Projektarbeit im
Studiengang IMIT-Angewandte Informatik (BSc)

Experimentelles Container-Deployment auf Industrie 4.0 Geräte

Monika Staciwa

296789

staciwa@uni-hildesheim.de

Betreuer:
Dr. Holger Eichelberger, SSE
Prof. Dr. Klaus Schmid, SSE

Arbeitsgruppe Software Systems Engineering • Institut für Informatik
Universität Hildesheim • Universitätsplatz 1 • D-31134 Hildesheim

Eigenständigkeitserklärung

Erklärung über das selbstständige Verfassen von "Experimentelles Container-Deployment auf Industrie 4.0 Geräte"

Ich versichere hiermit, dass ich die vorstehende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der obigen Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, habe ich in jedem Fall durch die Angabe der Quelle bzw. der Herkunft, auch der benutzten Sekundärliteratur, als Entlehnung kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet und anderen elektronischen Text- und Datensammlungen und dergleichen. Die eingereichte Arbeit ist nicht anderweitig als Prüfungsleistung verwendet worden oder in deutscher oder einer anderen Sprache als Veröffentlichung erschienen. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

Hildesheim, den 31.10.2020 _____

Monika Staciwa

Kurzfassung

In diesem Projekt wurde ein exemplarisches Industrie4.0-System mit Hilfe der Referenzimplementierung BaSyx aufgebaut. Das System besteht aus einer Verwaltungsschale, die die Steuerung von einem simulierten Milchpasteur ermöglicht. Das System ist durch die Benutzung von Docker Containern auf mehreren Geräten, Desktop-Computer, Raspberry Pi und PLCNext-Gerät, realisiert. Anschließend wurde auch die Performanz der Kommunikation zwischen den Geräten gemessen. Diese Arbeit bespricht die Möglichkeiten und die Hindernisse bei der Umsetzung.

Abstract

In this project an example Industry4.0-System was build using the reference implementation BaSyx. This system consists of the Asset Administration Shell, which controls a simulated milk pasteurizator. By using Docker containers, the system is deployed on several devices – desktop computer, Raspberry Pi and PLCNext Control device. Additionally the communication speed between the devices is measured. This paper discusses the possibilities and challenges of the system realization.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AAS	Asset Administration Shell
HTTP	Hypertext Transfer Protocol
I4.0	Industrie 4.0
OPC UA	OPC Unified Architecture
RAMI 4.0	Reference Architectural Model Industry 4.0
REST	Representational state transfer
SDK	Software Development Kit
VAB	Virtual Automation Bus
VWS	Verwaltungsschale

1 Einleitung

Auf den Märkten und in der Gesellschaft kann man einen Wandel beobachten, der den Produktlebenszyklus verkürzt und hohe Flexibilität von den Herstellern verlangt. Als Reaktion auf diese Entwicklung sollen die Produktionsanlagen intelligent werden. Daher sollen einzelne Fabrik-Bestandteile mit Hilfe moderner Informations- und Kommunikationstechnik vernetzt sein. Diese Vernetzung und der ständige Zugang zu aktuellen Informationen eröffnet neue Möglichkeiten, wie z.B. die Anwendung von Künstlicher Intelligenz oder Big-Data-Analysen, um die Produktionsprozesse weiter zu optimieren. Es soll die vierte industrielle Revolution – Industrie 4.0 (I4.0) – in Gang gesetzt werden.

Die von der Bundesregierung geförderte Plattform Industrie 4.0 erarbeitet in Kooperation mit Partnern aus der Wirtschaft und Forschung Lösungsansätze und Handlungsempfehlungen. Zwei Arbeitsergebnisse der Plattform sind das Referenzarchitekturmodell Industrie 4.0 (RAMI4.0)[1] und die Verwaltungsschale (VWS) [BMWi]. Es existieren einige Projekte, die die Referenzimplementierung dieser Ideen anstreben. Doch ist die praktische Umsetzung von Industrie 4.0 Konzepten ist noch in der Entwicklung.

In dieser Arbeit wird der Versuch unternommen basierend auf einer Referenzimplementierung ein einfaches Industrie 4.0-System zu erstellen. Die Systemkomponenten sollen mit Hilfe der Containertechnologie Docker auf unterschiedlichen Geräten realisiert werden und die Vernetzung zwischen ihnen getestet werden. Die Arbeit erfolgt im Rahmen des BMWi-Projekts IIP-Ecosphere[2], in dem die Container-Virtualisierung und die Verwaltungsschale als ein wichtiger Grundbaustein einer Plattform kombiniert werden sollen.

Der Zweck dieser Arbeit ist das Sammeln von Erfahrungen. Wir möchten herausfinden, inwieweit dieses Vorhaben umsetzbar ist. Bei der Realisation sollen Beobachtungen und aufgetretene Probleme festgehalten und geschildert werden.

Im nächsten Kapitel werden die einzelnen Bausteine wie RAMI4.0, die Referenzimplementierung BaSyx Middleware und die Docker Technologie vorgestellt. Im dritten Kapitel werden die bisherigen Probleme bei ähnlichen Vorhaben besprochen und unsere Herangehensweise präsentiert. Die Details der Planung und Umsetzung werden in Kapitel 4 beschrieben. Darauf folgt die Bewertung der Ergebnisse und die Performanzmessung. Im Kapitel 6 teilen wir unsere Beobachtungen und die Hindernisse mit, die während der Umsetzung vorgekommen sind. Das letzte Kapitel zieht ein Resümee über das Projekt.

[1] Plattform Industrie4.0 https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.html Stand: 27.10.2020

[2] IIP-Ecosphere Projekt https://www.iip-ecosphere.eu/ Stand: 27.10.2020

2 Verwaltungsschale in einem Industrie 4.0 System

Bevor wir uns der Problemidentifizierung und der Ausarbeitung einer Lösung widmen, wollen wir uns erst auf die Grundlagen konzentrieren. In den folgenden Abschnitten werden die Grundbegriffe der Industrie 4.0 angesprochen und ein Projekt, das die Industrie4.0-Referenzimplementierung realisiert, vorgestellt. Anschließend wird die Docker-Technologie beschrieben.

