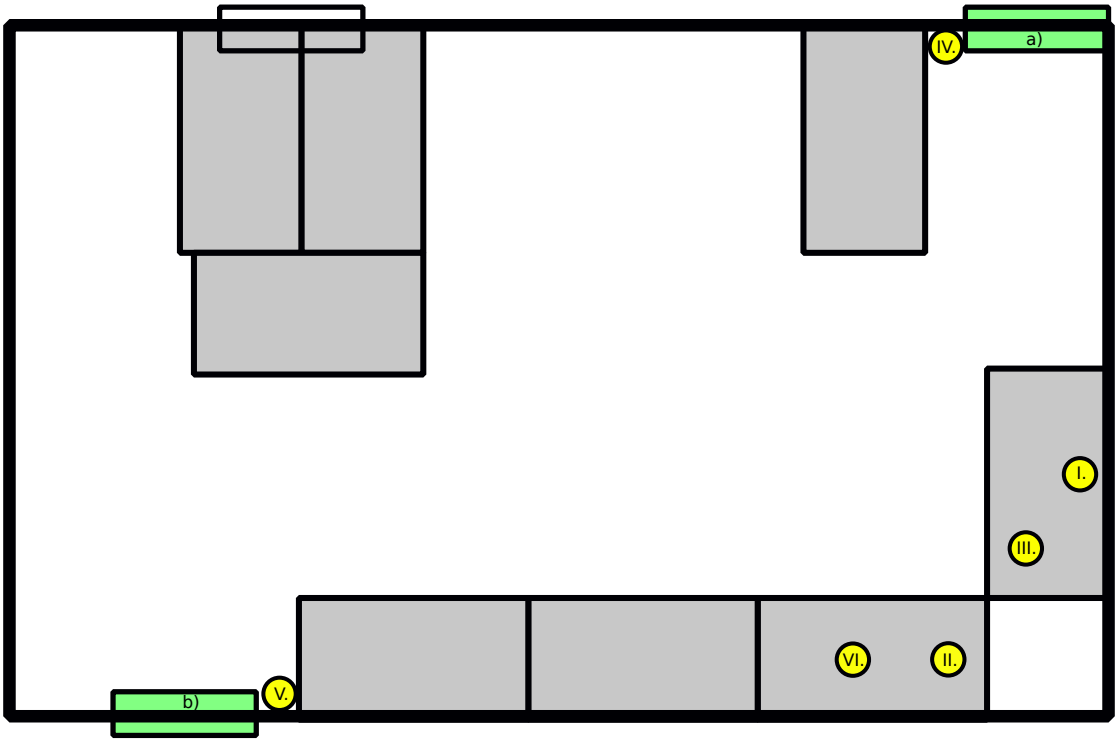

Abbildung 3.1: Raumschematik des Labors (Nicht maßstabsgetreu)

- Eine Workstation (I.) dient als Entwicklungsrechner. Die Desktopumgebung der Station wird mittels eines Agenten der Smart Home Umgebung verfügbar gemacht.

