Masterarbeit

DATENERFASSUNG FÜR BIG DATA

ausgeführt am

CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von
Bernhard Roßmann, BSc
1910322034

betreut und begutachtet von
FH-Prof. DI Dieter Lutzmayr

Graz, im Jänner 2021

Bernhard Roß
.....
Unterschrift

EHRENWÖRTLICHE ERKLÄRUNG

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

.....

Unterschrift

DANKSAGUNG

An dieser Stelle möchte ich mich bei allen bedanken, die mich im Laufe meines Studiums und vor allem während der Erstellung dieser Masterarbeit unterstützt haben. Vor Allem möchte ich meinen Dank an meinen Betreuer Herrn FH-Prof. DI Dieter Lutzmayr aussprechen, der mich, während den zwei Bachelorarbeiten und dieser Masterarbeit, betreut hat. Ebenso möchte ich mich bei meiner Familie und Freunden für die moralische Unterstützend während des Studiums herzlichst bedanken. Auch für die erhaltenen Vorschläge und dem Korrekturlesen bin ich sehr dankbar.

KURZFASSUNG

Big Data und Industrie 4.0 sind Schlagwörter, die in der Automatisierung omnipräsent geworden sind. Mit dem immer größer werdenden Angebot von Cloudanwendungen und einer schleichenden Abkehr der Grundeinstellung, dass sämtliche Daten im Unternehmen gespeichert werden müssen, tut sich eine Vielzahl von Möglichkeiten in der Automatisierungstechnik auf. Diese neuen Technologien, für künstliche Intelligenz oder Predictive-Maintenance, benötigen Daten, um ihre Aufgaben auszuführen, haben aber meist keine Werkzeuge für die Datenerfassung. Durch die langen Laufzeiten für Anlagen (> 20 Jahre) in der Automatisierungstechnik kann es bei Nachrüstungen zu Problemen kommen. In der Feldebene, der untersten Schicht in der Automatisierungstechnik, findet sich eine große Anzahl von Protokollen, die nicht in jeder neuen Technologie verfügbar sind. Genau an diesem Problem knüpft diese Arbeit an und es wird ein System entwickelt, welches von verschiedenen Protokollen Daten erfassen und diese an unterschiedliche Online-Datenbanken transferieren kann. In dieser Arbeit sollen wichtige Aspekte für ein solches System erörtert werden. Eingangs gibt der theoretische Teil einen Themenaufriss zur Industriellen Kommunikation. Genaueres Augenmerk wird auf eine sichere Datenübertragung gelegt, denn bei der Übertragung an Online-Datenbanken muss die Kommunikation verschlüsselt erfolgen. Es wird aber auch generelles Cloudcomputing behandelt, wobei der Fokus auf Online-Datenbanken und den Datenbankmodellen liegt. Das zu realisierende System soll so aufgebaut werden, dass zukünftig andere Protokolle für die Datenerfassung oder neue Online-Datenbank-Anbindungen hinzugefügt werden können. Im praktischen Teil wird das System, unter Berücksichtigung der erarbeiteten Resultate, umgesetzt und auf die Tauglichkeit in der Industrie untersucht.

ABSTRACT

Big Data and Industry 4.0 are keywords that have become omnipresent in automation. With the increasing range of cloud applications and a creeping departure from the basic attitude that all data must be stored in the company, a wide range of opportunities are opening up in automation technology. These new technologies for artificial intelligence or predictive maintenance require data to perform their tasks, but usually do not have tools for data collection. Due to the fact that in automation technology the lifetimes of plants are often more than 20 years, problems can arise in case of retrofitting. In the fieldbus, the lowest layer in automation technology, there is a large quantity of protocols that are not available in every new technology. This specific issue is the starting point of this work. The aim is to develop a system which can collect data from different protocols and transfer them to various online databases. Important aspects for such a system shall be discussed. The theoretical part starts with an overview of the topic of industrial communication. More attention will be paid to secure data transmission, as the communication must be encrypted during the transfer to online databases. Furthermore, general cloud computing, with a focus on online databases and database models will be discussed. The system to be developed is to be set up in a way, that in the future further protocols for data acquisition or new online database connections can be added. In the practical part, the system is developed and tested for suitability in industry.

INHALTSVERZEICHNIS

1 EINLEITUNG

Diese Masterarbeit befasst sich mit der Datenerfassung für *Big Data*. Neue Technologien wie *Machine Learning* oder *Predictive Maintenance* benötigen verschiedenste Daten, um zu funktionieren. In der Automatisierungstechnik sind Daten zwar im Überfluss vorhanden, doch aufgrund einer Vielzahl von proprietären Protokollen gestaltet sich die Datenerfassung oftmals schwierig. Um Daten für die neuen Technologien auf einem einheitlichen Weg zu erfassen, setzt sich diese Arbeit mit den wissenschaftlichen Aspekten zur Datenerfassung in der Automatisierungstechnik auseinander. Für die Speicherung der Daten sollen Cloudanwendungen und dabei im Speziellen Online-Datenbanken in Betracht gezogen werden.

Einleitend ist die Motivation für die Auswahl dieses Thema beschrieben, anschließend werden die Ziele, welche in der Arbeit erreicht werden sollen, definiert. Abschließend wird noch ein Überblick über die Gliederung der Arbeit gegeben.

1.1 Motivation

In der Welt der Automatisierungstechnik werden eine Vielzahl von Anlagen und Maschinen vertrieben, die verschiedenste Geräte von unterschiedlichen HerstellerInnen beinhalten. Heutige Anlagen und Maschinen sind hoch automatisiert und produzieren laufend enorme Datenmengen. Um die neuen Technologien von *Industrie 4.0* nutzen zu können müssen diese Daten den neuen Applikationen zugänglich gemacht werden. Gewisse Aufgaben, wie zum Beispiel Training für *künstliche Intelligenz,* können offline erledigt werden, doch wenn eine Anlage in Realtime überwacht werden soll, müssen die Daten ständig von den Applikationen geholt werden. In der Automatisierungstechnik gibt es viele unterschiedliche Protokolle, mit denen Geräte in der untersten Ebene kommunizieren. Grund hierfür ist die Tatsache, dass seit der Einführung von Bussystemen kein einheitlicher Standard definiert wurde und so die Hardware-HerstellerInnen ihre eigenen Protokolle bevorzugen. Von den meisten Hardware-HerstellerInnen werden zwar Zusatzkomponenten für andere Protokolle angeboten, jedoch müssen diese für jedes Gerät zugekauft werden. Sollen die Prozessdaten in eine spezielle Zieldestination (Datenbank, Online-Datenbank, Datenformat, usw.) transferiert werden, stellen die HerstellerInnen oder DrittanwenderIn Werkzeuge zur Verfügung. Bei spezielleren Anforderungen bedarf es oftmals eines Workarounds. Speziell für die Datenerfassung verschiedener Protokolle und für den Datentransfer in unterschiedliche Zieldestinationen gibt es nur wenige Auswahlmöglichkeiten. Bei diesen stehen aber oftmals nicht alle Protokolle oder Zieldestinationen zur Verfügung und auch die Konfigurationsmöglichkeiten sind beschränkt. Aufgrund der Zunahme von selbst entwickelten Applikationen, die immer öfter in Online-Diensten betrieben werden, wird eine generelle Lösung dieser Probleme forciert. Um diese Probleme lösen zu können, soll deshalb ein eigenes System für die Datenerfassung und Speicherung entwickelt werden. Somit kann ein flexibles System aufgebaut werden, welches zugleich standardisiert ist, um in verschiedenen Anwendungen zum Einsatz zu kommen.

1.2 Ziele

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung eines Systems für die Erfassung und Speicherung von Daten in der Automatisierungstechnik. Obwohl das Betriebssystem *Windows* von

Microsoft in der Automatisierungstechnik eine Art Monopolstellung genießt, gewinnen *Linux*-Betriebssysteme in letzter Zeit an Relevanz. Deshalb soll das zu entwickelnde System auf dem Betriebssystemen *Microsoft* und *Debian* betrieben werden können. Das System soll modular aufgebaut werden, um Wartung und Entwicklung zu erleichtert. Das System soll in der Lage sein mit verschiedenen Protokollen Daten zu erfassen und diese in unterschiedlichen Online-Datenbanken abzuspeichern. Welche Protokolle und Online-Datenbanken verwendet werden, wird im Zuge dieser Arbeit erarbeitet. Eine weitere Anforderung an das System ist die Erweiterbarkeit. Wie in der Motivation beschrieben, bieten viele Werkzeuge dieser Art nicht alle Protokolle oder Online-Datenbanken an und es gibt keine Möglichkeiten diese zu erweitern. Deshalb muss das System so aufgebaut werden, dass zu einem späteren Zeitpunkt zusätzliche Protokolle oder Datenbanken hinzugefügt werden können. In der Automatisierungstechnik müssen Daten oft von mehreren Geräten erfasst werden, die räumlich weit voneinander getrennt sind. Aus diesem Grund soll das System auch die Möglichkeit besitzen, verschiedene Module auf unterschiedlichen Geräten zu betreiben. Dadurch können Daten mit den Modulen zwischengespeichert werden, sodass nicht jeder Wert einzeln über das Netzwerk übertragen wird. Um die Akzeptanz bei den Nutzenden für die Speicherung der Daten außerhalb des eigenen Unternehmens zu erhöhen, bedarf es auch einer Auseinandersetzung mit dem Thema: *Sichere Datenübertragung*. In der Arbeit soll auf dieses Thema intensiver eingegangen werden und auch das zu entwickelnde System soll eine verschlüsselte Datenübertragung ermöglichen. In der Arbeit sollen wichtige Aspekte, welche für eine derartige Realisierung beachtet werden müssen, behandelt werden. Die Resultate dieser Untersuchungen sollen dann in die Entwicklung des Systems einfließen. Nach Fertigstellung soll eine Prüfung des Systems durchgeführt werden, um alle Funktionen zu testen und um Aussagen über die Performance treffen zu können.