2.1 Referenzarchitekturmodell Industrie 4.0

Das **Referenzarchitekturmodell Industrie 4.0** (RAMI 4.0)[3] stellt die wichtigsten Aspekte der Industrie 4.0 in einem dreidimensionalen Raum dar (Abb. 1). Die Achse „Life Cycle & Value Stream" bildet den Produktlebenszyklus ab – von der Entwicklung durch die Produktion bis zur Entsorgung eines Produkts. Den Zyklus unterteilt man in zwei Hauptphasen:

- **Typ** - Entwicklung von Bauplänen, Prototypen, Simulationen und ggf. Verbesserungen von Bauplänen
- **Instanz** - tatsächliche Produktion, Service und schließlich Auflösung einer Produktlinie

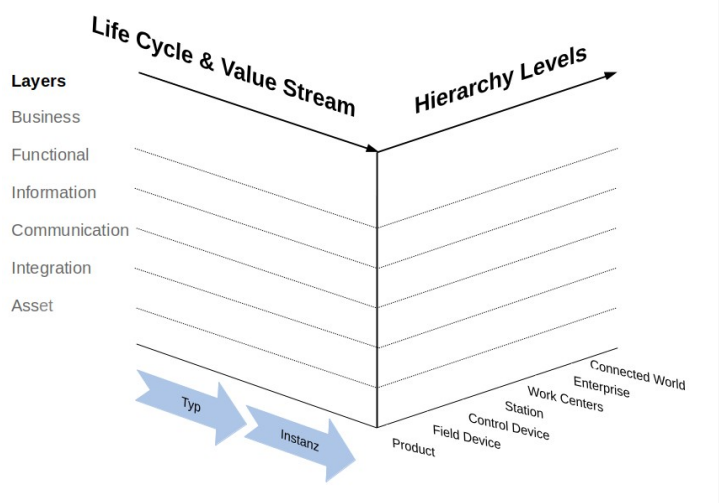

Abbildung 1: Referenzarchitekturmodell Industrie 4.0 nach [RAMI4.0].

Die Achse „Hierarchy Levels" stellt unterschiedliche Hierarchiestufen einer Produktionsanlage nach der IEC 62264 / DIN EN 62264 Normen-Reihe „Integration von Unternehmensführungs- und Leitsystemen" dar. Die letzte Stufe „Connected World" repräsentiert den Zugang zum Internet der Dinge, eine der Grundsäulen der Vernetzung in der Industrie 4.0 [RAMI4.0].

[3] Plattform Industrie4.0 https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.html Stand: 27.10.2020

Die dritte Dimension „Layers“ visualisiert vorwiegend die Festlegungen von geschäftlichen Aspekten der Produktion. In der Schicht „Communication“ definiert man, auf welchem Weg die Informationen zugänglich gemacht werden, also auch wie z.B. die Verwaltungsschalen untereinander kommunizieren sollen [RAMI4.0].

Eine *Verwaltungsschale* (Asset Administration Shell, AAS) wird auch ein digitaler Zwilling genannte. Zusammen mit dem zu verwaltenden Objekt (Asset) formt sie eine *Industrie 4.0 - Komponente.* Ein Asset kann beispielsweise ein Gerät (Säge, Bohrer) sein oder auch ein immaterielles Objekt wie Software oder ein Patent. Die Verwaltungsschale bildet eine Schnittstelle, die den Zugang zu Informationen und Funktionalitäten eines Assets ermöglicht.

Abbildung 2: I4.0-Komponente nach [BMWi, S. 16].

Mehrere I4.0-Komponenten schaffen eine I4.0-System. Eine I4.0-Komponente ist weltweit eindeutig identifizierbar. Die Komponenten eines I4.0-Systems sind normalerweise miteinander verbunden.

Eine Verwaltungsschale soll einen Gegenstand über den gesamten Produktlebenszyklus beschreiben. Der Gegenstand wird durch so genannte *Merkmale* definiert.

> *Merkmal: Werteeigenschaft einer Entität (eindeutig identifizierbarer Gegenstand), die sich in dem für die Anwendung relevanten Betrachtungszeitraum nicht verändert.* [BMWi, S. 12]

Dementsprechend kann für einen Bohrer als Gegenstand sowohl eine Gebrauchsanleitung als auch die Funktion Bohren ein Merkmal sein. Die Merkmale sind innerhalb einer Verwaltungsschale als *Teilmodelle* strukturiert.

Als Ziel wäre es möglich für jeden solchen Aspekt des Assets ein separates Teilmodel zu erstellen. Das bedeutet, dass bei einem Sägemaschinen-Asset ein Teilmodell „Sägen“ innerhalb der Verwaltungsschale zu erwarten ist. Damit lassen sich Sägemaschinen im Allgemeinen leichter vergleichen oder austauschen.

Wenn die Beschaffenheit eines Assets es zulässt, kann die Verwaltungsschale direkt auf dem Asset ausgeführt werden. Ansonsten wird sie zum Beispiel in einer Cloud-Anwendung oder auf einem Edge-Gerät platziert [BMWi].

2.2 Referenzimplementierung - Eclipse BaSyx

Es existieren bereits mehrere Projekte, die die I4.0-Platform umsetzen. Für diese Projektarbeit wird die Referenzimplementierung des Forschungsprojekts BaSys 4.0[4] verwendet - Eclipse BaSyx[5]. Das IIP-Ecosphere Projekt plant die BaSyx Implementierung einzusetzen, daher soll dessen Tauglichkeit überprüft werden.

Es handelt sich um eine OpenSource Implementierung, die in einem Wiki[6] dokumentiert ist. Es standen uns BaSyx Schulungsunterlagen[7] und Anwendungsbeispiele zur Verfügung. Die Anwendungsbeispiele sind in der Zeitschrift iX [Kuh19] zu finden.

BaSyx ist eine Middleware für die Entwicklung von I4.0 Produktionssystemen. Es ermöglicht sowohl zentrale als auch verteile Systeme zu implementieren. Das Projekt besteht aus drei Elementen:

- Software Development Kits (SDKs)
- Referenzimplementierungen
- Beispiele

Die BaSyx SDKs wurden in Java, C# und C++ umgesetzt. Dabei stellt die C++-Version nicht alle Funktionalitäten bereit und ist hauptsächlich für die Integration bestehender Geräte gedacht.