- Ein Server (II.), welcher standortungebundene Agenten ausführt und für rechen- und speicherintensive Aufgaben verwendet wird.
- Ein *Raspberry PI 2* (SOC) am Entwicklungsschreibtisch (III.). Hieran angebunden sind Umweltsensoren (BME280 sowie TSL2561), ein RGB Leuchtstreifen zur Erzeugung von Lichtstimmung, eine Warnlampe, ein Touchscreen für eine direkte Benutzerkontrolle der Umgebung, ein Schalter und ein Radio Frequency Identification (RFID) Leser zum Authentifizieren gegenüber der Arbeitsstation sowie eine Kamera zur Erfassung der Mimik des Entwicklers.
- Ein *Raspberry PI 2* (SOC) am Fenster (IV.). Dieser überprüft den Zustand des Fensters a) und verfügt über eine nach außen zeigende Kamera um Wetterverhältnisse zu beobachten.
- Ein *Raspberry PI 2* (SOC) an der Tür (V.). Hier ist ein Display (Mit SSD1306 Treiberchip) installiert, welcher Benutzer über den aktuellen Zustand des Smart Homes beim Betreten und Verlassen des Raums informiert. Zudem speist ein Türkontakt den Zustand der Eingangstür b) ins System ein. Des weiteren ermöglicht ein RFID Leser es, ein- und ausgehenden Benutzern sich an- und abzumelden.
- Ein *Raspberry PI 3* (SOC) im Smart Home Companion Minka (VI.). Dabei handelt es sich um einen Stellvertreter für die Installation in Form einer Katze mit motorisiert bewegbaren Ohren und Arm, RGB Augen sowie einem Ring bestehendend aus 60 RGB LEDS, der eine Plexiglasscheibe unterhalb der Katze beleuchtet. Minka verfügt zudem über eine Kamera, ein Mikrofon sowie einen Lautsprecher.

Zudem wurden zu entwickelnde Agenten ermittelt, die in Abbildung 3.2 dargestellt wurden und folgende Funktionalitäten erfüllen.

- *audio*: Gibt Audiodaten oder Text auf dem lokalen Audiogerät aus.
- *Broker*: Der Messagebroker, über welchen alle anderen Agenten miteinadner kommunizieren.
- *camera*: Liest Bilddaten einer lokalen Kamera aus und veröffentlicht sie im Netz.
- *cep*: Führt CEP für das restliche System nach vorgegebenen Regeln aus.
- *cic*: Bietet den Anwendern eine interaktive Möglichkeit zum Überwachen und Steuern des Systems.

