Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung

Alexander M. Sowitzki

Anforderungen an eine Cyber Physical Systems Abstraktionsschicht am Beispiel von Smart Environments

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander M. Sowitzki

Anforderungen an eine Cyber Physical Systems Abstraktionsschicht am Beispiel von Smart Environments

Ausarbeitung eingereicht im Rahmen der Arbeit zum Hauptseminar

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

Betreuender Prüfer: Herr Prof. Dr. Kai von Luck

Eingereicht am: 31. August 2018

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

In Cyber Physical Systems (CPS) Netzwerken wie z.B. Smart Homes werden Daten verschiedenster Sensoren verwendet, um Aktorik zu steuern. Die korrekte Funktionsweise dieser Aktorik ist entscheidend für das Wohlbefinden und die Sicherheit der Anwender. Aus diesem Grund ist es wichtig, ein solches System robust zu gestalten. Die verteilte Natur von Smart Environments erfordert zudem, dass Agenten innerhalb dieser Umgebung jene Verteiltheit erfassen und nutzen können. Zudem soll berücksichtigt werden, dass ein solches System stetigen Anpassungen unterworfen sein kann und dies unterstützen muss.

1.2 Motivation

Diese Ausarbeitung befasst sich mit der Ermittlung der Anforderungen, die sich aus jener Problemstellung an die ausführende Plattform ergeben. Verschiedene Lösungen für CPS fokussieren sich auf Teilprobleme, integrieren jedoch nicht das Konzept eines Gesamtsystems oder der Entwicklung von eigenständigen Agenten in ihre Zielstellung. Es ist der hier aktuelle wissenschaftliche Stand zu überprüfen und eine Fragestellung für das Projekt der Masterarbeit zu definieren.

2 Analyse der Problemstellung

2.1 Ausfallsicherheit

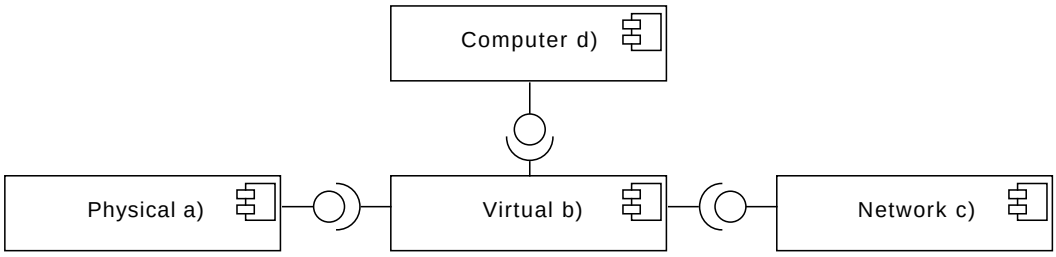

Abbildung 2.1: Bestandteile eines Cyber Physical Systems

Ein CPS besteht aus mehreren Komponenten, die unterschiedliche Herangehensweisen zur Erhöhung der Ausfallsicherheit erfordern. Im Groben setzt es sich aus drei Komponenten zusammen, welche in Abbildung 2.1 aufgezeigt sind: Der physikalische Teil (a) beinhaltet Elektrotechnik und Maschinerie, die einen direkten Einfluss auf die reale Wert ausüben. Der virtuelle Teil (b) ist ein Programm, welches mit dem physikalischem Teil interagiert, indem es Messwerte von jenem empfängt oder Kommandos versendet. Mehrere CPS können zusammen in einem Netzwerk (c) zusammengeschaltet werden, um kollaborativ komplexere Aufgaben zu erfüllen. Der virtuelle Teil ist zur Ausführung auf eine Computereinheit (d) angewiesen. [1]

Ein Ausfall der physikalischen Komponente (a) und die Anbindung an diese kann z.B. durch die redudante Auslegung von Hardwarekomponenten oder durch entsprechendes Design von elektronischen Komponenten erreicht werden. [2]