Die BaSyx-Systembestandteile werden als abstrakte Schnittstellen angeboten. Sie werden dann jeweils für unterschiedliche Kommunikationsstandards implementiert: HTTP/REST, OPC UA und ein BaSyx eigenes Kommunikationsprotokoll[8]. Eine Ende-zu-Ende Kommunikation soll durch das BaSyx eigene Kommunikationskonzept Virtual Automation Bus (VAB) gesichert werden. Im Allgemeinen soll der VAB unterschiedliche Protokolle auf BaSyx-native abbilden.

Die BaSyx Middleware ermöglicht die Erstellung einer Verwaltungsschale für ein gegebenes Gerät. Die Eigenschaften und Funktionalitäten des Geräts können übersichtlich in Teilmodelle aufgeteilt werden. Informationen und Funktionalitäten, die Sensoren und Aktuatoren[9] anbieten, werden durch eine Geräteführungskomponente in Echtzeit einer VWS zur Verfügung gestellt.

---

4 BaSys 4 Internseite https://www.basys40.de/, Stand: 27.10.2020

5 Projekt BaSyx https://www.eclipse.org/basyx/, Stand: 29.10.2020

6 Projekt BaSyx Wiki https://wiki.eclipse.org/BaSyx, Stand: 29.10.2020

7 Die Unterlagen sind nicht öffentlich zugänglich.

8 VAB, Projekt BaSyx Wiki https://wiki.eclipse.org/BaSyx_/_Documentation_/_VAB, Stand: 29.10.2020

9 Als Aktor, auch Aktuator werden meist Baueinheiten bezeichnet, die durch ein elektrisches Signal ein gesteuerten Prozess lenken z.B. Schließen und Öffnen von einem Abfluss oder Erhöhung von Temperatur [Wiki].

Die Funktionalitäten von mehreren Geräteführungskomponenten können zu einer abstrakteren Funktionalität zusammengefügt werden. In einem solchem Fall wird die neue Funktionalität durch eine Gruppenführungskomponente gesteuert. VWS und Führungskomponente sind in einem Verzeichnisdienst (Registry) mit eindeutigem Identifier und Adresse angemeldet. Die Registry dient als ein Namensdienst für das System und erlaubt die Verteilung einzelner Komponenten auf mehrere Rechner.

Abbildung 3 veranschaulicht die Architektur eines BaSyx-Systems. Die Anwendung fragt bei der Registry nach der Verwaltungsschale (VWS) für das gewünschtes Gerät. Danach kann man von der Anwendung direkt mit der VWS kommunizieren und durch sie das Gerät auf Feldebene ansteuern [Kuh19, Teil 1].

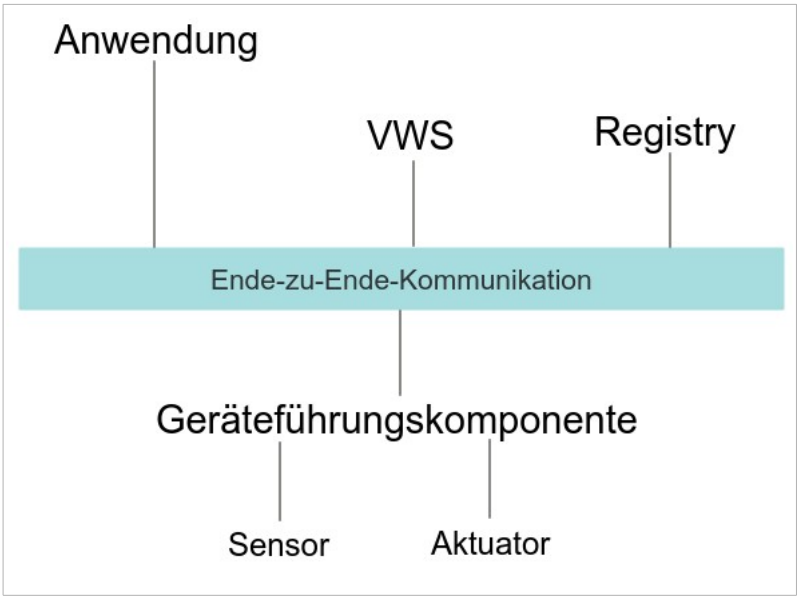

Abbildung 3: Die BaSyx-Architektur nach [Kuh19, Teil 1].

2.3 Containervirtualisierung mit Docker

Docker ermöglicht ein vom Betriebssystem unabhängiges und getrenntes Ausführen von Anwendungen in Docker-Containern und erleichtert damit die Anwendungsbereitstellung.

Unter Linux werden Container durch spezielle Kernelfunktionalitäten realisiert. Diese Art der Virtualisierung gilt als ressourcenschonender im Vergleich mit anderen Virtualisierungstechniken wie z.B. Virtuelle Maschinen, da es nicht notwendig ist für jeden Container ein separates Betriebssystem zu starten/virtualisieren. Dies ist eine wichtige Eigenschaft, denn die I4.0-Systeme sollen auch auf Geräten mit begrenzter Rechen- und Speicherkapazität ausführbar sein.

Docker verwendet unter Windows Hyper-V oder alternativ VirtualBox und unter MacOS VitrualBox als Virtualisierungstechnik. Dies setzt eine Hardwarevirtualisierung voraus.

Ein zentraler Begriff der Docker Technologie ist das Image. Ein *Image* ist eine Vorlage für einen Container und es definiert alles was die in dem Container auszuführende Applikation benötigt [McK17].

Die Plattform Docker Hub bietet zahlreiche Images zum Herunterladen an. Diese Images können direkt verwendet werden oder können als Basis-Images für eigene Anwendungen eingesetzt werden. Um ein eigenes Image zu kreieren braucht man ein *Dockerfile*. Es enthält die Instruktionen, wie das Image erstellt werden soll [Kro17].

Das Dockerfile definiert u.a.:

- auf welchem Image das zu bauende Image basiert,
- welche Dateien ins Dateisystem des Images kopiert werden (z.B. eine Java-Anwendung),
- welche Befehle nach dem Starten des Containers, ausgeführt werden sollen (z.B. für ein Javaprogramm: `java Anwendung.class`),
- wo die Ausgabe-Werte des Images gespeichert werden sollen.

3 Problemstellung

In diesem Kapitel wird die Problematik der praktischen Anwendung von I4.0-Konzepten skizziert und ein Lösungsvorschlag vorgestellt.