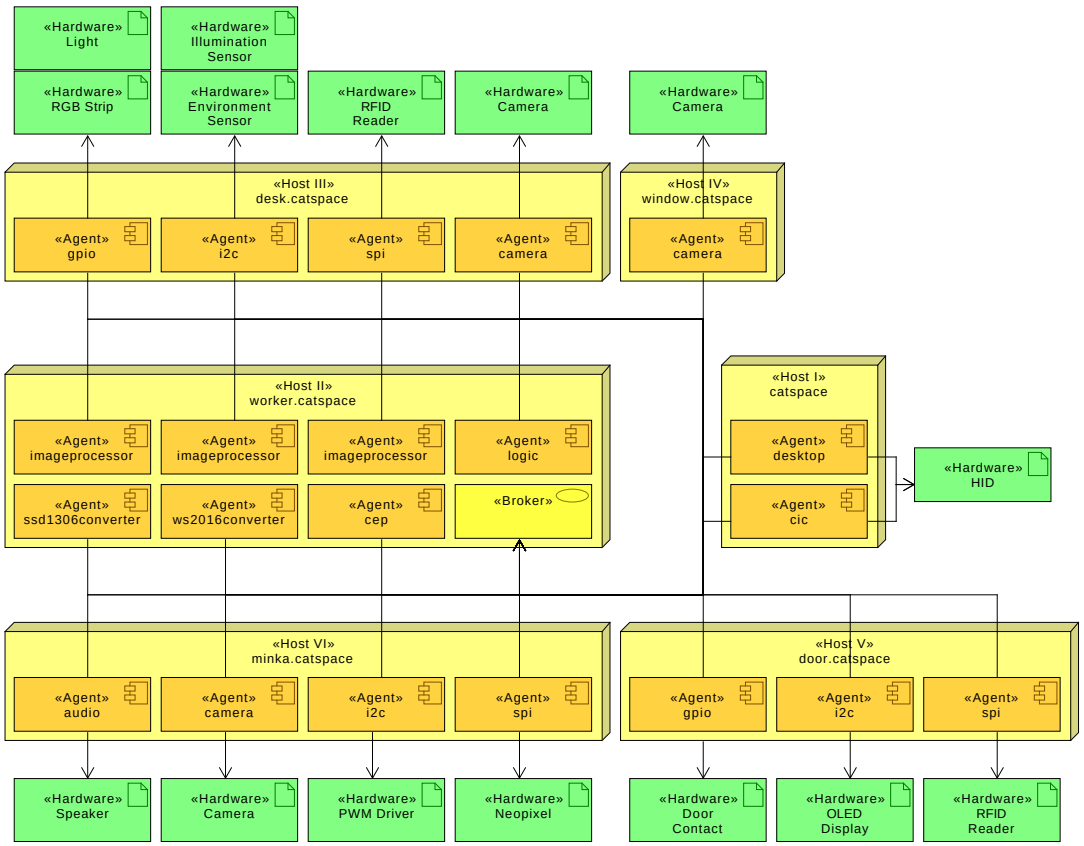

Abbildung 3.2: Agentübersicht

- *Desktop*: Integriert einen Computerdesktop ins System. So kann z.B. die lokale Audioausgabe stumm geschaltet oder der Bildschirmschoner aktiviert werden.
- *i2c*: Steuert Hardwarekomponenten an, die über I2C an das lokale System angebunden sind. Da der Zugriff auf verschiedene I2C Geräte auf dem selben Bus seriell erfolgen muss, ist die Bündelung von Zugriffsroutinen in einem Agenten ratsam, um nicht komplexere Synchronisationsmaßnahmen umsetzen zu müssen.
- *imageprocessor*: Verarbeitet die Bilddaten einer Quelle (z.B. einer Kamera) nach bestimmten Kriterien. Diese Arbeit kann u.a. aus Rotation, Skalierung, Reduzierung der Framerate oder dem Hinzufügen eines Zeitstempels bestehen. Ein Agent ist dabei für einen Bildstrom zuständig.

- *spi*: Steuert Hardwarekomponenten an, die über SPI an das lokale System angebunden sind. Der Zugriff auf mehrere Geräte am selben Bus erfolgt hier, wie bei I2C seriell, weswegen auch hier ein Agent pro Bus verwendet wird.
- *ssd1306converter*: Für die Darstellung eines Bildes auf einem Bildschirm mit SSD1306 Treiber müssen Bilder konvertiert werden. Um das Computersystem an der Hardware zu entlasten übernimmt dieser Agent die Aufgabe. Ein Agent ist dabei für einen Bildschirm zuständig.
- *ws2812converter*: WS2812 LEDs werden zwar über den SPI Bus angesteuert, benötigen jedoch eine spezielle Datenkodierung, da sie nur über einen Dateneingang ohne zusätzliche Taktleitung verfügen. Da der Konvertierungsprozess von Farbkanalinformationen zu den Daten, die letztlich an die LEDs gesandt werden, je nach Anzahl der LEDs aufwändig ist, wird auch dieser Vorgang ausgelagert. Ein Agent ist dabei für eine LED Anordnung zuständig.

3.2 Orchestration

Die eben genannte Aufstellung zeigt, dass eine Vielzahl von Agenten auf mehreren System verwaltet werden müssen. Da diese Agenten laufend entwickelt werden, ist der Verwaltungsaufwand für das Aktualisierungen, Testen und Überwachen hoch. Um dem entgegenzuwirken, werden Agenten als Microservices bereitgestellt und beteiligte Systeme in einem Cluster angebunden. Die entsprechenden Services werden dann mittels einer Deploymentkonfiguration korrekt verteilt.