Um den Ausfall der virtuellen Komponente (b) zu verhindern, bietet es sich auch hier an, Redundanz einzuplanen. So können mehrere Computersysteme (d) installiert werden, auf denen je eine Kopie der Komponente läuft. Jede Kopie erhält gleichberechtigten Zugriff auf die physikalische Komponente und stimmt sich mit den anderen Kopien ab. Dieser Ansatz setzt eine

eng gekoppelte Netzwerkassoziation voraus. Eine andere Möglichkeit zur Verminderung der Auswirkungen eines Ausfalls ist, einen sogenannten Watchdog in die physikalische Komponente (a) einzubauen. Dies kann in Form eines Mikrochips oder Field-programmable Gate Array (FPGAs) umgesetzt werden. Dieses Subsystem überwacht die Kommunikation mit der virtuellen Komponente (b). Sollte diese illegale Steuersignale senden oder über einen längeren Zeitraum nicht erreichbar sein, so kann es die Komponente in einen Notfallmodus schalten und hier Basisfunktionalität garantieren. [3][4]

Der Zusammenbruch des Netzwerkes (c) kann in verschiedenen Schweregraden auftreten. Um dem totalen Ausfall der Netzwerkinfrastruktur vorzubeugen, wären mehrere Netzwerkanbindungen eine Möglichkeit, wobei Netzwerke im Allgemeinen bereits robust ausgelegt sein sollten und deren Ausfall für nicht umgehbare Probleme sprechen würde. Ein Ausfall von Netzwerkdiensten wie der Namensauflösung oder dem Message Broker kann durch Clusteringlösungen entgegengesteuert werden. [5]

2.1.1 Clustering

Eine Möglichkeit zur Erhöhung der Ausfallsicherheit ist der Einsatz eines Clusteringsystems oder Schwarms. Eine Lösung hierfür ist der Einsatz des Containersystems Docker unter Zuhilfenahme von Docker Swarm und Kubernetes. Programme werden hier mitsamt einer Laufzeitumgebung in sogenannte Container verpackt und auf Knoten des Clusters ausgeführt. Fällt ein Knoten aus, so wird das Programm auf anderen Knoten ad hoc neu gestartet. Durch die Verteilung des Schwarms auf mehrere unabhängige Computersysteme können hardwareferne Teile des Systems bis auf kurzfristige Ausfälle verlässlich betreiben werden. Innerhalb dieses Clusters können zudem Message Broker und Zusatzdienste wie Namensdienste mit Redundanzen ausgeführt werden. [6][7]

Die Computereinheit in CPS muss in den meisten Fällen einen Embeddedanteil haben, um mit der physikalischen Komponente kommunizieren zu können. Wenn auf dieser Computereinheit ein vollwertiges Betriebssystem wie Linux eingesetzt wird, kann die Einheit als Knoten im Cluster mitwirken. Dies entkompliziert die Infrastruktur, da keine zusätzlichen Rechensysteme eingerichtet werden müssen, sondern CPS Komponenten direkt für den Cluster mitverwendet werden können. [8]

Vorhergegangene Arbeiten haben gezeigt, dass Schwarmlösungen mit Docker auch auf Embeddedsystemen verwendet werden können. Die direkte Interaktion mit Hardware aus Containern heraus ist jedoch nur eingeschränkt möglich. Aus diesem Grund bietet es sich an,