Die breite Umsetzung von Industrie 4.0 scheitert momentan unter anderem an der Diversität der Schnittstellen, der verfügbaren Plattformen und Geräte. Verfügbare Standards wie das Referenzarchitekturmodell Industrie 4.0 oder die Verwaltungsschale werden von Plattformanbietern zurzeit, wenn überhaupt, nur zögerlich umgesetzt. Virtualisierung durch Container könnte hier Abhilfe schaffen, indem benötigte Komponenten für die jeweilige Ausführungsumgebung, z.B., ein Edge-Device auf einfache Weise verfügbar gemacht werden können.

Diese Arbeit möchte folgende Punkte untersuchen:

- Ist es möglich eine VWS mit einer der vorhanden Referenzimplementierungen zu erstellen, die realistisch ein exemplarisches I4.0-Asset abbildet?
- Wie aufwendig ist es ein I4.0-System mit einer der vorhanden Referenzimplementierungen zu erstellen?
- Ist es möglich Docker-Images mit einer VWS zu erstellen, die auf unterschiedlichen I4.0-Geräten ausführbar sind?
- Ist die Kommunikation zwischen den Komponenten eines solches Systems gewährleistet?

3.1 Lösungsidee

Um die oben aufgelistete Fragen anzugehen, wollen wir eine Simulation eines I4.0-Asset als ein Java Programm erstellen. Das Programm soll ein reales Gerät insofern nachahmen, so dass die Nachbildung eines Steuer- und Informationsabfragevorgangs möglich ist. Es soll z.B. Funktionalitäten besitzen, deren Aktivierung den Zustand des Geräts verändert.

Die uns zugänglichen BaSyx-Anwendungsbeispiele wurden mit Java 8 umgesetzt. Um mögliche Komplikationen in der ersten Entwicklungsphase zu vermeiden, haben wir uns auch für diese Version der Sprache entschieden. Sollte sich dies als nachteilig erweisen, wird eine neuere Java Version eingesetzt.

Für das exemplarische Asset soll eine VWS mit BaSyx erstellt werden. Um ein funktionsfähiges Gesamtsystem zu ermöglichen, werden weitere Komponenten (Registry, Geräteführungskomponente) der Referenzimplementierung verwendet.

Die Komponenten des Systems sollen dann in Docker-Images eingebunden werden. Die Ausführbarkeit der entsprechenden Docker-Container soll auf folgenden Geräten getestet:

- einem üblichen Desktopcomputer,
- dem Einplatinencomputer Raspberry Pi,
- und einem Gerät der PLCNext Technologie[10]. PLCNext Geräte werden für die Steuerung von Maschinen oder ganzen Anlagen eingesetzt. Sie agieren in Echtzeit. Diese Steuerungsgeräte besitzen das offene Linux-Betriebssystem und unterstützen neben den typischen SPS[11]-Sprachen auch höhere Programmiersprachen.

Anschließend wollen wir eine Kommunikation der einzelnen Komponenten über ein Netzwerk anstoßen. Das heißt, von einer Anwendung die auf einem Rechner läuft, soll die VWS, die sich auf einem anderen Rechner befindet, angesprochen werden und über diese dann die Steuerung einer Asset-Funktionalität getestet werden.

[10] Hersteller Internetseite: https://www.phoenixcontact.com/online/portal/de?1dmy&urile=wcm:path:/dede/web/main/products/subcategory_pages/PLCnext_Controls_P-21-14/30b12f75-d769-4f0e-a783-4986ae3ae247 Stand: 23.10.2020

[11] Speicherprogrammierbare Steuerung

4 Implementierung

In diesem Teil der Arbeit werden die Umsetzungsschritte beschrieben. Erst wird das zu simulierende I4.0-Gerät vorgestellt. Danach wird die Struktur des I4.0-Systems und deren Bestandteilen geschildert. Die Auflistung aller Hard- und Software-Komponenten ist im Punkt 4.2.1 zu finden. Die Realisierung der Komponente als Docker Container erfolgt im letzten Abschnitt des Kapitels.

4.1 Gerät

Als Beispiel für ein Industriegerät wird ein Milchpasteur mit einem Java Programm simuliert. Ein Milchpasteur ist ein Gerät, das Milch durch Erhitzen auf 90°C pasteurisiert. Es besteht aus einem Tank, einem Erhitzer, einem Füll- und Temperatursensor. Als Aktoren dienen eine Pumpe zum Befüllen und Entleeren des Tanks und ein Schalter für die Heizung.

Der Arbeitsvorgang zum Pasteurisieren verläuft in folgenden Schritten:

1. Der Tank wird mit Flüssigkeit gefüllt.
2. Die Flüssigkeit wird bis zur maximalen Temperatur erwärmt.
3. Der Tank wird geleert.

Der Milchpasteur wird wie folgt modelliert:

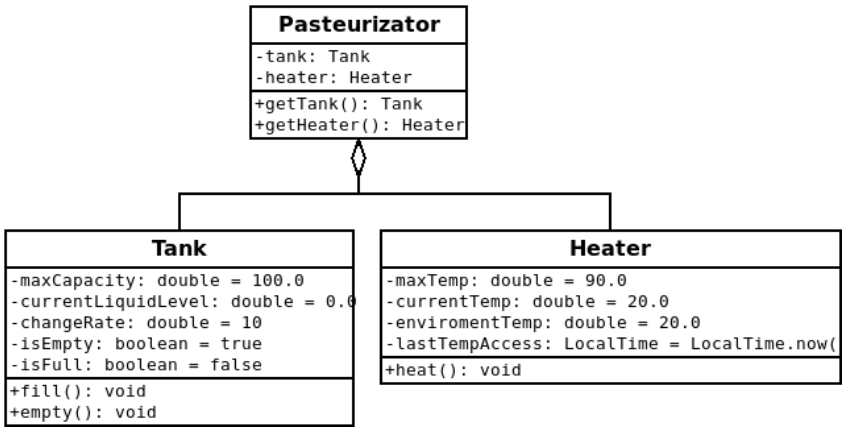

Abbildung 4: Milchpasteur – UML Model.