1.3 Gliederung der Arbeit

Die ersten Kapitel der Arbeit befassen sich mit der Theorie, die für die Realisierung des Systems notwendig ist. In Kapitel 2 werden die Industrielle Kommunikation und diverse Protokolle behandelt. Kapitel 3 befasst sich mit der sicheren Datenübertragung zwischen Applikationen und den verschiedenen kryptographische Verfahren. Um Daten abzuspeichern, braucht es eine Auseinandersetzung mit den verschiedenen Datenbankmodellen. Kapitel 4 behandelt die Relevantesten und nimmt eine Darstellung über die Verbreitung von Datenbanken und Datenbanksystemen vor. Das fünfte Kapitel widmet sich dem Cloud-Computing. Es erarbeitet die verschiedenen Arten des Cloud-Computings und ordnet die Möglichkeiten für das Cloud-Computing in der Industrie ein. In Kapitel 6 werden verschiedene Arten für die Kommunikation zwischen einzelnen Applikationen erläutert, denn durch die Modularität des Systems müssen verschiedene Teile miteinander kommunizieren. In der Softwareentwicklung ist Open-Source-Code ein fixer Bestandteil des Entwicklungsprozesses. Auch in dieser Arbeit wird auf Open-Source-Code zurückgegriffen und deshalb widmet sich Kapitel 7 Open-Source-Lizenzen.

Die eigentliche Umsetzung des Systems beginnt in Kapitel 8 mit der Definition des Gesamt-Layouts. Dabei werden alle Module, Schnittstellen und generellen Anforderungen definiert. Danach erfolgt in den Kapiteln 9, 10 und 11 die Umsetzung der einzelnen Module. Kapitel 12 schließt die Umsetzung mit einem Test, wobei alle Komponenten auf Fehler und Einsatzfähigkeit getestet werden.

2 INDUSTRIELLE KOMMUNIKATION

Ein wichtiger Teil der Automatisierungstechnik ist die Kommunikation. Mit der zunehmenden Digitalisierung und intelligenten Sensorik ist dieses Themengebiet ein wesentlicher Faktor, welcher bereits bei der Planung berücksichtigt werden soll. In der Praxis zeigt sich aber häufig, dass die Komponenten und Architekturen von Hardwaredesignern festgelegt werden, welche nur eingeschränkte Kenntnisse von diesem vielseitigen Themengebiet haben. Das lässt sich aber auch damit begründen, dass je nach Anlage und Prozess verschieden schnelle Prozessgeschwindigkeiten und damit die nötige Kommunikationslast variieren und diese fundamentalen Parameter nicht an die Hardwaredesigner kommuniziert werden, oder bei der Hardwareauslegung noch nicht feststehen. In der Industrie kommen auch verschiedene Anforderungen zu tragen, da viele Bereiche nicht miteinander verglichen werden können. Eine Gliederung der unterschiedlichen Bereiche kann aus der Automatisierungspyramide, welche in Abbildung 1 dargestellt ist, entnommen werden:[1]

Feldebene

Die Feldebene beschreibt das elektrische Equipment des Produktionsbereichs, wo die Wertschöpfung der Produkte erfolgt. Sie enthält Temperaturfühler, Lichtschranken und viele weitere Sensoren, die im wesentlichen physikalische Größen in elektrische Größen umwandeln. Die Gegenseite bilden Aktoren in Form von Ventilen, Schütze oder elektrischen Reglern. Die Feldebene liefert und verarbeitet dementsprechend Daten der nächsthöheren Ebene in Form von Ein- und Ausgangssignalen.

Steuerungsebene

In der Steuerungsebene werden die Signale von der Feldebene mittels *speicherprogrammierbarer Steuerungen* (SPS) eingelesen, intern verarbeitet und daraus entsprechende Ausgangssignale an die Feldebene zurückgesendet. Die Steuerungsebene bildet das Gehirn der Prozesssteuerung, da der Ablauf in den SPSen hinterlegt ist. Für die dezentral gesteuerten Maschinen- und Analagenteile ist die Steuerungsebene somit entscheidend.

Prozessleitebene

Die Leitebene bietet unter anderem eine Visualisierung produktionsrelevanter Vorgänge. Dabei werden die Prozesse, prozessrelevante Größen oder sonstige wichtige Indikatoren mittels Prozessleit-, *Human Machine Interface* (HMI)- und *Supervisory Control- and Data Acquisition* (SCADA)-Systemen einfach und leicht verständlich für die AnwenderInnen zur Verfügung gestellt. Die Leitebene dient somit als Schnittstelle zwischen Mensch und Maschine, die Informationen grafisch darstellt und mit der die Prozesse von den AnwenderInnen gesteuert werden können.

Betriebsleitebene

In der Betriebsleitebene wird auf Basies der Betriebs- Maschinen- und Personaldatenerfassung mit dem *Manufacturing Execution System* (MES) die Steuerung, Lenkung und Kontrolle der Produktion realisiert. Sie dient auch als Bindeglied zwischen Maschinensteuerung und Unternehmensebene. Das MES ist

[1] Vgl. Siepmann (2016), S. 49 - 51.

dementsprechend für die Produktionsfeinplanung sowie -datenerfassung der untergeordneten Ebenen zuständig.

Unternehmensebene

Die Unternehmensebene dient zur Produktionsgrobplanung und Bestellabwicklung der industriellen Fertigung. Dabei verwaltet das *Enterprise Resource Planning* (ERP)-System alle relevanten Daten und gibt Anforderungen und Planungen an die darunterliegenden Ebenen weiter.

Wie aus den beschriebenen Ebenen hervorgeht, ist die Kommunikation ein wesentlicher Bestandteil der Automatisierungstechnik. Daten müssen zwischen den Ebenen, aber auch in den Eben ausgetauscht werden. Ein Nachteil ist, dass sich in den unteren Ebenen noch kein Standard durchgesetzt hat und es somit verschiedene Geräte mit unterschiedlichen Protokollen gibt. Soll zu einem späteren Zeitpunkt eine Kommunikation zwischen solchen Geräten implementiert werden, ist das mit einem hohen Aufwand und hohen Kosten verbunden, da oft Schnittstellenkarten nachgerüstet werden müssen. Dass jedes Gerät alle Protokolle abhandeln kann, ist eine Utopie, da für verschiedene Ebenen unterschiedliche Anforderungen benötigt werden.

Auf die Betrachtung des aktuelleren *Referenzarchitekturmodells Industrie 4.0* (RAMI) wird bewusst verzichtet, da dieses Kapitel rein die Kommunikation betrachtet. Dafür wäre das dreidimensionale RAMI zu komplex da es auch viele andere Aspekte mit einbezieht.

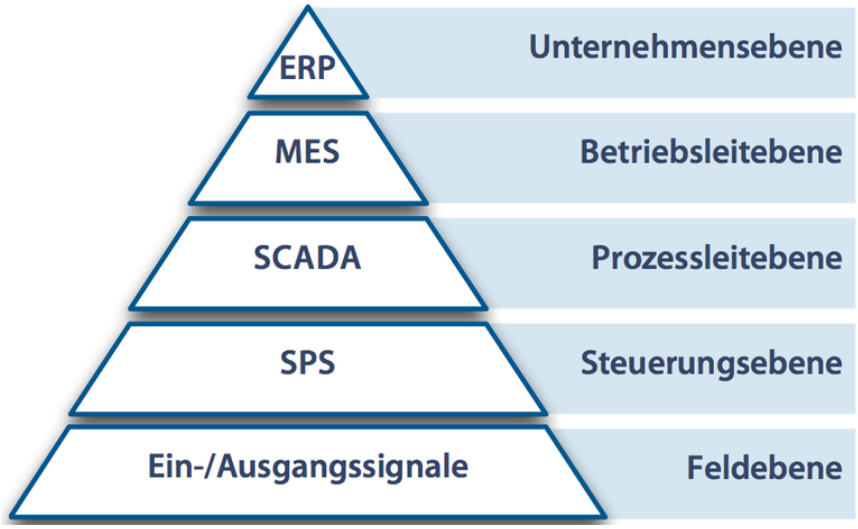

Abbildung 1: Automatisierungspyramide, Quelle: Gronau (2020), Online-Quelle [31.7.2020].