die Softwarekomponente eines CPS in mehrere Agenten zu unterteilen (Vgl. Abb. 2.2). Ein so separierter hardwarenaher Agent interagiert direkt mit dem physikalischen Anteil und ist folglich an diesen gekoppelt. Daher bietet es sich an, ihn außerhalb des Clusters zu betreiben. Dieser Agent übernimmt keine eigenständige Kontrolle über die Hardware, sondern bindet diese in grundlegendster Weise in die Virtualität ein. Übergeordnete Agenten, die dank dieses Agenten indirekt Zugriff auf die Hardware haben, ohne an diese gekoppelt zu sein, führen die eigentliche Verarbeitung der Daten durch. Dies kann anwendungsfallgerechter durchgeführt werden, da diese Agenten innerhalb des Clusters z.B. bedarfsgerecht Rechenkapazität erhalten können. Weitere Agenten übernehmen die Aggregation und Entscheidungsfindung aufgrund dieser Daten. Da diese Prozesse auf den konkreten Anwendungsfall zugeschnitten werden müssen, und deren Komplexität mutmaßlich höher ist, kann das Clustering dabei helfen, fehlerhafte Agenten zurückzusetzen oder Agenten mit ressourcenaufwändiger Datenverarbeitung auf andere Knoten zu verlagern. [8][9]

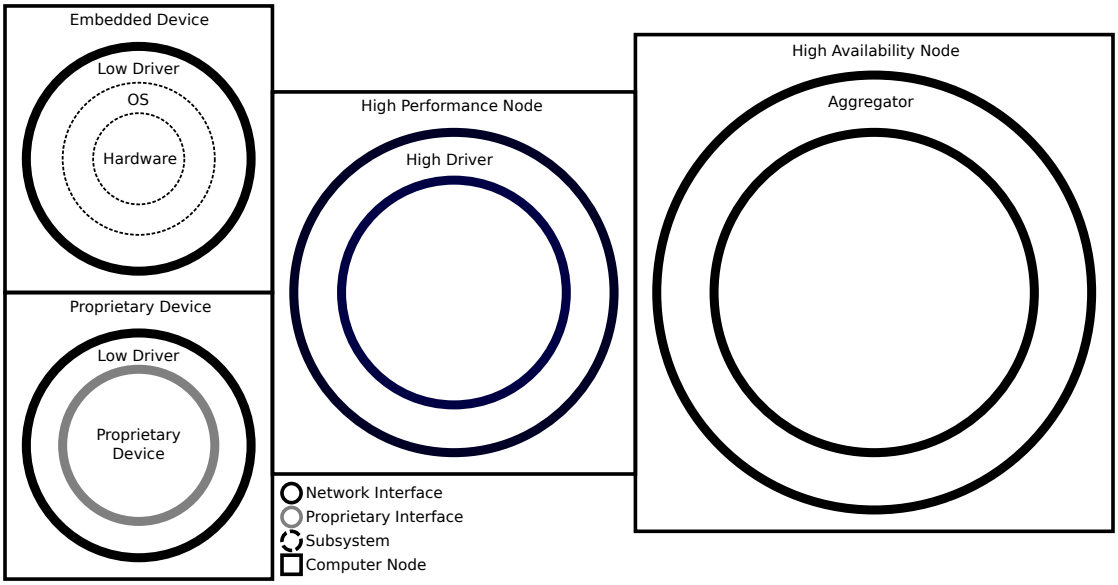

Abbildung 2.2: Verteiltes Zwiebelmodell

2.1.2 Kommunikationsform

Es bestehen verschiedene Möglichkeiten, wie CPS Agenten miteinander kommunizieren können. Da davon ausgegangen werden muss, dass Agenten ausfallen oder neustarten müssen, ist eine Architektur nötig, die es Agenten ermöglicht, unmittelbar den Zugriff auf alle relevanten Informationen zu erhalten. Dies lässt sich gut mit einer Publish Subscribe Architektur realisieren, welche oft einen Message Broker voraussetzt. Diese Realisierung führt zu einer Abhängigkeit zu einen zusätzlichen Netzwerkdienst, wodurch jedoch das Kommunikationsprotokoll zwischen den Agenten deutlich vereinfacht wird, da sogenanntes Retainment und Quality of Service (QoS) durch den Dienst realisiert wird. Zur Umsetzung von diesem kann auf eine Vielzahl von Projekten zugegriffen werden, von denen einige unter Open Source Lizenz veröffentlicht werden, weswegen hier keine Entwicklungszeit investiert werden muss. [10][11][12]