3.2.1 Docker

Eine Möglichkeit zur Umsetzung von Microservicearchitekturen ist Docker. Hier wird das eigentliche Programm mit allen benötigten Abhängigkeiten in ein sogenanntes Image gepackt, wodurch sich die Portabilität deutlich erhöht. Mit neuen Dockerversionen ist es zudem mit sogenannten Manifesten möglich, Images für unterschiedliche Architekturen unter einer Referenz zu bündeln, wodurch Multiplatformfähigkeiten einfach realisiert werden können. [17]

Docker Images sollten im Gegensatz zu normalen Betriebssysteminstallationen auf den exakten Anwendungsfall zugeschnitten sein. Während Betriebssysteme eine Vielzahl von Anwendungen unterstützen müssen, iterativ und partiell aktualisierbar sind und selten das physikalische

Datenträgermedium wechseln, weswegen Aktualisierungen so häufig und schnell erfolgen können. Für eine Aktualisierung werden Docker Images neu erstellt, was bedeutet, dass Buildprozesse im Normalfall von vorne durchgeführt werden und nach Fertigstellung vollständig zum Zielsystem übertragen werden müssen. Deshalb ist es von großer Bedeutung, den Buildprozess kurz und unterteilbar zu halten und die Imagegröße klein zu halten, um die Entwicklungszeit nicht unnötig zu verlängern. [18]

Für diesen Anwendungfall wurde deshalb folgende Imagestruktur erstellt. Siehe zudem Abbildung 3.3.

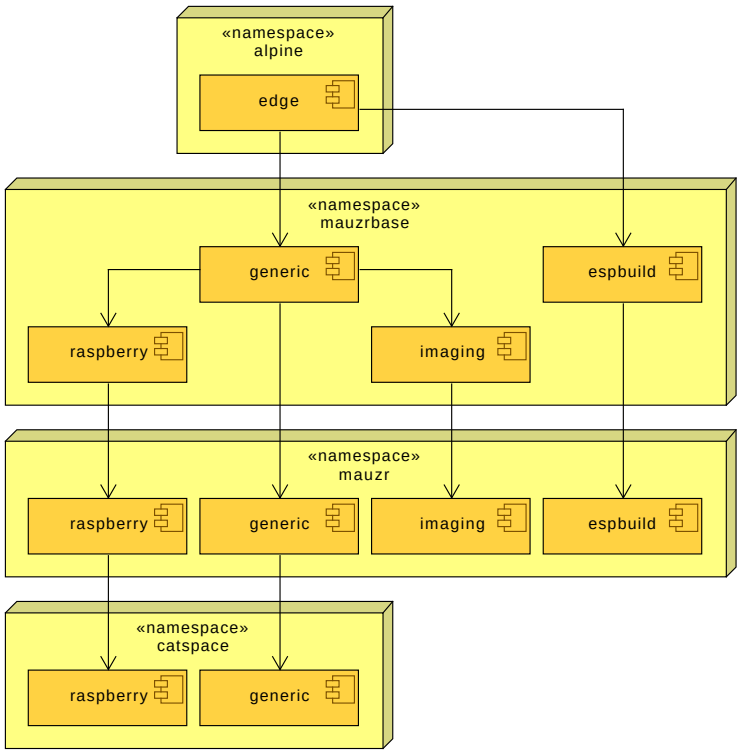

Abbildung 3.3: Struktur der verwendeten Dockerimages

Als Grundlage für alle Images wird ein Image des Betriebssystems Alpine verwendet, welches sich durch eine umfangreiche Paketauswahl sowie sehr kleine Imagegrößen auszeichnet. Für die weitere Entwicklung sind mindestens zwei Namespaces nötig. Um die Arbeit an einer universellen IoT Infrastruktur von der konkreten Entwicklung im Labor zu trennen, werden

generisch einsetzbaren Komponenten in der Bibliothek *mauzr* zusammengefasst. Konkrete Agenten für das Labor werden im Namespace *catspace* untergebracht.

Da für einige Anwendungsfälle Abhängigkeiten gebaut werden müssen, wurde zusätzlich der Namespace *mauzrbase* erstellt. Hier werden Images mit von mauzr und catspace benötigten Abhängigkeiten bereitgestellt und deren Buildprozess somit ausgelagert. Dadurch muss bei einer Veränderung von mauzr die Abhängigkeit selbst auf einem anderem Rechner nicht erneut gebaut werden.