2.1 OSI-Modell

Eine wichtige Rolle bei der Kommunikation zwischen verschiedenen Systemen nehmen Protokolle ein. Um diese in kleinere Funktionseinheiten zu gliedern, wurde das *Open Systems Interconnection* (OSI) Modell entworfen, welches in diesem Kapitel näher erörtert wird:[2]

[2] Vgl. Sen (2014), S. 47 - 49.

Das OSI-Referenz-Modell dient als Basis bei der Entwicklung von *System-Vernetzung-Standards*. Es beinhaltet alle Aspekte der Netzwerkkommunikation und wird von der *International Standards Organization* (ISO) veröffentlicht. Das OSI-Modell ist kein Protokoll oder eine Definition von Regeln, sondern ein Framework zur Definition und Gliederung von Protokollen. Dabei wurde eine Schichtenarchitektur entwickelt (Abbildung 2), welche die Entwicklung von Netzwerksystemen erleichtert. Diese Architektur kann für alle Kommunikationsarten zwischen allen Computern eingesetzt werden.

Jede Schicht (sog. Layer) hat seine eigene Aufgabe und es gibt einen hierarchischen Aufbau:

- Layer 7: Anwendungsschicht — Prozesse zu den Applikationen
- Layer 6: Darstellungsschicht — Daten Repräsentation
- Layer 5: Sitzungsschicht — Interhost Kommunikation
- Layer 4: Transportschicht — Ende-zu-Ende Verbindung
- Layer 3: Vermittlungsschicht — Adressierung und Routing
- Layer 2: Sicherungsschicht — Medien Zugang
- Layer 1: Bitübertragungsschicht — Binäre Übertragung

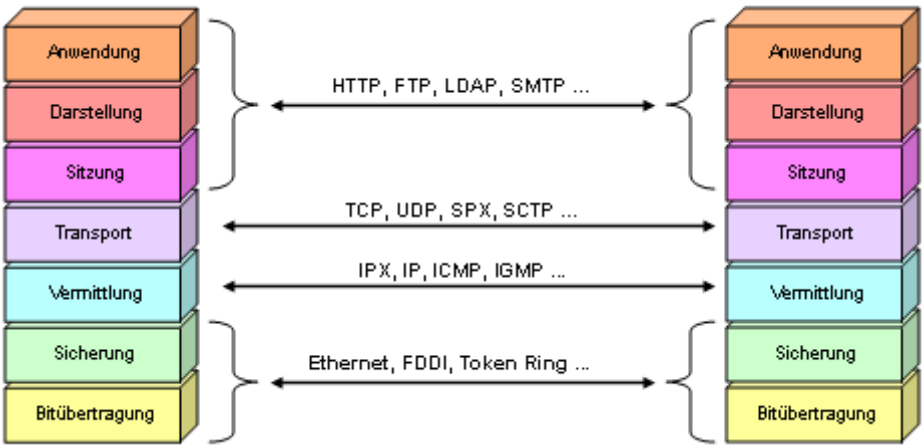

Abbildung 2: OSI-Referenz-Modell, Quelle: Aunkofer (2010), Online-Quelle [17.08.2020].

Findet die Entwicklung von Protokollen aufgrund dieses Schichtenmodells statt, können die Schichten unabhängig voneinander entwickelt werden. Denn jede Schicht hat ihre vorgegebene Funktion und ist somit ein eigenes Paket. Werden die Schichten exakt eingehalten, wird dadurch auch eine große Flexibilität generiert. Das bedeutet, dass eine Änderung in einer Schicht keine Auswirkungen auf eine andere Schicht haben darf. Auch das Verwenden von anderen Schichten, von anderen Protokollen, ist dadurch möglich.

Für den Transport werden die Quelldateien in Pakete gekapselt und von der obersten Schicht, der Anwendungsschicht, nach unten transportiert. Dabei fügt jeder Layer Kontrollinformationen vor das Paket ein, den sogenannten Header. Kommt das Paket mit allen Headern zur Bitübertragungsschicht, wird es über ein physikalisches Übertragungsmedium übertragen. Am Zielgerät durchläuft das Datenpaket dann die umgekehrte Reihenfolge als bei dem Sender. Die einzelnen Schichten am Zielgerät und Sender kommunizieren nur mit gleichen Schichten des anderen Gerätes und sind durch Rollen und Konventionen geregelt. Durch das Schichtenmodell müssen Zwischengeräte wie Switches oder Router nicht die gesamte Nachricht verstehen, sondern können nur aufgrund der unteren Schichten die Datenpakete weiterleiten.

2.2 Feldbus

In der Automatisierung kann zwischen zwei Kommunikationstypen unterschieden werden: Dem Feldbus und der Kommunikation in den höheren Ebenen der Automatisierungspyramide. Die Unterscheidung zielt speziell auf die unterschiedlichen Anforderungen ab, welche gravierend verschieden sind. Eine genaue Definition des Feldbusses lässt sich nur schwer finden, da dieser in der Vergangenheit etliche Wandlungen durchlaufen hat. In der IEC 61158-2 ist eine sehr allgemeine Definition angegeben: “A fieldbus is a digital, serial, multidrop, data bus for communication with industrial control and instrumentation devices such as - but not limited to - transducers, actuators and local controllers.”[3]

Nur wenige Einführungen in der Automatisierungstechnik hatten so große Effekte wie der Feldbus. Bevor dieser Einzug nahm, herrschten sternförmige Punkt-zu-Punkt Verbindungen vor, welche von den Steuerungen zu den Aktoren oder Sensoren ausgingen. Ziel des Feldbusses ist es, dezentrale Ein- und Ausgabegeräte miteinander zu verbinden und so wirklich verteilte Systeme zu erschaffen. Dies hat den Vorteil, dass nicht jedes digitale oder analoge Signal zur Steuerung verdrahtet werden muss, sondern zu verteilten Ein- und Ausgabegeräten, welche die Daten von und zur Steuerung über das Datenkabel liefert. Ein weiterer Punkt war auch die Intention, die 4 - 20 mA Verkabelung zu vermeiden, da mit Bussystemen nicht nur Ist-Daten empfangen werden können, sondern auch Diagnosedaten. Aber auch in die Gegenrichtung können Daten gesendet werden, beispielsweise für die Parametrierung. Heute werden Feldbusse in fast allen Bereichen eingesetzt: Prozesssteuerung, Fabrik-, Gebäude- und Hausautomatisierung, Automobil- und Eisenbahntechnik oder in der Avionik.[4]

Wie eingangs erwähnt, gelten für den Feldbus andere Anforderungen als für die bekannte IT-Kommunikation. Nachfolgend sind die wichtigsten Aspekte laut Schnell und Wimann angeführt:[5]

Topologie

Für den Einsatz in der Automatisierungstechnik ist es nötig, dass der Feldbus die Automatisierungsgeräte über geringe Distanzen von wenigen Metern, aber auch über einige Kilometer verbinden kann. Des Weiteren muss eine flächendeckende Verlegung möglich sein, um die in der Prozessumgebung verteilten Geräte miteinander zu verbinden. Dabei können verschiedene Topologien, wie Stern, Linie oder gemischte zum Einsatz kommen. Um logische Abschnitte „quasi“ autonom arbeiten zu lassen wird Flexibilität der Teilnehmer gefordert.

Zeitverhalten

In der Automatisierungstechnik ist bei der Abarbeitung sämtlicher Funktionen und Kommunikation Echtzeitverhalten gefordert. Echtzeit bedeutet, dass die Abläufe deterministisch und innerhalb definierter Grenzen erfolgen müssen. Diese Grenzen können von Prozess zu Prozess unterschiedlich sein, bewegen sich zwischen Millisekunden bis hin zu einigen Sekunden, und werden deshalb bei der Realisierung oder Planung definiert. Für die Absicherung der echtzeitfähigen Kommunikation werden verschiedene

[3] IEC 61158-2 (2003), S. 15.

[4] Vgl. Sauter (2007), S. 13-1 f.

[5] Vgl. Schnell/Wimann (2019), S. 138 f.

Zugriffsverfahren verwendet, zum Beispiel „Polling“ wegen der äquidistanten Abtastung. Des Weiteren sollten kurze Datentelegramme verwendet werden, welche einen hohen Nutzdatenanteil beinhalten.

Zuverlässigkeit

Da alle Teilnehmer bei einem Feldbus denselben seriellen Übertragungskanal nutzen, resultiert unweigerlich eine strukturelle Verschlechterung der Zuverlässigkeit. Bei herkömmlichen Übertragungsstrukturen hatte jeder Teilnehmer einen eigenen Übertragungskanal, die nun alle zusammengefasst sind. Somit müssen Maßnahmen ergriffen werden, um die Ausfallwahrscheinlichkeit positiv zu beeinflussen, damit die Zuverlässigkeit wieder akzeptable Werte annimmt. Dabei werden unter anderem auftretende Fehler lokalisiert, um fehlerhafte Geräte von der Kommunikation auszuschließen. Ein großer Vorteil der Feldbusse ist auch die höhere *Elektromagnetische Verträglichkeit* (EMV) gegenüber herkömmlichen IT-Kommunikationsarten. Des Weiteren werden durch Codesicherungsverfahren die übertragenen Daten vor nicht erkennbaren Fehlern, die beispielsweise durch elektromagnetische Einflüsse auftreten können, weitestgehend überwacht.