2.1.3 Selbstheilung

Der klassische Ansatz von Clustering beinhaltet, dass der Ausfall einer Programminstanz, wie z.B. eines Message Brokers, für Clientprogramme nicht bemerkbar ist, da der Zustand des Programms über alle Instanzen verteilt ist. Zwecks Komplexitätssenkung oder aufgrund von beschränkten Hardwareressourcen kann es jedoch nötig sein, nur eine Instanz eines Dienstes zu betreiben. Sollte dieser ausfallen, so betrifft dies sämtliche Clienten, da der Dienstzustand zumindest teilweise verloren geht. [13]

Da nicht davon ausgegangen werden kann, dass für alle Szenarien ein Cluster an Message Brokern existiert, sollten Agenten in der Lage sein, den Zustand des Systems nach einem Ausfall wiederherzustellen. Dazu kann ein Agent anhand der für Informationen konfigurierten Dienstgüte entscheiden, ob der Wert vorgehalten werden sollte. Bricht die Verbindung zum Agenten ab, so wird der Wert erneut veröffentlicht. Dies hat den Vorteil, dass dadurch sogar ein Wechsel zu einem Message Broker einer anderen Domäne erfolgen kann (Vgl. Abbildung 2.3). [14]

Wenn sichergestellt wird, dass sich alle Agenten zum neuen Message Broker verbinden, werden dadurch ohne zusätzlichen Aufwand alle Zustände automatisch wieder in das System eingepflegt. Durch geschicktes Vermeiden von Raceconditions kann dadurch der Gesamtzustand wiederhergestellt werden. Durch Synchronisierungsalgorithmen für verteilte Systeme kann zudem, wie oben angemerkt, über das Publish Subscribe System der gemeinsame Zugriff auf eine exklusive Hardwareverbindung geregelt werden.

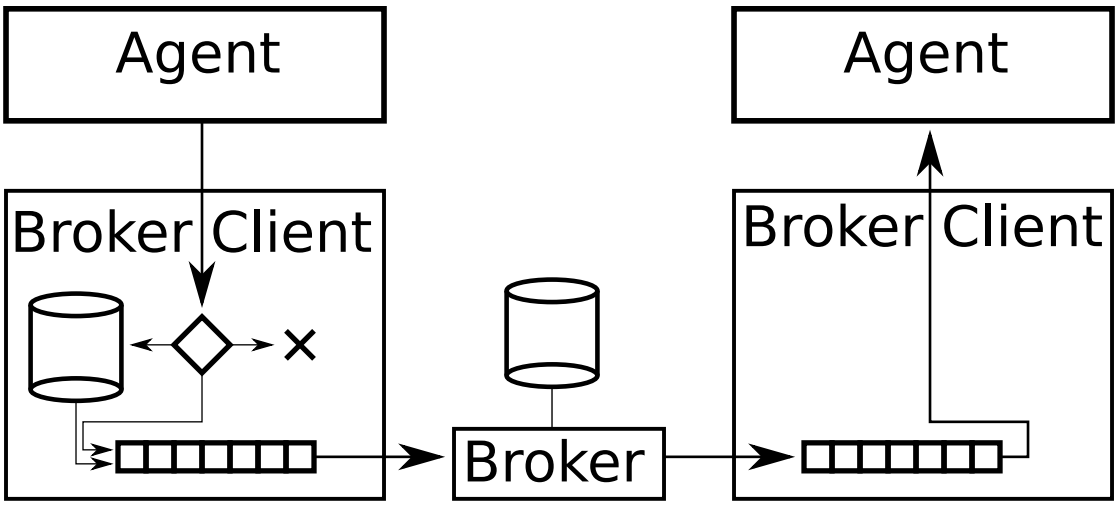

Abbildung 2.3: Kommunikationsstack zwischen Agenten

2.2 Erweiterbarkeit

2.2.1 Nachrichtenformat

In vielen Systemen wird die Serialisierungssprache JavaScript Object Notation (JSON) verwendet. JSON bietet sich für kleinere, niedrigfrequente Datenmengen an, da Daten einfach für Menschen lesbar sind, was Wartung und Dokumentation vereinfacht. Da in JSON jedoch als Klartextstring kodiert wird, sind Speicherineffizienz und Kodierungsaufwand hoch. Dadurch können Aufgaben wie die Übermittlung von hochfrequenten Bilddaten wie die einer Überwachungskamera nur mit überproportionierter Hardware realisiert werden. [15][16]