4.2 Umsetzung - Industrie 4.0 Infrastruktur mit BaSyx

Das gesamte Projekt – Milchpasteur, die VWS und die abfragende Anwendung – ist als Maven Projekt realisiert. Wir haben uns für dieses Werkzeug entschieden, da sowohl die BaSyx Komponente selbst als auch alle uns zur Verfügung stehende Anwendungsbeispiele Maven Projekte sind. Es gibt hier für den Milchpasteur drei Klassen: `Pasteurizator`, `Tank` und `Heater`. `PasteurizatorControlComponent` implementiert die Geräteführungskomponente für den Milchpasteur. Das gesamte System, die VWS, die Registry, die Geräteführungskomponente und der Milchpasteur, wird vom `InfrustructureStarter` initialisiert.
`Client` ist die Anwendung, die über Netzwerk mit der VWS kommuniziert.

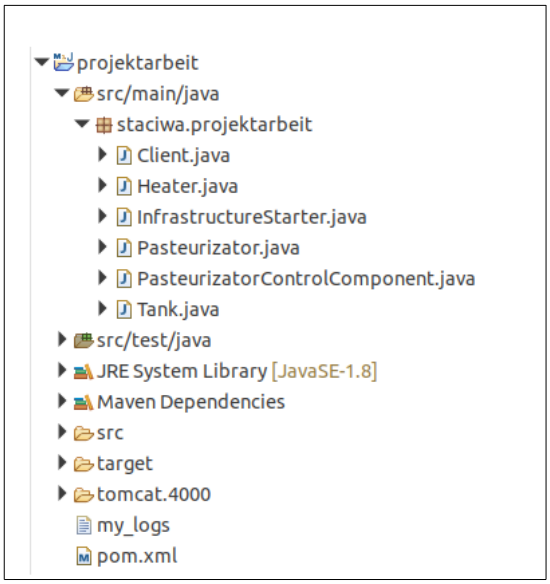

Abbildung 5: Projektarbeit in Eclipse IDE.

Abhängig davon welche Klasse in der pom-Datei als `mainClass` angegeben wird, wird das Projekt von Maven zur einem `InfrastrukturStarter` oder zu einem `Client` kompiliert.

Die VWS für den Milchpasteur ist mit zwei Submodels realisiert – Tank und Heater.

Die Abbildung 6 zeigt wie die VWS mit BaSyx Bausteinen aufgebaut wurde. Die Instanz der Klasse `VABMultiSubmodelProvider` enthält die VWS und ihre Submodelle. Diese Instanz wird in einen `HttpServlet` eingebettet und anschließen auf einen HTTP Server ausgeführt.
Die VWS, die Registry und die Geräteführungskomponente sind unter einer IP-Adresse erreichbar. Die VWS und die Registry sind auf dem Port 4000 und die Geräteführungskomponente auf dem Port 4001 verfügbar.

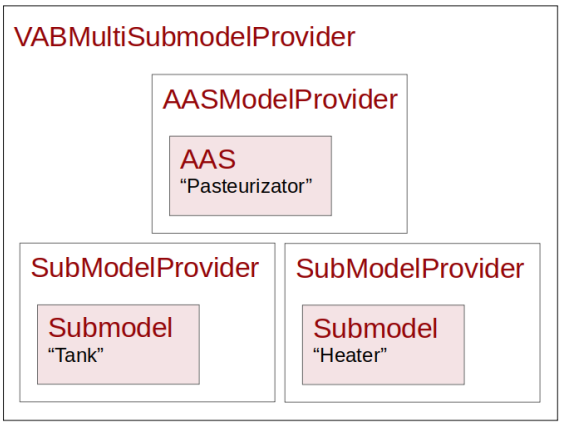

Abbildung 6: Verwaltungsschale realisiert mit BaSyx – Übersicht der Verschachtlung.

4.2.1 Verwendete Hard- und Software

- PC 1: Linux Ubuntu 18.04.5 LTS, 64bit AMD FX-6100 Six-Core, RAM 16GiB
- PC 2: Linux Ubuntu 18.04.5 LTS, 64bit Intel Pentium 2.20GHz x 2, RAM 4GiB
- PC 3: Windows 10 Pro, 64bit Intel Core 1.86 GHz x 2, RAM 4GiB
- Raspberry Pi 3 Model B: Raspbian GNU/Linux 10, 32bit ARMv7, RAM 1GiB
- PCLNext-1 (AXC 3152): Linux BusyBox, x86 64bit 1.3GHz x 2, RAM 2 GiB
- PCLNext-2 (AXC 2152): Linux BusyBox, 32bit ARM CortexA9 800MHz x 2, RAM 0.5 GiB[12]
- Java 8
- Eclipse Basyx: Stand 21.07.2020
  (Commit-Id: 6362cacf6390650a5121648a89f76ad5a8986670)
- Docker Engine: 19.03.13
- Apache Maven 3.6.0
- Eclipse IDE: Version 4.14.0
- Versionsverwaltung: Git

[12] Angabe aus der Hersteller-Internetseite: https://www.phoenixcontact.com/online/portal/de?uri=pxc-oc-itemdetail:pid=2404267&library=dede&tab=1#AllgemeineDaten Stand: 26.10.2020

4.2.2 Deployment mit Docker

Um das Docker-Image zu erzeugen, muss zuerst eine JAR-Datei aus dem Java Projekt mit allen Abhängigkeiten (den benutzen BaSyx Komponenten) erstellt werden. Das JAR-Paket wird in dieser Arbeit mit dem Build-Management-Tool Maven gebaut.

Ein wichtiger Aspekt bei der Erstellung des Docker-Images ist die Entscheidung auf welchem bestehenden Image es basieren soll. Für die Anwendung wird eine Image benötigt, das Java 8 enthält. Zusätzlich ist die Wahl des Basis-Images abhängig von der CPU-Architektur des Gerätes, auf welchem der Docker Container ausgeführt werden soll.

Auf der Plattform Docker Hub werden mehrere Images für die Intel/AMD Architektur, die Java 8 beinhalten, angeboten. Für dieses Projekt habe ich zwei Images ausprobiert - das offizielle Oracle `java`-Image und das `openjdk`-Image. Beide Images haben gut funktioniert. Wir haben uns letztendlich für das `openjdk`-Image entschieden. OpenJDK ist OpenSource und herstellerneutral, was im Rahmen des IIP-Ecosphere Projekts gewünscht ist.