Grundsätzlich gibt es vier Zweige für Images:

- *espbuild*: Dient zum Bauen und Bereitstellen von *mauzr* auf ESP Systemen, einer Mikrochipgattung der Firma *espressif*.
- *generic*: Ist die Grundlage für alle folgenden Varianten und enthält *mauzr* inklusive Abhängikeiten, um generische Funktionalitäten auszuführen. Dies bedeutet, dass größere Abhängigkeiten wie GUI Funktionalität nicht inbegriffen sind.
- *imaging*: Baut auf generic auf und erweitert es um benötigte Abhängigkeiten zur Bildverwertung. Dies schließt Bibliotheken wie openCV ein.
- *raspberry*: Zur vollständigen Nutzung (Insbesondere Pullup Widerständen für GPOs und der Ansteuerung von WS2812 LEDs) auf *Raspbery PIs* werden plattformspezifische Abhängigkeiten benötigt. Diese werden in diesem Zweig bereitgestellt.

3.2.2 Kubernetes

Um die verwendeten Computersysteme zu einem Cluster zusammenzufügen, wird Kubernetes verwendet [19]. Kubernetes ermöglicht hier, Hosts mit einem laufenden Docker Dienst gebündelt zu verwalten und zu überwachen. Kubernetes unterstützt mit geringem Konfigurationsaufwand den Betrieb von Knoten mit ARM Architektur. [20]

Tainting Zur Zuordnung von Dockercontainern zu Knoten wird sogenanntes Tainting eingesetzt. Beim Deployment der Knoten wird angegeben, dass diese auf einem Knoten ausgeführt werden sollen, die für generische Agenten geeignet sind. Ansonsten wird der exakte Hostname spezifiziert, wodurch der Container immer auf dem entsprechenden System mit dessen Hardware ausgeführt wird.

Hardwarezugriff Indem beim Deployment eine privilegierte Ausführung gefordert wird, kann der Programm im Container auf Blockdevices und dahinter liegender Hardware zugreifen, als wenn er als natives Programm ausgeführt werden würde.

Dashboard Kubernetes wird über Kommandozeilenwerkzeuge konfiguriert. Während dies für komplexe Konfigurationen effizient ist, gestaltet sich die Überwachung und Steuerung der Agentenmenge als unübersichtlich. Aus diesem Grund wird das Webfrontend Kubernetes Dashboard eingesetzt, welches eine grafische Übersicht über alle Ressourcen von Kubernetes bietet. Es können Zustand und Logs der Agenten angezeigt werden, Kommandos in Agentencontainern ausgeführt werden und Agenten skaliert sowie an- und ausgeschaltet werden. Mit der Komponente *Heapster* ist es zudem möglich, den Verbrauch von Prozessor- und Speicherkapazität jedes einzelnen Agenten zu ermitteln. Denkbar ist auch eine automatische Alarmierung beim Ausfall von Agenten.

3.3 Kommunikation

Die Kommunikation zwischen den Agenten ist themen- und nachrichtenorientiert, weswegen sich ein Message Broker mit Publish Subscribe Architektur anbietet. Jeder Agent erhält eine eigene Kennung, weswegen Zugriff auf Daten beschränkt werden kann. Es soll Retainment, also das Vorhalten von Nachrichten und Transportverschlüsselung unterstützt werden. Zudem soll es möglich sein, mit anderen Brokerinstanzen ein Föderationsnetzwerk aufzubauen.

Für diesen Anwendungsfall bietet sich MQTT an, da mehrere Messagebrokerimplementierungen die Anforderungen vollständig unterstützen. Es wird ein Broker innerhalb des Clusters installiert, über welchen der komplette Nachrichtenaustausch statt findet.