Eine andere Möglichkeit ist die Verwendung von Google Protocol Buffer. Google Protocol Buffer kodiert Daten binär und speichereffizient. Ein Nachteil ist jedoch, dass ein Codegenerator für jede zu verwendende Sprache vorhanden sein muss, der aus einer Formatbeschreibung statischen Code generiert. Dies belastet die Entwicklungsdauer und schränkt die Auswahl an verwendbaren Sprachen stark ein. [17]

Für das Projekt ist die Mischung beider Szenarien wünschenswert. Innerhalb des Publish Subscribe Verzeichnisses soll für jedes Topic festgehalten werden (z.B. wie bei Multipurpose Internet Mail Extensions (MIME) Type bei E-Mails), welcher Datentyp in jenem veröffentlicht wird. Dabei kann es sich um ein einzelnes Datum handeln, aber auch um eine Liste zusammengehörender Werte. Um keine unnötige Funktionseinschränkung vorzunehmen, sollte es möglich

sein, neben komplexeren Daten, wie z.B. Bildern, Datentypen in einer Liste zusammenzufassen, die allgemein in Programmiersprachen verfügbar sind, also Integer, Floats und deren Derivate. Beispiele für komplexe Datentypen wären ein JPEG Bild, eine Temperaturmessung mit Wert und Messzeitpunkt sowie eine Liste vordefinierter Länge mit Farbkanälen zur Steuerung eines Beleuchtungsaufbaus. [18]

2.2.2 Benötigte Funktionen

Es ist davon auszugehen, dass sich Anforderungen an ein Smart Environment innerhalb einer längeren Lebensdauer kontinuierlich ändern: Neue Gerätschaften werden hinzugefügt, Räume ummöbliert, neue Bewohner bekommen hinzu. Aus diesem Grund ist es kritisch, eine unkomplizierte Erweiterung zu ermöglichen.

Ein Sonderszenario ist die Einbindung einer Komponente, die nicht direkt für das bestehende Smart Environments entwickelt wurde. Für den Fall, dass eine Komponente mit dem Message Broker des Systems kompatibel ist, sollte das Nachrichtenformat des Systems einfach zugänglich gestaltet werden. Ansonsten ist es erforderlich einen Agenten zu implementieren, der Kommunikation mit jener Komponente übernimmt (Siehe Abbildung 2.2). [19]

Diese Umsetzung sollte zielgerichtet und schnell umsetzbar sein. Dazu ist eine Abstraktion und Bündelung von allgemein benötigten Funktionen nötig. Diese lauten:

- **Identifizieren und Verbinden mit Message Brokern**: Alle Agenten benötigen Zugriff auf die Publish Subscribe Architektur. Die Abstraktionsschicht soll für einen oder mehrere Agenten den zuständigen Message Broker identifizieren, sich mit ihm authentifizieren und die Verbindung bereitstellen. [20]
- **Bereitstellen von clientseitigen QoS**: Der Message Broker kümmert sich nach Erhalt einer Nachricht darum, deren QoS Dienstgrad umzusetzen. Bis die Nachricht dort ankommt ist der Client dafür zuständig, diese Nachricht ggf. zu speichern und erneut zu versenden - u.U. sogar nach einem Neustart. [14]
- **Auswahl an oft genutzen Agenten**: Agenten, die bspw. Zugriff auf serielle Schnittstellen oder Inter-Integrated Circuit (I2C) bereitstellen, werden von Agenten benötigt, die über diese Kommunikationswege mit anderen Hardwarebausteinen Daten austauschen müssen.