Das `openjdk`-Image ist sowohl mit einem `x86-64,` als auch einem `ARM`-Tag gekennzeichnet. Dies ist aber vermutlich ein Fehler. Images, die auf dem `openjdk`-Image basierten, waren auf dem Raspberry Pi (ARM CPU) nicht ausführbar. Statt dessen wurden die Images für die ARM-Prozessor-Geräte mit dem `armv7/armhf-java8`-Image gebaut. Es war das einzige Image mit Java 8 für ARM, das auf Docker Hub zu finden war.

Tabelle 1: Aufteilung einzelner Systemkomponenten in Docker-Images

Image	**Inhalt**	**Basis-Image**	**Host-Rechner**
`InfrastructureStarter`	VWS, Registry, Control Component	openjdk	PC-2, PLCNext-1
		armv7/armhf-java8	Raspberry Pi
`Client`	abfragende Anwendung	openjdk	PC-1, Windows-PC

Die Container wurden in drei Varianten erstellt. In der ersten Variante werden beide, der `Client` und der `InfrastructurStarter`, auf zwei Desktop-Rechnern (PC 1 und PC 2) ausgeführt (siehe Punkt 4.2.1). In der zweiten Variante läuft die VWS auf dem Raspberry Pi und wird von PC 1 angesprochen. In der letzten Variante befindet sich die VWS auf einem PLCNext-Gerät und wird von PC 3 angesprochen. In allen Varianten befinden sich die Rechner innerhalb eines Netzwerks. Abbildung 7 veranschaulicht die Verteilung der Komponenten.

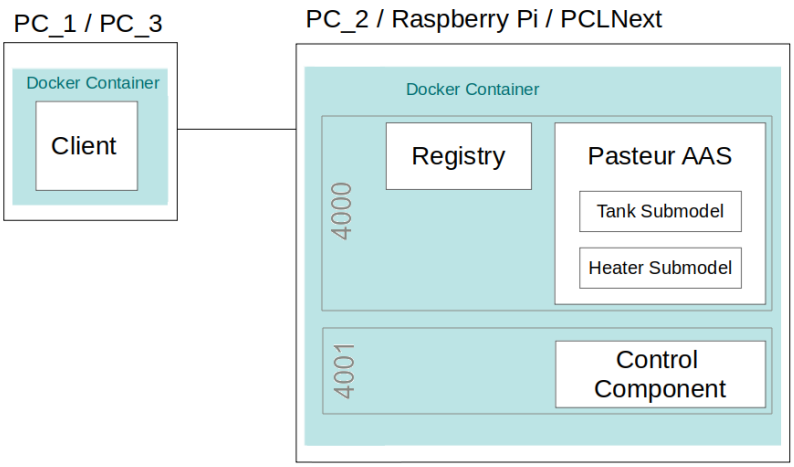

Abbildung 7: Die Verteilung der Infrastruktur-Komponenten

5 Validierung

Es war möglich einen Großteil des Plans aus Kapitel 3 umsetzen. Es wurde zusätzlich die Kommunikationsgeschwindigkeit zwischen den einzelnen Geräten gemessen. In diesem Kapitel werden die Ergebnisse präsentiert und die Probleme besprochen.

Die Erstellung von ausführbaren Docker Containern mit dem `InfrastrukturStarter` war für alle Geräte außer den PLCNext-2 AXC 2152 möglich. Das Gerät verfügte nicht über genug Speicherplatz. Eine Volumenreduktion des Projekts war nicht möglich, da alle Komponenten des Systems auf einem Gerät realisiert werden sollten. Würde man jedoch die Struktur des Systems ändern und beispielsweise die Geräteführungskomponente auf den AXC 2152 separat realisieren, wäre dies durchführbar.

Um zu überprüfen, ob die Kommunikation zwischen den Containern möglich ist und alle Funktionalitäten verfügbar sind, wurde eine abfragende Anwendung, der `Client` geschrieben. Die Anwendung ist wie in Abb. 7 dargestellt, realisiert.

Der `Client` verbindet sich über die VWS mit beiden Submodellen – `Heater` und `Tank`. Bei dem Tank erfragt er die maximale Füllkapazität und den aktuellen Flüssigkeitsstand. Von dem Erhitzer bekommt er Information über die aktuelle Temperatur. Danach ruft der `Client` die Funktion `fillTank` auf. Ist die maximale Füllkapazität erreicht wird automatisch der Erhitzer aktiviert. Er ist so lange aktiv bis die Temperatur auf 90°C gestiegen ist. Infolgedessen schaltet sich der Erhitzer ab.

```
Welcome to the Pasteurizator Simulation CLIENT

Submodel "Tank" with short ID "submodel_tank" has been retrieved.

Submodel "Heater" with short ID "submodel_heater" has been retrieved.

        The maximal capacity of the tank: 100.0
        Current liquid level: 0.0
        Current temperatur of the liquid: 20.0°C

Filling the tank with liquid and heating the liquid.
        Current liquid level: 100.0
        Current temperatur of the liquid: 89.85°C
        Temperatur of the liquid after 5s: 89.6°C

Emptying the tank.
        Current liquid level: 0.0
```

Abbildung 8: Anwendung Client – Ausgabewerte.

Der `Client` erfragt erneut den Flüssigkeitsstand und -temperatur im Tank. Im Anschluss daran wartet das Programm fünf Sekunden und erfragt die Temperatur zum dritten Mal. Schließlich

wird die Methode `emptyTank` aufgerufen und zuletzt der Flüssigkeitsstand abgefragt. Die Programmausgabewerte sind in Abbildung 8 präsentiert.

Dieser Vorgang wurde in alle drei Varianten positiv getestet:

PC 1 ↔ PC 2

PC 1 ↔ Raspberry Pi

PC 3 ↔ PLCNext-1

5.1 Performanzmessung

Es wurde die Performanz des erstellten I4.0-System gemessen. Dafür wurde ein weiteres Projekt `performanzmessung` erstellt, das auf dem Projekt `projektarbeit` basiert. Mit Hilfe dieses neuen Projektes sollten die Dauer der Property-Abfragen und der Funktionsaufrufe festgestellt werden. Als Referenz wurde der Java Standardweg für Remote Procedure Call (RPC): Java RMI (Remote Method Invocation)[13] verwendet.