Stromversorgung

Für eine korrekte Arbeitsweise in Industrieanlagen ist eine zuverlässige Gewährleistung der Hilfsenergie unumgänglich. Deshalb werden bei allen Anlagen individuelle Lösungen konzipiert, wodurch auch redundante oder fehlertolerante Stromversorgungen zum Einsatz kommen.

Einsatzbedingungen

Feldbusse werden in rauen Umgebungen eingesetzt, weshalb bei der Wahl des Feldbusses auf diese Rücksicht genommen werden muss. Zu den Einsatzbedingungen zählen: großer Temperatur- und Feuchteberreich, mechanische Schwingungen, schädliche Partikel in der Umgebungsluft oder Explosions-Schutz.

Flexibilität

Bei Feldbussystemen wird ein gewisser Grad an Flexibilität erwartet. So sollen zusätzliche Teilnehmer später einfach eingebunden werden können und Werkzeuge zur Inbetriebnahme diese erleichtern.

Wirtschaftlichkeit

In der Geschäftswelt stellen die Kosten einen wichtigen Punkt bei der Projektierung von Feldbussystemen dar. Die Kundinnen und Kunden sind nicht bereit mehr als 10 - 20 % zusätzlich für Bussysteme bei einem Automatisierungsgerät auszugeben. Daher werden gestaffelte Lösungen angeboten. Diese erlauben eine bedarfsgerechte Erweiterung der Basisfunktionalitäten, wie Diagnosemöglichkeiten. Es soll aber auch die Entwicklungs- und Serviceumgebung preiswert sein.

Wie zu erkennen ist, wird auf Sicherheit nicht explizit eingegangen. Grund dafür ist, dass Feldbusse meist abgetrennt sind von anderen Netzwerken und somit nur ein geringes Bedrohungspotenzial besteht. Durch die zunehmende Vernetzung der Geräte mit dem Schlagwort „Industrie 4.0“, wo nun alle Geräte miteinander kommunizieren sollen, fließen nun jedoch vermehrt Sicherheitsaspekte in die Feldbusplanung ein.

2.3 Möglichkeiten zur Datenerfassung in der Automatisierungstechnik

Wie in der Einleitung bereits erwähnt, ist ein Bestandteil dieser Arbeit das Erfassen von Daten von verschiedenen Quellen in der Automatisierungstechnik. Um dieses Ziel zu erreichen, muss zuerst evaluiert werden, wie die Daten erfasst werden können. Die Geräte in der Automatisierungstechnik haben keinen gemeinsamen Standard für eine Schnittstelle. Auch Protokolle können unterschiedlich sein, obwohl dieselbe physikalische Übertragungshardware eingesetzt wird. Das liegt zum einen daran, dass die Hardwarekomponenten meist unterschiedliche Anforderungen besitzen oder aufgrund ihres Alters neuere Schnittstellen noch nicht verfügbar waren. So finden sich beispielsweise Kontroller für diverse Steuerungsaufgaben, die nur eine RS-232 Schnittstelle besitzen. Diese Kontroller können zwar über diese Schnittstelle programmiert und ausgelesen werden, jedoch ist diese Schnittstelle nicht mehr weit verbreitet und in den allermeisten Fällen wird ein Adapter zur Kommunikation benötigt. Somit ist es für diese Geräte mit mehr Aufwand verbunden Daten zu erhalten, da meist zusätzliche Hardware benötigt wird und auch der Programmieraufwand höher ist. Ein weiteres Problem sind, wie bereits erwähnt, die unterschiedlichen Protokolle, die die verschiedenen HerstellerInnen verwenden. Zwar hat sich laut Volz, Ethernet heute schon fast bei den SPSen etabliert, was auf den Grund zurückzuführen ist, dass es Bestrebungen gibt Ethernet als Basis für Echtzeitprotokolle zu nutzen. Die Gründe dafür sind beispielsweise, dass dadurch eine einfachere Anbindung an die bestehenden IT-Systeme möglich ist, größere Adressbereiche zur Verfügung stehen oder auch größere Datenmengen übertragen werden können.[6] Somit etabliert sich zwar ein physikalischer Standard bei den SPSen, womit ein großes Gebiet in der Automatisierung abgedeckt wird, jedoch verwenden viele HerstellerInnen unterschiedliche Protokolle wie *Profinet* oder *EtherCat* und können somit nicht standardmäßig miteinander kommunizieren. Auch das Erfassen von Daten gestaltet sich schwierig, da für diese Protokolle Treiber entwickelt oder zugekauft werden müssen.

Wie vielfältig der Bereich der Protokolle ist zeigt Abbildung 3. Aus dieser Grafik geht hervor, dass seit den 2000er Jahren einige ethernet-basierte Feldbusse entstanden sind, welche hellblau eingefärbt sind. Besonderes Augenmerk sollte auch auf *Open Platform Communications Unified Architecture* (OPC UA) gelegt werden, da diese Technologie eine immer größere Verbreitung findet und in der Fachwelt oft als zukünftiger Standard für den Datenaustausch zwischen Systemen erachtet wird. Zwar wird Profinet von 75 % der Maschinenbauunternehmen in Deutschland eingesetzt, aber auch OPC UA wird bereits von jedem vierten Unternehmen verwendet und soll in Zukunft auf einem Wert von 42 % steigen.[7] Laut Definition der OPC Foundation, welche den OPC UA Standard veröffentlicht, lautet die Definition für OPC UA wie folgt:

„OPC-Unified Architecture (OPC UA) ist der Datenaustausch-Standard für eine sichere, zuverlässige, Hersteller- und Plattform-unabhängige Kommunikation. Sie ermöglicht einen Betriebssystem-übergreifenden Datenaustausch zwischen Produkten unterschiedlicher Hersteller. Der OPC UA-Standard

[6] Vgl. Volz (2010), S. 129 f.

[7] Vgl. Rothhöft (2020), Online-Quelle [18.09.2020], S. 23 f.

besteht aus Spezifikationen welche in enger Zusammenarbeit zwischen Herstellern, Anwendern, Forschungsinstituten und Konsortien entstanden sind, um den sicheren Informationsaustausch in heterogenen Systemen zu ermöglichen."[8]

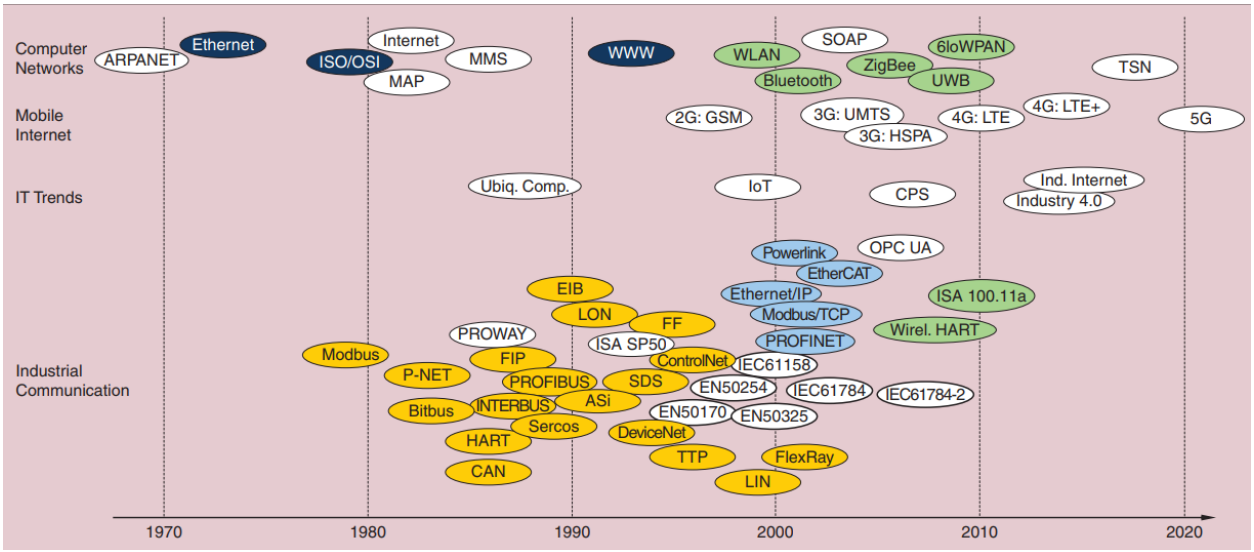

Abbildung 3: Meilensteine in der Evolution der Industriellen Kommunikation, Quelle: Wollschlager/Sauter/Jasperneite (2017), S. 19.