- **Rein Publish Subscribe basierte Konfiguration**: Bis auf die Zugangsinformationen zum Publish Subscribe System soll die komplette Agentenkonfiguration über das Verzeichnis laufen. Dies ermöglicht die bedarfsgerechte Erstellung, Einstellung und Verlagerung von Agenten. Ein weiterer Vorteil durch die Zentralisierung ist eine gesteigerte Übersicht und geringerer Wartungsaufwand.

- **Zentrales Logging**: Um einen Überblick in diesem verteilten System zu behalten, ist eine zentrale Übersicht über den Status eines Agenten und eventuelle Fehlermeldungen essentiell. Dies kann über verschiedene Arten wie z.B. einem zentralem Loggingserver durch systemd übernommen werden. Eine weitere Möglichkeit ist das Versenden entsprechender Logausgaben über das Publish Subscribe System, wodurch sie durch andere Agenten wie z.B. Watchdogs verarbeitet werden und zu einem selbstregulierendem System beitragen können. [3]

2.3 Konferenzen und Literatur

Die “Internet of Things Conference” ist eine Konferenz für aktuelle Themen aus dem Bereich Internet of Things (IoT), welcher auch für diese Ausarbeitung relevant ist. Wie in vorherigen Arbeiten aufgezeigt, überschneiden sich CPS und IoT signifikant, weswegen Erkenntnisse auf diesem Bereich verwendet werden können. Für CPS existiert seit diesem Jahr die “IEEE International Conference on Industrial Cyber-Physical Systems”, welche in die selbe Kerbe schlägt. Explizit für Smart Environments findet die “International Conference on Intelligent Environments” in Rom statt. [21][22][23][24]

Für einen Überblick über CPS eignet sich die Acatech Studie “Integrierte Forschungsagenda Cyber-Physical Systems”, welche sich mit grundlegenden Fragestellungen und Szenarien beschäftigt. Bei der Sichtung der Literatur zeigte sich, dass Teilgebiete der Problemstellung teilweise im Detail, teilweise grob, behandelt wurden. Auf Grundlage dieser Quellen konnte auf neue Erkenntnisse geschlossen werden.

3 Abschluss

3.1 Offene Punkte

Die Erstellung von ausfallsicheren Hardwarebausteinen wird bewusst ausgelassen, da dies eine starke Fokussierung auf den physikalischen Teil des CPS erfordern würde. Verschiedenste Fehlerszenarien wie Kurzschlüsse oder Geräte, die durch einen Fehler komplette Bussysteme blockieren können, bedürfen elektronischer Bauelemente wie Optokopplern oder zusätzlichen Mikrochips und führen zu einer komplexen Verdrahtung, die zu einem hohen Zeit- und Kostenrahmen führt. Dies sprengt den Rahmen der Arbeit, aus welchem Grund Schwachpunkte, die sich präferiert über zusätzliche oder geänderte Hardware am Besten lösen lassen würden, zurückgestellt werden und sich auf die Ausfallsicherheit der virtuellen Seite und deren Verbindung zum physikalischem Teil gesondert konzentriert wird.

Ein entscheidender Faktor für die Qualität von Software ist das Durchführen von Tests und Audits. Besonders für ein verteiltes System sind Ansätze für eine vollständige und umfassende Testlösung herausfordernd. Mit diesen Themen wird sich innerhalb des Laborumfeldes bereits befasst, weswegen dieses Themengebiet offen gehalten wird. [25]

3.2 Fragestellung für weitere Arbeiten

Aus der Analyse dieser Arbeit ergibt sich, dass es für die Umsetzung einer robusten und erweiterbaren CPS Infrastruktur bereits Bausteine zur Umsetzung gibt, jedoch keine Architektur an sich. So existieren z.B. Lösungen für das Aggregieren von Daten mit Complex Event Processing. Keine der Lösungen bietet jedoch für sich alleine Redundanz, Anbindung zu Publish Subscribe Systemen, effiziente Serialisierung oder Richtwerte, wie Daten im Publish Subscribe Verzeichnis strukturiert werden sollten. Es soll also untersucht werden, wie diese Lösungen eingegliedert sowie abstrahiert werden können, um auf dieser Basis Entwicklern eine vielseitig verwendbare Architektur bereitstellen zu können. [26][27][28]