Die Klasse `InfrastructureStarter` wurde von `projektarbeit` übernommen und um eine neue Test-Funktion des Tanks erweitert. Der `Client` erfragt ein statisches (die maximale Tank-Kapazität) und ein dynamisches Attribut (der aktuelle Flüssigkeitsstand im Tank) des Milchpasteurs. Zusätzlich macht er einen Test-Funktionsaufruf. Die neue Test-Funktion gibt nur die Zeichenkette „Test Funktion“ zurück.

Das `performanzmessung` Projekt und der RMI Funktionsaufruf werden analog zu dem `projektarbeit` Projekt ausgeführt, d.h. die Komponenten sind in Containern realisiert (wie in Abb. 7). Die Verteilung der Container auf unterschiedlichen Geräten ist in der Tabelle 2 abzulesen.

Tabelle 2: Performanzmessung – Verteilung der Komponente auf Geräten.

BaSyx Client/ RMI Client	**BaSyx InfrastructureStarter/ RMI Server**
PC 1	PC 2
PC 1	Raspberry Pi
PC 3	PLCNext-1

[13] Oracle Internetseite https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html Stand: 28.10.2020

Jede der Anfragen/Aufrufe wird 10.000 Mal nacheinander wiederholt. Die Werte in Tabelle 3 sind Durchschnittswerte.

Tabelle 3: Performanzmessung – durchschnittliche Werte.

		PC	Raspberry Pi	PCLNext AXC 3152
		in Millisekunden		
BaSyx	statischer Attribut	9,2	9,7	3,96
	dynamischer Attribut	5,8	7,8	2,9
	Funktionsaufruf	22,8	25,3	22,8
RMI Funktionsaufruf		3,9	4	1,81

Im Allgemeinen kann man sagen, dass ein BaSyx Funktionsaufruf deutlich länger als eine Attribut-Abfrage dauert. Im Vergleich mit Java RMI braucht ein BaSyx Funktionsaufruf durchschnittlich um fast 20 Millisekunden länger. Der BaSyx-Middleware kommt mit einem gewissen Overhead.

Der Raspberry Pi scheint ein wenig langsamer als ein Desktop Computer zu sein – es braucht im Durschnitt etwa 2,5 Millisekunden länger für einen Funktionsaufruf.

Bei Property-Abfragen und RMI ist das PLCNext Gerät AXC 3152 deutlich performanter als die restlichen Geräte. Dafür ist bei einem BaSyx Funktionsaufruf diese Tendenz nicht mehr sichtbar.

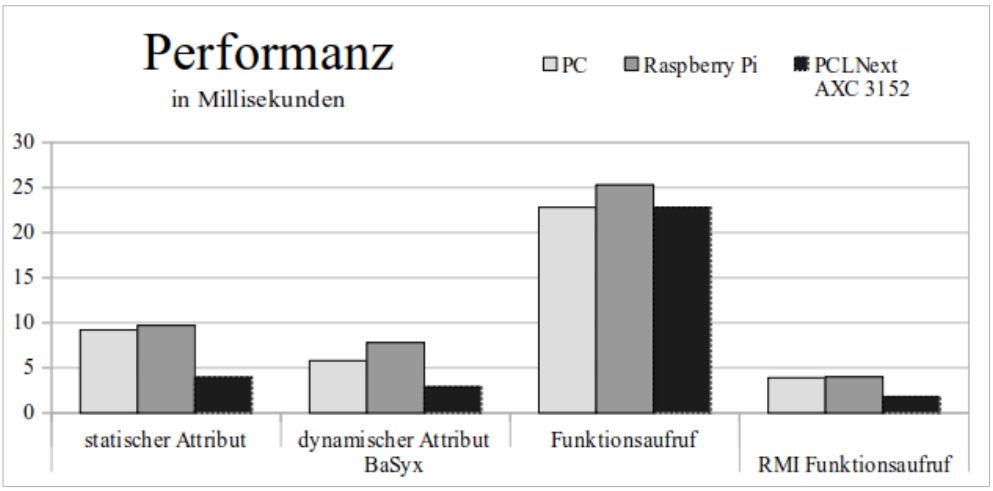

Abbildung 9: Performanzmessung – Ergebnisse.

6 Diskussion

In diesem Kapitel möchte ich die Probleme schildern, die mich viel Zeit bei der Lösung gekostet haben.

Die erste Hürde bei der Arbeit mit BaSyx kann das Einrichten eines Projekts in einer Entwicklungsumgebung sein. Eine gute Einleitung dazu findet man im iX Artikel [Kuh19].

Die Arbeit mit BaSyx wird dadurch erschwert, dass keine Versionierung existiert, weder im Bezug auf die API noch in Bezug auf das verwendete Kommunikationsprotokoll. Es ist dadurch nicht ersichtlich wann es zu inkompatiblen Änderungen bei verschieden Softwareständen kommt. Es führte dazu, dass die BaSyx Software, die an unterschiedlichen Tagen herunterladen wurde, zwei unterschiedliche nicht kompatible Softwarestände waren. In der frühen Phase der Entwicklung hatte ich, ohne es zu wissen, zwei Varianten der BaSyx Komponenten auf zwei unterschiedlichen Rechnern. Infolgedessen war es nicht möglich die Verwaltungsschale und den Client über das Netzwerk zu verbinden. Die Fehlermeldung war nicht aussagekräftig. Daher ist es auf jeden Fall ratsam eine Variante der BaSyx Komponente zu speichern und nur mit dieser zu arbeiten.

Ein weiteres Problem, das die fehlende Versionierung verursacht hat, ist, dass die Beispiele des Tutorials aus der Zeitschrift iX [Kuh19] auf einer älteren Variation des Programms basierten und damit leider nicht mehr anwendbar waren.

In der Dokumentation auf der Internetseite des BaSyx-Projekts waren hilfreiche Beispiele zu finden[14]. Sie ermöglichen einen guten Einstieg in die Konzepte von BaSyx. Ansonsten wirkt die Dokumentation eher chaotisch aufgebaut. Es gibt z.B. zwei Unterseiten die der Control Component gewidmet sind. Eine Seite kann man gut über das Hauptmenü erreichen. Die andere Seite mit mehr detaillierten Angaben habe ich erst über eine Suchmaschine finden können.[15]

Daher mussten doch einige Aspekte durch Trial and Error erforscht werden. Beispielweise bin ich auf Probleme gestoßen, wenn ich eine Methode zweimal nacheinander aufrufen wollte. Es hat sich herausgestellt, dass die Control Component wie eine Zustandsmaschine agiert. Beim Aufruf einer bereit gestellten Funktionalität ändert sich vermutlich der Zustand der Control Component von IDLE auf EXECUTE. Beim Entwerfen einer Methode in der Control Component muss man darauf achten den Zustand am Ende auf COMPLETE zu setzen. Bleibt die Control Component in Zustand EXECUTE, ist ein wiederholter Funktionsaufruf nicht möglich.