Als Resümee dieses Unterkapitels geht hervor, dass es eine große Anzahl an diversen Technologien und Protokollen in der Automatisierungstechnik gibt. So gibt es Kabel- und Funk-basierte Technologien die sich aber auch in ihrer Ausführung unterscheiden können. Im Zuge dieser Arbeit werden hauptsächlich ethernet-basierte Protokolle näher betrachtet, da diese Hardware am häufigsten verbreitet ist. Eine Auswahl von Protokollen, die für Realisierung dieses Systems in Betracht gezogen werden, sind in Kapitel 6 angeführt und werden dort intensiver behandelt.

[8] Burke (2018), Online-Quelle [19.08.2020].

3 SICHERHEIT BEI DER DATENÜBERTRAGUNG

Bei der Realisierung eines Projektes mit Datensammlung und Cloudcomputing muss schon bei der Projektierung ein wesentliches Augenmerk auf die Sicherheit der Daten gelegt werden. Die Folgen, welche entstehen, wenn Daten eines Unternehmens zu wenig oder gar nicht geschützt werden, kann des Öfteren den Medien entnommen werden. Dabei können verschiedene Szenarien eintreten wie Erpressung, Verfahren wegen Vernachlässigung des Schutzes von Daten oder auch ein Imageverlust. Dieses Themengebiet gewinnt auch immer mehr an Bedeutung, da eine immer größere Anzahl von Geräten vernetzt werden und miteinander kommunizieren. Die Daten werden auch vermehrt zur Speicherung und Verarbeitung in Clouds geschickt und verweilen dann nicht mehr in den abgeschotteten internen Netzwerken. Verschiedene Cloud-Anbieter setzen auch bereits eine Verschlüsselung von Daten voraus, da die Bedeutung von Datensicherheit und deren Relevanz von einigen Personen noch vernachlässigt wird. Aber auch KundInnen verlangen einen sicheren Umgang mit Daten, wenn diese für Diagnose- oder Aufzeichnungszwecke an den Hersteller oder an die Herstellerin gesendet werden sollen. Da die Sicherheit von Daten eine hohe Relevanz besitzt, befassen sich auch mehrere Normen damit. So handelt die IEC 62443 über die Sicherheit industrieller Kommunikationsnetze und die DIN ISO/IEC 27001 behandelt die Informationssicherheit von Organisationen.

Für die Erreichung der IT-Sicherheit müssen folgende Punkte beachtet werden:[9]

1. **Verfügbarkeit**
   Die Daten, Dienstleistungen, Funktionen, und der Gleichen müssen wie vorgesehen genutzt werden können. Die Verfügbarkeit ist ein Maß der Wahrscheinlichkeit, dass das System die Leistung erbringt, wenn diese erforderlich ist. Stehen Systeme unerwartet nicht zur Verfügung, kann ein wirtschaftlicher Schaden entstehen. Dies ist der Fall, wenn zum Beispiel ein Onlineshop durch eine Fehlfunktion keine Bestellungen annehmen kann und dadurch kein Umsatz generiert werden kann.
2. **Datenexistenz**
   Daten sind ein teures Gut und der Verlust kann erhebliche Auswirkungen auf ein Unternehmen haben. Gehen beispielsweise Kunden- und Projektdaten oder sonstige wichtige Daten verloren, kann die Existenz des Unternehmens bedroht sein. Werden solche Verluste publik, zieht das auch Vertrauenseinbußen bei Kundinnen und Kunden mit sich.
3. **Integrität**
   Unter Integrität wird die Richtigkeit der Daten verstanden. Dies bedeutet, dass die Daten unverändert und vollständig vorliegen. Bei der Verarbeitung und Übertragung von Daten werden Maßnahmen getroffen, um die Integrität festzustellen. Es kann zwar nicht verhindert werden, dass Daten verändert werden, aber Mithilfe von Prüfsummen können Veränderungen im Nachhinein festgestellt werden.
4. **Vertraulichkeit**
   Bei der Vertraulichkeit sollen die Daten gegenüber Dritten nicht zugänglich sein. Deshalb soll der

[9] Vgl. Gadatsch/Mangiapane (2017), S. 17 - 22.

Klartext nur von Berechtigten zugänglich sein. Um dies zu gewährleisten, müssen Daten bei den Übertragungen verschlüsselt werden.

5. **Verbindlichkeit**

   Bei der Verbindlichkeit geht es um die Sicherstellung, dass die ausgetauschten Daten verpflichtend sind. Diese Verbindlichkeit ist beispielsweise im E-Commerce wichtig, dass ein gültiges und rechtsfähiges Geschäft zustande kommen kann. Dabei umfasst die Verbindlichkeit folgende Aspekte:

   Nichtabstreitbarkeit:

   Dabei geht es darum, dass die Herkunft oder der Erhalt einer Nachricht nicht abgestritten werden kann.

   Authentizität:

   Die Authentizität soll sicherstellen, dass Empfänger und Sender der Nachricht echt sind. Dafür werden digitale Unterschriften oder Zertifikate verwendet.

3.1 Kryptographische Verfahren

Um die oben genannten Ziele zu erreichen, werden nun die symmetrische und asymmetrische Verschlüsselung sowie Hashfunktionen behandelt. Diese dienen unter anderem als Grundlage moderner Verfahren, wie beispielsweise *Transport Layer Security* (TLS), auf das im nächsten Abschnitt näher eingegangen wird. Die wichtigsten Verfahren der kryptographischen Verschlüsselung sind:[10]

Symmetrische Verschlüsselung

Um Nachrichten geheim zu übertragen, müssen diese verschlüsselt werden. Eine Methode ist die symmetrische Verschlüsselung. Dabei wird die Nachricht so verändert, dass nur Berechtigte die Nachrichten lesen können, aber keine Angreifer. Dafür wird bei der symmetrischen Verschlüsselung ein sogenannter „Schlüssel“ benötigt, der dem Sender und Empfänger vor Übermittlung der Daten bekannt sein muss. Mit diesem Schlüssel wird die zu übertragende Mitteilung verschlüsselt und nur mit diesem Schlüssel kann die Mitteilung wieder entschlüsselt werden. Deshalb ist es essenziell, dass der Schlüssel geheim gehalten wird. Das Funktionsprinzip der Verschlüsselung liegt in einer Funktion **f**, die aus der Nachricht **m** und dem Schlüssel **k** einen Geheimtext **c** generieren kann. Die Funktion **f** muss aber mit einer Funktion **f*** umkehrbar sein, dass aus dem Geheimtext **c** und dem Schlüssel **k** wieder die ursprüngliche Nachricht **m** rekonstruiert werden kann. Das Prinzip ist in Abbildung 4 dargestellt. Die symmetrische Verschlüsselung kann des Weiteren in Blockchiffren und Stromchiffren unterteilt werden. Bei den Blockchiffren wird die Nachricht in Blöcke fester Länge unterteilt und jeder Block wird mit dem Schlüssel verschlüsselt. Beispiele für diese Art der Verschlüsselung ist der *Data Encryption Standard* (DES) und dessen Nachfolger *Advanced Encryption Standard* (AES). Bei der Stromchiffre wird eine Nachricht nicht in Blöcke unterteilt, sondern mit einem Schlüssel, der die gleiche Länge hat wie die Nachricht. Der Nachteil dieser Methode ist, wie schon erwähnt, dass der Schlüssel immer gleich lang sein muss, wie die Nachricht

[10] Vgl. Beutelspacher/Schwenk/Wolfenstetter (2015), S. 9 - 18.

und deshalb bei jeder Nachricht generiert werden muss. Der bekannteste Vertreter dieser Methode ist das *One-Time-Pad* Verfahren.

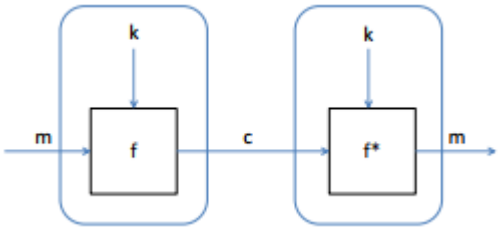

Abbildung 4: Funktionsweise symmetrische Verschlüsselung, Quelle: Beutelspacher/Schwenk/Wolfenstetter (2015), S. 10.

Asymmetrische Verschlüsselung

Ein Problem der symmetrischen Verschlüsselung ist, dass alle Beteiligten denselben Schlüssel für die Verschlüsselung und Entschlüsselung verwenden und es bis zur Entdeckung der asymmetrischen Verschlüsselung nicht möglich war, über unsichere Kommunikationskanäle, diese auszutauschen. Bei der asymmetrischen Verschlüsselung wird ein Schlüsselpaar generiert, welches sich aus einem öffentlichen Schlüssel **e** und einem privaten Schlüssel **d** zusammensetzt. Jeder Teilnehmer erhält ein eigenes Schlüsselpaar mit unterschiedlichen Schlüsseln. Das Funktionsprinzip liegt darin, dass ein Algorithmus **f** die Nachricht **m** mit dem öffentlichen Schlüssel **e** verschlüsselt und daraus die verschlüsselte Nachricht **c** resultiert. Ein weiterer Algorithmus **f*** kann dann die verschlüsselte Nachricht **c** mit dem privaten Schlüssel **d** wieder in die ursprüngliche Nachricht **m** zurückwandeln. Dabei darf es nicht möglich sein, mit dem öffentlichen Schlüssel **e** die ursprüngliche Nachricht **m** zu erhalten, oder mit dem öffentlichen Schlüssel **d** auf den privaten Schlüssel **e** rückschließen zu können. Das Prinzip ist in Abbildung 5 dargestellt. Will somit ein Teilnehmer **A** eine Nachricht an einen Teilnehmer **B** senden, holt sich **A** den öffentlichen Schlüssel **e** von **B**. Dann verschlüsselt **A** die Nachricht **m** mit dem öffentlichen Schlüssel **e** und sendet das Resultat **c** an **B**. Dieser kann dann mit dem privaten Schlüssel **d** aus dem erhaltenen **c** wieder die Nachricht **m** generieren. Der RSA-Algorithmus nach den Erfindern Rivest, Shamir und Adleman dient als Prototyp der Public-Key-Kryptographie.