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."[29]. 1, 4, 5, 7, 8

Cluster Siehe Clustering. 3–5, 10, 11

Clustering Eine Technik, die mehrere separate Instanzen einer Ressource nutzt, um Leistungssteigerung und Ausfallsicherheit zu erreichen. 10

CPS *Cyber Physical System*, die Kombination von Geräten mit Elektronische Datenverarbeitung (EDV) Komponenten und der Zusammenführung dieser in ein Netzwerk. 1–5, 8, 9

Docker Eine Ausführungsplattform zum virtualisiertem Ausführen von Containern. 3, 10, 11

Docker Swarm Ein auf Docker aufsetzendens System zum Verteilen von Containern innerhalb eines Clusters. 3

EDV *Elektronische Datenverarbeitung*, das Bearbeiten und Erfassen von Daten durch computergestützte Systeme. 10

FPGA Ein Integrierter Schaltkreis der nach dem Fertigungsprozess von Entwicklern konfiguriert werden kann. 3

Google Protocol Buffer Ein von Google entwickeltes Protokoll zum sprachunabhängigem Austausch von Daten. 6

HTTP *Hypertext Transfer Protocol*, Zustandsloses Protokoll zur Übertragung von annotierten Daten. Kernbestandteil des heutigen Internets. 11

I2C Ein Busprotokoll für die Ansteuerung von Teilkomponenten innerhalb einer Hardwarekomponente. 7

IoT *Internet of Things*, die Vernetzung von Geräten zum Datenaustausch. 8

JPEG Ein Verlustbehaftetes Kompressionsformat für Bilddaten. 7

JSON Ein leichtgewichtiges Datenformat, welches von Menschen leicht lesbar und von Maschinen leicht verarbeitbar sein soll. 6

Kubernetes Ein auf Docker aufsetzendes Containerökosystem zum Bilden eines Clusters. 3

Linux Ein offenes und oft genutztes Betriebssystem mit einer Vielzahl von Anwendungsgebieten. 3, 11

Message Broker Ein Programm dessen Aufgabe es ist, Nachrichten zwischen verschiedenen Programmen über unterschiedliche Protokolle auszutauschen. Eine Unterart der Middleware. 3, 5, 7

Middleware Ein universelles Verbindungsprogramm zwischen verschiedenen Anwendungen. 11

MIME Type Eine Kennzeichnunsweise für Typ und Format von Dateien im Kontext von E-Mail und Hypertext Transfer Protocol (HTTP). 6

Open Source Softwareprojekte, deren Quellcode der Allgemeinheit frei zugänglich ist. 5

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. 5–9

QoS Das Angebot einer Funktion mit garantierten Parametern im Bereich der Dienstgüte. 5, 7

Retainment Das Umwandeln eins, auf einem Topic veröffentlichen, Ereignisses in den permanenten Zustand des Topics. 5

Smart Environment Eine Umgebung mit vernetzten automatisierbaren Geräten und Computern. 1, 7, 8, 11

Smart Home Ein Haushalt mit Smart Environment Fähigkeiten. 1

systemd Eine Softwaresuite zum zum Bereitstellen von Basisfunktionen unter Linux. 8

Literatur

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

[2] Rolf Isermann, Ralf Schwarz und Stefan Stolzl. “Fault-tolerant drive-by-wire systems”. In: *IEEE Control Systems* 22.5 (2002), S. 64–81.

[3] Jim Lamberson. “Single and Multistage Watchdog Timers”. In: *PDF). Sensoray. Retrieved* 10 (2013).