3.4 Lizenz

Um das Projekt auch nach seinem Abschluss aktiv zu halten, wird angefertigter Quellcode unter AGPL 3 veröffentlicht. Dadurch kann der komplette Quellcode transparent bezogen und überprüft werden. Dies stellt zudem einen Vorteil während der Entwicklung dar, da viele Plattformen ihre Dienste für Open Source Projekte komplett kostenfrei anbieten. So ist es nicht nötig zusätzlichen Aufwand für Build- und Versionierungssysteme zu erfüllen. [21]

4 Schluss

4.1 Fazit

Während der initiale Aufwand zum Einrichten dieser Infrastruktur mit Schwerpunkt auf der Einarbeitungszeit zu Beginn ins Gewicht fällt, so erspart die Einrichtung langfristig sich wiederholende Aufgaben. Dadurch kann während der Entwicklungsphase mehr Zeit in die eigentliche Aufgabe investiert werden. Während der Funktionstests zeigte sich, dass vor der Dockerisierung von Agenten diese direkt auf dem entsprechenden System getestet werden sollten. Da ein Bug ([22]) in Docker es momentan unmöglich macht, Images nur nativ zu bauen, muss der Buildprozess auf mehrere Plattformen verteilt werden. Dies bringt mit sich, dass bis zum Abschluss des Buildprozesses mit dem Debugging gewartet werden muss, während ohne Dockerumgebung Programme unmittelbar getestet werden können.

4.2 Ausblick

Während der Umsetzung der Testumgebung musste keine Einschränkung hingenommen werden. Deshalb ist damit zu rechnen, dass bei der Umsetzung von Folgeprojekten eine weite Auswahl von Anwendungsfällen abgedeckt werden kann. Das Folgeprojekt besteht darin, auf Grundlage dieses Systems einen Smarthomecompanion zu entwickeln, der dieses System verwenden kann.

Glossar

Agent "Ein Agent ist ein Computersystem, das sich in einer bestimmten Umgebung befindet und welches fähig ist, eigenständige Aktionen in dieser Umgebung durchzuführen, um seine (vorgegebenen) Ziele zu erreichen."[23]. 4, 6, 8, 13–15, 17, 18, 20

CEP *Complex Event Processing*, Behandelt die Analyse, Erkennung, Gruppierung und Verarbeitung von zusammenhängenden Ereignissen. 4, 13

CPS *Cyber Physical System*, die Kombination von Geräten mit Elektronische Datenverarbeitung (EDV) Komponenten und der Zusammenführung dieser in ein Netzwerk. 1, 3–7

DevOps Ein Bereich der Arbeit eines Informatikers, der die Schnittmenge Zwischen den Bereichen Qualitätssicherung, Entwicklung und Wartung bildet. 2

EDV *Elektronische Datenverarbeitung*, das Bearbeiten und Erfassen von Daten durch computergestützte Systeme. 20

Framework Softwarebibliotheken, die die Abstraktion von Teilproblemen realisieren. 2

I2C Ein Busprotokoll für die Ansteuerung von Teilkomponenten innerhalb einer Hardwarekomponente. 14, 15

IoT *Internet of Things*, die Vernetzung von Geräten zum Datenaustausch. 1, 3, 6, 7

Multiagentensystem Eine Umgebung, welche aus mehreren Agenten besteht, die kooperativ ein gemeinsames Problem zu lösen versuchen. 1, 4, 6, 7

Open Source Softwareprojekte, deren Quellcode der Allgemeinheit frei zugänglich ist. 7

Publish Subscribe Ein Softwarepattern für den Nachrichtenaustausch zwischen mehreren Teilnehmern. Nachrichten werden von Quellen nicht direkt an Senken gesendet, sondern in bestimmten Kanälen eines Dienstes veröffentlicht. Teilnehmer, die entsprechende Nachrichten erhalten sollen, abonnieren entsprechende Kanäle. 18

QoS Das Angebot einer Funktion mit garantierten Parametern im Bereich der Dienstgüte. 4

RFID *Radio Frequency Identification*, Technologie zum Austausch von Daten über Funk. 13

Smart Environment Eine Umgebung mit vernetzten automatisierbaren Geräten und Computern. 21

Smart Home Ein Haushalt mit Smart Environment Fähigkeiten. 2, 5, 12, 13

SPI Ein Busprotokoll für die Ansteuerung von Teilkomponenten innerhalb einer Hardwarekomponente. 15

Literatur

[1] Edward R Griffor u. a. *Framework for Cyber-Physical Systems: Volume 2, Working Group Reports*. Techn. Ber. 2017.