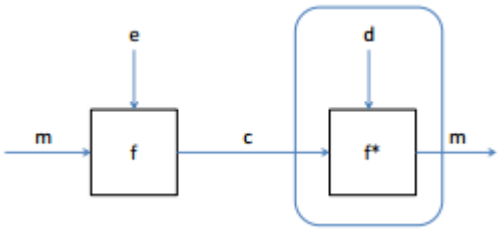

Abbildung 5: Funktionsweise asymmetrische Verschlüsselung, Quelle: Beutelspacher/Schwenk/Wolfenstetter (2015), S. 15.

Hashfunktionen

Hashfunktionen werden verwendet, um einen unmanipulierbaren Fingerabdruck einer Nachricht zu erstellen. Diese Funktionen sind kollisionsfreie Einweg-Funktionen, welche eine Nachricht beliebiger Länge auf einen Hashwert fester Länge komprimieren. Eine typische Länge einer Hashfunktion ist 160 Bit. Einweg-Funktionen können eine Transformation nur in eine Richtung vornehmen und es ist unmöglich

wieder den Ausgangswert zu erhalten. Prüfsummen eignen sich nicht zur Bildung von kryptographischen Hashfunktionen, da es möglich ist, verschiedene Nachrichten mit derselben Prüfsumme zu erzeugen. In der Praxis werden Hashfunktionen unter anderem dazu verwendet, die Integrität der übermittelten Nachricht sicherzustellen. Beispiele von Hashfunktionen sind MD5, RIPEMD, SHA-1 und SHA-2.

3.2 TLS

Ein in der Praxis häufig vorkommendes Protokoll zur sicheren Datenübertragung ist das TLS-Verfahren, der direkte Nachfolger von *Secure Socket Layer* (SSL). Es wird beispielsweise verwendet, um den Inhalt von Webseiten verschlüsselt zu übertragen und um die Webserver zu authentifizieren. Dieses Verfahren nimmt aber auch für die Übertragung von Daten in die Cloud eine wichtige Rolle ein, da viele Cloudanbieter für die Datenübertragung in die Cloud dieses Protokoll verlangen. Das Protokoll unterteilt sich in einen asymmetrischen Teil, wo Authentifizierung, Sitzungsparameter und ein Session-Key ausgetauscht werden und einen symmetrischen Teil, wo die eigentlichen Daten mit dem Session-Key übertragen werden. Da dieses Protokoll in weiterer Folge für die Realisierung benötigt wird, werden die einzelnen Schritte des Protokolls hier angeführt:[11]

1. Als erster Schritt sendet der Client dem Server eine „Client Hello“ Nachricht. In dieser Nachricht sind enthalten: Zeitstempel, Session-ID, unterstützte Kryptoverfahren (für Schlüsselaustausch, symmetrische Verschlüsselung, *Message Authentication Code* (MAC)-Algorithmen und *Pseudo Random Number Generator* (PRNG)-Funktionen) des Clients und Kompressionsverfahren.
2. Nach Erhalt dieser Nachricht, antwortet der Server mit einer „Server Hello“ Nachricht. In seiner Nachricht sendet der Server jeweils ein von ihm unterstütztes Kryptoverfahren wie der Client, eine Zufallszahl und die Session-ID des Clients. Gibt es bei einem Kryptoverfahren keine Überschneidungen, kann der Handshake nicht weitergeführt werden.
3. Des Weiteren schickt der Server entweder sein Zertifikat oder seinen öffentlichen Schlüssel allein. Der Server kann gegebenenfalls auch das Zertifikat des Clients anfordern, wobei hier die vom Server vertrauten Zertifizierungsstellen für die gegenseitige Authentifizierung mitgeschickt werden. Abschließend sendet der Server eine „Server Hello Done“-Nachricht.
4. Wird vom Server das Zertifikat des Clients angefordert, antwortet der Client mit seinem eigenen Zertifikat.
5. Danach überprüft der Client die vom Server erhaltenen Nachrichten: Zertifikat und Ciphers (gewählte Algorithmen) des Servers. Darauffolgend beginnt der Schlüsselaustausch mit RSA oder dem Diffie-Hellmann-Verfahren. Bei RSA wird ein Pre-Master Secret generiert und mit dem öffentlichen Schlüssel des Servers verschlüsselt und übertragen.
6. Der finale Sitzungsschlüssel wird dann von Client und Server berechnet. Dazu werden die ausgetauschten Zufallszahlen und das Pre-Master Secret mehrfach miteinander verrechnet.
7. Im Laufe einer Sitzung können die verwendeten Ciphers jederzeit durch die Nachricht „ChangeCiperSpc“ geändert werden.

[11] Vgl. Wendzel (2018), S. 177 f.

8. Ist der Handshake erfolgreich abgeschlossen, werden die Daten mithilfe des Sitzungsschlüssels symmetrisch verschlüsselt und mit dem Kommunikationspartner ausgetauscht.

In Abbildung 6 ist der Ablauf von TLS grafisch dargestellt.

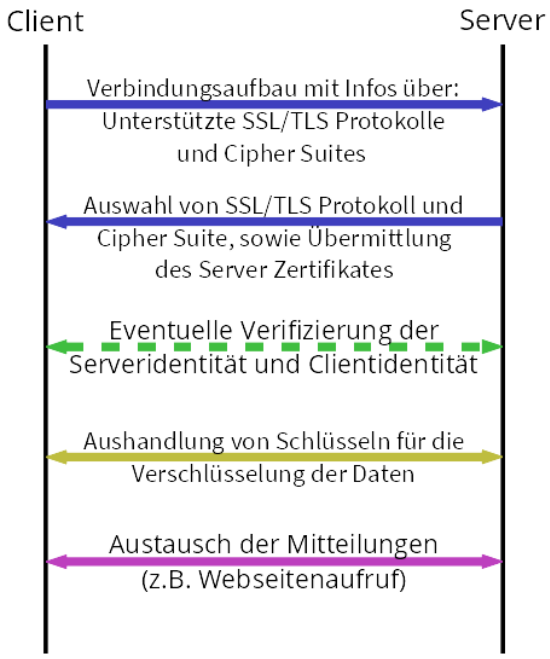

Abbildung 6: Einfache Darstellung von TLS, Quelle: Securai GmbH (o. J.), Online-Quelle [20.08.2020].

4 DATENBANKSYSTEM

Ein Datenbanksystem dient zum Speichern und Verwalten von Daten. Dabei besteht ein Datenbanksystem aus zwei Komponenten, der *Datenbank* (DB), in der die Daten abgelegt werden und dem *Datenbankmanagementsysteme* (DBMS), welches die Verwaltung und Zugriffe auf die DB verwaltet. Die DB ist nur ein File auf das das DBMS lesend und schreibend zugreift. Im DBMS sind sämtliche administrative Funktionen hinterlegt. Clients verbinden sich damit und können danach über das DBMS auf die Datenbanken zugreifen. Folgende Anforderungen bestehen an ein Datenbanksystem:[12]

- **Atomisierung**
  Daten eines Datensatzes sollen in ihre kleinsten Bestandteile zerlegt werden.
- **Datenkonsistenz**
  Eine Datenbank muss konsistent sein und darf keine Inkonsistenzen aufweisen.
- **Redundanzfreiheit**
  Informationen sollen nur einmal gespeichert werden, sodass Änderungen einer Information nicht mehrfach ausgeführt werden müssen. Das könnte zu Inkonsistenzen führen.
- **Datensicherheit**
  Die Daten in der Datenbank müssen sicher sein, deshalb werden Änderungen der Daten auch immer sofort gespeichert.
- **Datenschutz**
  Zugriff auf die Daten dürfen nur autorisierte Personen erhalten.
- **Mehrbenutzerbetrieb**
  Der Zugriff auf die Daten einer Datenbank muss mehreren Benutzerinnen und Benutzern gleichzeitig möglich sein. Dabei darf es zu keinen Inkonsistenzen kommen.

Auch wenn immer mehr Daten in Online-Datenbanken gespeichert werden, ist ein Verständnis über den grundsätzlichen Aufbau von Datenbanksystemen unabdingbar. Auch Online-Datenbanken haben dieselben Strukturen wie herkömmliche Datenbanksysteme, nur dass diese extern gehostet werden.