[4] Vinod Chandra und MR Verma. “A fail-safe interlocking system for railways”. In: *IEEE Design & Test of Computers* 8.1 (1991), S. 58–66.

[5] Yiyi Huang u. a. “Diagnosing network disruptions with network-wide analysis”. In: *ACM SIGMETRICS Performance Evaluation Review*. Bd. 35. 1. ACM. 2007, S. 61–72.

[6] *Docker*. Englisch. URL: https://www.docker.com (besucht am 06. 07. 2018).

[7] *Kubernetes*. Englisch. URL: https://kubernetes.io (besucht am 06. 07. 2018).

[8] Alexander Sowitzki. “Eine Infrastruktur für Cyber Physical Systems”. 10. Feb. 2018. URL: https://dev.eqrx.net/research/gpj.pdf (besucht am 06. 07. 2018).

[9] Jonathan Corbet, Alessandro Rubini und Greg Kroah-Hartman. *Linux Device Drivers: Where the Kernel Meets the Hardware*. Ö'Reilly Media, Inc.", 2005.

[10] *Mosquitto*. Englisch. URL: https://mosquitto.org (besucht am 06. 07. 2018).

[11] Satyavrat Wagle. “Semantic data extraction over MQTT for IoTcentric wireless sensor networks”. In: *Internet of Things and Applications (IOTA), International Conference on*. IEEE. 2016, S. 227–232.

[12] Patrick Th Eugster u. a. “The many faces of publish/subscribe”. In: *ACM computing surveys (CSUR)* 35.2 (2003), S. 114–131.

[13] *Clustering MQTT – Introduction & Benefits*. Englisch. URL: https://www.hivemq.com/blog/clustering-mqtt-introduction-benefits (besucht am 06. 07. 2018).

[14] Dinesh Thangavel u. a. “Performance evaluation of MQTT and CoAP via a common middleware”. In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE. 2014, S. 1–6.

[15] *JSON*. Englisch. URL: http://www.json.org (besucht am 06. 07. 2018).

[16] Kazuaki Maeda. “Performance evaluation of object serialization libraries in XML, JSON and binary formats”. In: *Digital Information and Communication Technology and it’s Applications (DICTAP), 2012 Second International Conference on*. IEEE. 2012, S. 177–182.

[17] *Protocol Buffers*. Englisch. URL: https://developers.google.com/protocol-buffers (besucht am 06. 07. 2018).

[18] Adrian Sampson u. a. “EnerJ: Approximate data types for safe and general low-power computation”. In: *ACM SIGPLAN Notices*. Bd. 46. 6. ACM. 2011, S. 164–174.

[19] Arne Bröring u. a. “Enabling IoT ecosystems through platform interoperability”. In: *IEEE software* 34.1 (2017), S. 54–61.

[20] Christian Lesjak u. a. “Securing smart maintenance services: Hardware-security and TLS for MQTT”. In: *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*. IEEE. 2015, S. 1243–1250.

[21] Alexander Sowitzki. “Cyber Physical Systems”. 31. Aug. 2017. URL: https://dev.eqrx.net/research/gsem.pdf (besucht am 06. 07. 2018).

[22] *Docker*. Englisch. URL: https://iotcon.de/en (besucht am 06. 07. 2018).

[23] *Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. Englisch. URL: https://icps2018.net (besucht am 06. 07. 2018).

[24] *The International Conference on Intelligent Environments*. Englisch. URL: http://www.intenv.org (besucht am 06. 07. 2018).

[25] Tobias Eichler. “Agentenbasierte Middleware zur Entwicklerunterstützung in einem Smart-Home-Labor”. 2. Okt. 2014. URL: https://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/master/eichler.pdf (besucht am 06. 07. 2018).

[26] *Node-RED*. Englisch. URL: https://nodered.org (besucht am 06. 07. 2018).

[27] *Esper*. Englisch. URL: http://www.espertech.com/esper (besucht am 06. 07. 2018).

[28] *openHAB*. Englisch. URL: https://www.openhab.org (besucht am 06. 07. 2018).

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