[14] Eclipse BaSyx Wiki https://wiki.eclipse.org/BaSyx_/_Introductory_Examples, Stand: 27.10.2020

[15] https://wiki.eclipse.org/BaSyx_/_Documentation_/_ControlComponent, Stand: 14.10.2020
https://wiki.eclipse.org/BaSyx_/_Documentation_/_API_/_ControlComponent, Stand: 14.10.2020

Eine andere Erkenntnis, die ich gewonnen habe, bezieht sich auf das Netzwerk zwischen den BaSyx Komponenten. Die IP-Adresse einer Verwaltungsschale kann nicht als „localhost“ angegeben werden. In einem solchen Fall startet die VWS korrekt und ist von Applikationen, die auf demselben Rechner laufen, ansprechbar. Es ist aber nicht möglich die Verwaltungsschale von außen anzusprechen. Es muss dafür die IP-Adresse des Geräts, die es im Netzwerk besitzt, angegeben werden.

Eine zusätzliche Herausforderung bei der Umsetzung des Projekts war, dass ich außer der BaSyx Middleware sehr viele andere für mich neue Konzepte und Programme auf einmal lernen musste.
Zum Beispiel hat sich die Erstellung einer korrekten pom.xml für das Build Tool Maven als zeitaufwendig gezeigt. Hat man keine Erfahrung mit Maven, sollte schon eine gewisse Zeit für die Einarbeitung einplanen werden.

Darüber hinaus muss auch die Funktionsweise von Docker erlernt werden und gewisse Aspekte des Netzwerk-Kommunikation beachtet werden. Beispielsweise muss beim Starten des Docker Containers die Portweiterleitung vom Container auf den Host berücksichtigt werden. Und zwar nicht nur für der Verwaltungsschale, sondern auch entsprechend für die Registry und die Control Component.

Erwähnenswert ist auch die Tatsache, dass ein Windows/Mac-Rechner, auf welchem Docker laufen soll, Hardware-Virtualisierung unterstützten muss. Diese Funktionalität muss im BIOS aktiviert sein. Falls man mit Docker unter Windows arbeitet, muss der Benutzer ein Mitglied der Benutzergruppe Docker sein. Bei den PLCNext Geräten sind Root-Rechte notwendig, um den Docker Demon zu starten, der für das Arbeiten mit den Docker-Images und Containern notwendig ist. Beim AXC 2152 darf der Container nicht zu groß sein, da das Gerät nur begrenzten Speicherplatz besitzt.

7 Schlussbemerkungen

Es war möglich eine funktionierende Industrie 4.0 Infrastruktur mit der Referenzimplementierung BaSyx zu erstellen. Das simulierte I4.0-Gerät konnte zufriedenstellend als digitaler Zwilling nachgebaut werden. Die Funktionalitäten des Milchpasteurs ließen sich durch die VWS für externe Anwendungen bereitstellen.

Die einzelnen System-Komponenten konnten in unterschiedlichen Umgebungen und auf allen Geräten, mit Ausnahme des PLCNext AXC 2152, ausgeführt werden. Das Problem des fehlenden Speicherplatzes auf dem AXC 2152 wurde nicht näher angegangen. Es schließt das Gerät aber nicht aus, da sich das Problem lösen lässt, indem man das Volumen der zu realisierenden Komponenten reduziert. Im Allgemeinen hat sich der Einsatz der Docker Technologie als unproblematisch herausgestellt.

Die Kommunikation zwischen einzelnen Komponenten ist gewährleistet. Trotz eines gewissen Overheads, das das BaSyx Framework mit sich bringt, ist die Kommunikationsgeschwindigkeit als gut zu bewerten. Zusätzlich hat sich das PLCNext Gerät AXC 3152 als besonders performant herausgestellt.

Für die weitere Entwicklung wäre es interessant eine Infrastruktur mit mehreren Verwaltungsschalen verteilt auf unterschiedlichen Geräten zu erstellen. Als Konsequenz daraus könnte man auch die Registry und die Geräteführungskomponenten auf getrennten Rechner verteilen und die Kommunikationsmöglichkeiten zwischen allen Akteuren untersuchen.

Schließlich wäre das Auslesen von realen Umgebungsdaten eines Geräts über ein Industrie 4.0-Protokoll ein weiterer Entwicklungsschritt.

Literaturverzeichnis

[BaSyx] Eclipse BaSyx Projekt Dokumentation. Online verfügbar: https://wiki.eclipse.org/BaSyx, Letzter Besuch: 10.10.2020

[BMWi] Verwaltungsschale in der Praxis. Herausgegeben von Bundesministerium für Wirtschaft und Energie (BMWi). 2020. Online verfügbar https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2020-verwaltungsschale-in-der-praxis.html, Letzter Besuch: 23.10.2020.

[Kro17] J. Krochmalski. Docker and Kubernetes for Java Developers. Packt Publishing Ltd. Birmingham, 2017.

[Kuh19] T. Kuhn, F. Schnicke, C. Ziesche. BaSys Tutorial, Teil 1-3. Magazin für professionelle Informationstechnik (iX). Heise Zeitschriften Verlag, 2019.

[McK17] R. McKendrick , S. Gallagher. Mastering Docker. 2. Auflage. Packt Publishing Ltd. Birmingham, 2017.

[RAMI4.0] RAMI 4.0 – Ein Orientierungsrahmen für die Digitalisierung. Herausgegeben von Bundesministerium für Wirtschaft und Energie (BMWi). Online verfügbar https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/rami40-einfuehrung-2018.html, Letzter Besuch: 23.10.2020.

[Wiki] Wikipedia, Artikel: Aktor. Online verfügbar https://de.wikipedia.org/wiki/Aktor, Letzter Besuch: 31.10.2020.