4.1 Datenbankmodelle

Ein Datenbanksystem hat nicht immer den gleichen Aufbau, sondern es gibt unterschiedliche Datenbankmodelle. Unter Datenbankmodell wird der grundsätzliche Aufbau eines Datenbanksystems verstanden. Dabei gibt es verschiedene Anbieter, die meist ein fixes Datenbankmodell verwenden, wobei aber auch Mischformen vorkommen. Für die richtige Wahl eines Datenbankmodells müssen die Eigenschaften, Vor- und Nachteile betrachtet werden. Da im weiteren Verlauf dieser Arbeit ein Datenbankmodell für die Speicherung der Prozessdaten ausgewählt werden muss, soll daher auf die heute relevantesten Datenbankmodelle eingegangen werden.

[12] Vgl. Bühler/Schlaich/Sinner (2019), S. 54 f.

4.1.1 Relationales Datenbankmodell

Das relationale Datenbankmodell wurde von dem englischen Mathematiker Edgar Frank Codd 1970 vorgestellt. Bei diesem Datenbankmodell gibt es nur ein Konstrukt: eine Tabelle, welche im Datenbankkontext Relation genannt wird. In solch einer Relation befinden sich die verschiedenen Datensätze, die auch eine spezielle Bezeichnung besitzen: Tupel. Bei der Definition eines Datenbankschemas, muss die Struktur eines Tupels definiert werden, denn in einer Relation müssen beim relationalen Datenbankmodell alle Tupel die gleichen Attribute haben. Für die Attribute werden vordefinierte Wertebereiche verwendet, z.B. Ganzzahlen (Integer, …), Gleitkommazahlen (Double, …), Zeichen und Zeichenketten (Char, String) oder andere definierte Datentypen. Eine Datenbank kann frei aufgebaut werden, denn es gibt keine Definitionen bezüglich Anzahl von Tabellen, Tupel oder Attributen von Tupel. Um Beziehungen zwischen den Daten zu erzeugen, werden Indizes verwendet. Dabei haben die Tupel in einer Relation meist einen eindeutigen Index, der bei einer Beziehung zu einem anderen Tupel in einer anderen Relation einfach als Attribut in dem zweiten Tupel mitgespeichert wird. Das relationale Datenbankmodell hat auch den Vorteil, dass die Relationen leicht von Menschen gelesen und verstanden werden können, denn mit der tabellarischen Ansicht von Daten ist der Mensch vertraut. Für die Datenabfrage und -speicherung wird *Structured Query Language* (SQL) verwendet. Mit dieser Datenbanksprache können die Daten schon vorsortiert ausgegeben werden.[13]

Relationale Datenbankmodelle finden vor allem dort Verwendung, wo strukturierte Daten abgelegt werden müssen, welche immer demselben Schema folgen. Bei diesen Datenbanken wird danach gestrebt, dass keine Daten doppelt abgelegt werden. Wie im oberen Abschnitt beschrieben, werden dafür Relationen verwendet. Es gibt verschiedene Relationstypen, welche beim Hinzufügen in die Datenbank eventuell überprüft werden müssen. Die Relationen stellen bei größeren Datenmengen ein Problem dar, da bei Anfragen oder Schreibbefehlen die Bedingungen überprüft werden müssen. Bei Schreibbefehlen darf, wenn ein Feld als UNIQUE Key deklariert ist, der Wert des Feldes nur einmal in der Tabelle vorkommen. Und auch bei Abfragen von verknüpften Datensätzen müssen alle eingebundenen Tabellen durchsucht werden. Deshalb werden bei größeren Datenmengen mittlerweile andere Datenbankmodelle verwendet, die in die Rubrik NotOnly SQL fallen. Dazu zählen die folgenden Datenbankmodelle. Auch die Limitierung auf elementare Datentypen ist ein Problem, weshalb bereits die meisten Anbieter von Datenbanken es ermöglicht haben, nicht atomare Daten, wie Arrays, abzuspeichern.

4.1.2 Objektrelationales Datenbankmodell

Objektrelationale Datenbanken sind eine Weiterentwicklung von relationalen Datenbanken und benutzen verschiedene Konzepte der Objektorientierung. Grund dafür ist, dass reine relationale Datenbanken strikte Restriktionen hinsichtlich der Attribute eines Tupels besitzen und nur bestimmte Datentypen verwendet werden dürfen. Mit objektrelationalen Datenbanken ist die Verwaltung beliebiger Arten von Daten möglich, wodurch reale Objekte mit unterschiedlichen Eigenschaften besser verwaltet werden können. Das objektrelationale Datenbankmodell ist eine Kombination aus relationalen und objektorientierten Konzepten,

[13] Vgl. Meier (2017), S. 15 - 17.

wobei es keine einheitliche Definition dieses Datenbankmodells gibt. Es kann auch eine bessere Konsistenz der Datenbank erreicht werden, denn durch Typisierung von Datentypen können sinnlose Vergleiche verhindert werden. Ein weiterer Vorteil von objektrelationalen Datenbanken ist die schnellere Entwicklung durch die Wiederverwendung von bereits entwickelter Software. Der Aspekt, warum dieses Datenbankmodell aber hauptsächlich eingesetzt wird, ist, wie bereits erwähnt, die Möglichkeit auch nicht atomare Daten als Attribut abzuspeichern. Beispielsweise können Attribute eines Tupels Referenzen, andere Tupel, Kollektionen, Tabellen, strukturierte Werte oder auch Large Objects enthalten.[14]

Wie aus der zitierten Quelle hervorgeht, gibt es keine exakte Definition des objektrelationalen Datenbankmodells. Deshalb werden auch oftmals objektrelationale Datenbanken mit relationalen Datenbanken zusammengeführt. Eine bekannte objektrelationale Datenbank ist PostgreSQL.

4.1.3 Dokument-orientiertes Datenbankmodell

Dokument-orientierte Datenbanksysteme sind darauf ausgelegt, um Dokument- oder halbstrukturierte Daten zu speichern, abrufen und zu verwalten. Mit immer wachsendem Datenaufkommen im Internet, aber auch in anderen Bereichen, bekommt das Speichern von schemafreien Daten einen immer höheren Stellenwert. Bei dem dokument-orientierten Datenbanksystem werden die Daten, wie der Name vermuten lässt, in Dokumenten gespeichert. Dabei referenziert ein Schlüssel jeweils auf ein Dokument und basiert somit auf einer Schlüssel-Werte Sammlung. Der Schlüssel ist in der Datenbank eindeutig. Der Wert, also das Dokument, ist kein atomarer Datentyp, sondern kann beliebige Formen annehmen, wie beispielsweise Real, Integer, andere Schlüssel oder Arrays. Die Dokumente speichern ihre Daten nicht wie die relationalen Datenbanksysteme in Tabellen, sondern ein Dokument verwendet Standardformate wie *Extensible Markup Language* (XML), YAML Ain't Markup *Language* (YAML) oder *JavaScript Object Notation* (JSON). Aber auch binäre Formate wie PDF oder Microsoft-Office-Formate sind möglich. Da die Daten in einzelnen Dokumenten schemafrei abgespeichert werden, gibt es keine Relationen zwischen den einzelnen Daten. Sind diese, wie bei dem relationalen Datenbanksystem, erwünscht, müssen diese in der Applikationsschicht implementiert werden. Große Vorteile von diesem Datenbanksystem sind die einfache Skalierung und schnellen Reaktionszeiten. So können die Lasten von diesem *Not Only SQL* (NoSQL) Datenbanksystem auf mehrere Server verteilt werden.[15]

4.1.4 Time-Series Datenbankmodell

In diesem Abschnitt soll auch auf ein spezielles Datenbankmodell eingegangen werden. Das Time-Series Datenbankmodell wurde speziell für die Speicherung und Verarbeitung von Zeitreihen entworfen. Dabei geht es um Daten, die immer einen Zeitpunkt zugeordnet werden müssen. Mit Aufkommen von *Internet of Things* (IoT), wo Sensoren, Messgeräte, Computer usw. zyklisch oder azyklisch Daten bezogen auf die Zeit liefern, werden Datenbanksysteme, die für Zeitreihen optimiert werden, immer wichtiger. Aber auch in der Finanzbranche gibt es einen großen Bedarf für das Speichern und Verarbeiten von Zeitreihen, denn

[14] Vgl. Türker/Saake (2005), S. 11 - 14.

[15] Vgl. Kumar/Srividya/Mohanavalli (2017), S. 1 f.

Aktienwerte, Exchange-Traded Funds (ETF) oder Wechselkurse sind auch ein gutes Beispiel für Zeitreihen. Natürlich können diese Daten auch in anderen Datenbankmodellen gespeichert werden, jedoch sind die Time-Series Datenbanksysteme bei der Performance bezüglich Zeitauswertungen besser und haben auch spezielle Methoden, um Zeitreihenfunktionen durchzuführen. Um einen Einblick über Time-Series Datenbanken zu gewähren, wird in Anschluss kurz das Konzept der InfluxDB V2.0 vorgestellt, wobei sich diese Version noch in der Beta Phase befindet. Sie ist, wie in Abbildung 7 dargestellt, die am häufigsten eingesetzte Time-Series Datenbank.