[2] *Node-Red*. Englisch. URL: https://nodered.org (besucht am 09. 02. 2017).

[3] *openHAB*. Englisch. URL: https://www.openhab.org (besucht am 09. 02. 2017).

[4] *Cyber-Physical Systems ganz konkret*. URL: https://www.elektronikpraxis.vogel.de/cyber-physical-systems-ganz-konkret-a-440171 (besucht am 09. 02. 2017).

[5] *Cyber-physische Systeme*. URL: http://wirtschaftslexikon.gabler.de/Definition/cyber-physische-systeme.html (besucht am 09. 02. 2017).

[6] Eva Geisberger und Manfred Broy. *acatech STUDIE*. 2012.

[7] *Was sind Cyber-Physische Systeme?* URL: http://www.ideen2020.de/de/1906 (besucht am 09. 02. 2017).

[8] Teodora Sanislav u. a. "Multi-agent architecture for reliable Cyber-Physical Systems (CPS)". In: *Computers and Communications (ISCC), 2017 IEEE Symposium on*. IEEE. 2017, S. 170–175.

[9] Asok Ray. "Autonomous perception and decision-making in cyber-physical systems". In: *Computer Science & Education (ICCSE), 2013 8th International Conference on*. IEEE. 2013, S. 1–10.

[10] *Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. URL: https://www.vdi.de/uploads/media/Stellungnahme_Cyber-Physical_Systems.pdf (besucht am 09. 02. 2017).

[11] *Cyber Physical Systems - Zentrale Bausteine von Industrie 4.0*. URL: https://www.computerwoche.de/a/zentrale-bausteine-von-industrie-4-0,3090293 (besucht am 09. 02. 2017).

[12] Edward R Griffor u. a. *Framework for Cyber-Physical Systems: Volume 1, Overview*. Techn. Ber. 2017.

[13] *Cyber Physical Systems sinnvoll 'modellieren'*. URL: http://www.computer-automation.de/steuerungsebene/steuern-regeln/artikel/130407 (besucht am 09. 02. 2017).

[14] *Nationale Roadmap Embedded Systems*. URL: http://www.softwaresysteme.pt-dlr.de/media/content/Roadmap-Embedded-Systems.pdf (besucht am 09. 02. 2017).

[15] *Raspberry PI*. Englisch. URL: https://www.raspberrypi.org/ (besucht am 09. 02. 2017).

[16] *BeagleBone*. Englisch. URL: https://beagleboard.org/bone (besucht am 09. 02. 2017).

[17] *docker*. Englisch. URL: https://www.docker.com (besucht am 09. 02. 2017).

[18] *docker*. Englisch. URL: https://docs.docker.com/storage/storagedriver (besucht am 09. 02. 2017).

[19] *Production-Grade Container Orchestration*. Englisch. URL: https://kubernetes.io (besucht am 09. 02. 2017).

[20] *Building a multi-platform Kubernetes cluster on bare metal with kubeadm*. Englisch. URL: https://github.com/luxas/kubeadm-workshop (besucht am 09. 02. 2017).

[21] *GNU Affero General Public License*. Englisch. URL: https://www.gnu.org/licenses/agpl-3.0.de.html (besucht am 09. 02. 2017).

[22] *Docker CLI Issue Tracker: Image Manifest Override*. Englisch. URL: https://github.com/docker/cli/issues/327 (besucht am 08. 02. 2017).

[23] *VDE/VDI 2653. Agentensysteme in der Automatisierungstechnik*. Juni 2010.