InfluxDB ist eine Time-Series Datenbank für schnelle Schreibzyklen und Leseanfragen, wobei die Datenbank als Open-Source Variante wie auch als Enterprise Bezahl-Version angeboten wird. Für das Speichern von Daten muss kein Schema vordefiniert werden und somit können neue Einträge fließend in eine bestehende Messung (Tabelle im relationalen Datenbankmodell) eingefügt werden. Ein Messpunkt besteht immer aus einem Zeitpunkt, einem oder mehreren Feldern und optionalen Tags. Der Zeitpunkt kann übergeben werden oder wird sonst von InfluxDB selbst bei Eintreffen zugewiesen. Gespeichert wird die Zeit immer als *Request for Comments 3339* (RFC) Coordinated Universal Time UTC Format und es sind auch Angaben in Nanosekunden möglich. Mit den Feldern werden die Messwerte aufgenommen. Ein Feld besteht immer aus einem Schlüssel und dem dazugehörigen Wert. Das basiert darauf, dass pro Eintrag mehrere Werte (Felder) übergeben werden können. Verfügbare Datenformate sind 64-Bit Gleitkommerzahlen, 64-Bit vorzeichenlose und vorzeichenbehaftete Integer, String und Boolean. Tags können als nähere Beschreibung oder zur Unterscheidung einzelner Einträge eingesetzt werden. Sie sind auch als Schlüssel-Wert-Paar ausgeführt, wobei die Werte immer als String abgespeichert werden. Das ist ein Unterschied zu den Feldern und ein weiterer ist, dass die Feld-Werte nicht sortiert abgelegt werden. Daher benötigt eine Abfrage, die auf Feld-Werte filtert (z.B. Ausgabe von Werten zwischen x und y), eine längere Zeit, da alle Einträge durchsucht werden müssen. Des Weiteren muss eine Messung immer einer Aufbewahrungsrichtlinie zugewiesen werden. In dieser wird definiert, ob Daten nach einer gewissen Zeit automatisch gelöst werden und wie lange Daten im Cache bleiben sollen. Sollen die Daten aus dem Cache entfernt werden, werden sie komprimiert und, wie auch im Cache, sortiert in *Time-Structured-Merge-Tree-Dateien* auf der Festplatte abgelegt. Bei der endgültigen Löschung von Daten, werden dann einfach die Dateien gelöscht, wohingegen die Löschung bei relationalen Datenbanken oft ein größeres Problem darstellt. Die Speicher-Engine hat eine ähnliche Funktionsweise, wie ein *Log-Structured-Merge-Tree*, wobei es vier Komprimierungsstufen gibt. Um im Falle eines Systemabsturzes den Cache nicht zu verlieren, ist der Cache eigentlich nur eine Kopie einer *Write-Ahead-Log-Datei*. Diese dient nur zur Sicherheit und updatet den Cache, denn alle Anfragen werden vom Cache behandelt.[16]

[16] Vgl. InfluxDB (o. J.), Online-Quelle [07.09.2020].

4.2 Verbreitung der Datenbanken und Datenbanksysteme

In Tabelle 1 ist ein Ranking ausgewählter Datenbanken angeführt. Das Ranking wurde im September 2020 von *solid IT* durchgeführt und basiert hauptsächlich auf Internetdaten, wie Suchanfragen auf Google, oder Relevanz in sozialen Medien. Hierbei ist zu sehen, dass relationale Datenbanken die ersten Plätze einnehmen und andere Datenbankmodelle erst später Folgen. Es ist aber zu erwähnen, dass relationale- und objektrelationale Datenbanken nicht separat geführt werden und als relational bezeichnet werden. Es haben alle relationalen Datenbanken ein Multi-Modell, was bedeutet, dass sie nicht nur ein reines relationales Datenbankmodell besitzen, sondern auch andere Datenbankmodelle integriert haben.

Rang	Name	Datenbankmodell	
1	Oracle	Relational	Multi-Modell
2	MySQL	Relational	Multi-Modell
3	Microsoft SQL Server	Relational	Multi-Modell
4	Postgres	Relational	Multi-Modell
5	MongoDB	Document	Multi-Modell
	…		
16	Amazon DynamoDB	Multi-Modell	
17	Microsoft Azure	Relational	Multi-Modell
	…		
21	Neo4j	Graph	
	...		
25	Microsoft Azure Cosmos DB	Multi-Modell	
	…		
29	InfluxDB	Time-Series	
51	Kdb+	Time-Series	

Tabelle 1: Rangliste ausgewählter Datenbanken, Quelle: solid IT (2020), Online-Quelle [07.09.2020] (leicht modifiziert).

Abbildung 7 zeigt die prozentuelle Verteilung der einzelnen Datenbankmodelle. Auch bei dieser Darstellung wird nicht zwischen relationalen- und objektrelationalen Datenbankmodell unterschieden. In dieser Darstellung ist auch die massive Dominanz der relationalen Datenbanken ersichtlich. Aus den Daten der letzten 24 Monate ist aber auch ersichtlich, dass die Time-Series Datenbanken den größten Zuwachs (bei jedoch kleinem Startwert) erhält, wenngleich die relationalen Datenbanken auf einem gleichbleibenden Wert stagnieren.[17]

[17] Vgl. solid IT (2020), Online-Quelle [07.09.2020].

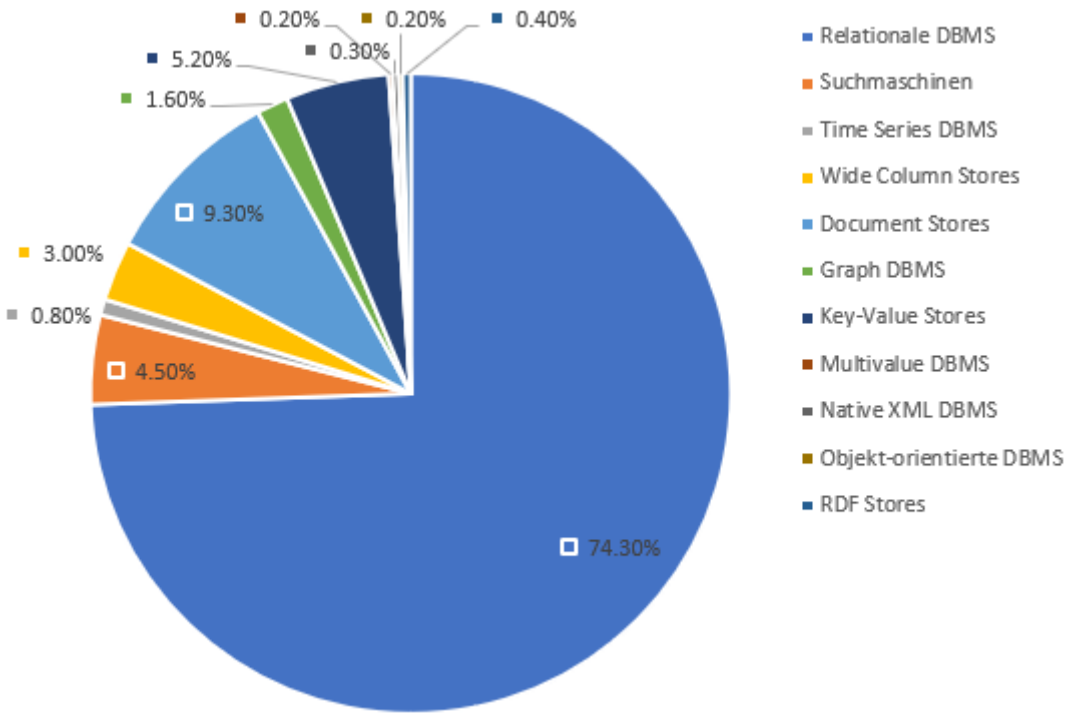

Abbildung 7: Prozentualer Anteil verschiedener Datenbankmodelle, Quelle: solid IT (2020), Online-Quelle [07.09.2020] (leicht modifiziert).

5 CLOUD-COMPUTING

In diesem Kapitel wird auf das Cloud-Computing eingegangen. In den Medien wird dieses Thema immer präsenter, jedoch können sich viele Menschen nicht so richtig vorstellen, was damit gemeint ist. Bei näherer Betrachtung wird schnell klar, dass dies ein sehr umfassendes Themengebiet ist. Im Allgemeinen geht es beim Cloud-Computing darum, dass Kundinnen und Kunden Computer, Programme oder Dienste nicht mehr auf der Hardware vor Ort betreiben, sondern in großen Rechenzentren der Cloudanbieter. Die Vorteile für die Interessenten sind dabei, dass sie sich nicht mehr um die Hardware kümmern müssen und auch eine einfache Skalierung möglich ist, so das Versprechen der Cloudanbieter. Ob diese Technologie nur Vorteile bringt und auch in der produktionellen Industrie einsetzbar ist, soll in diesem Kapitel behandelt werden. Als Cloudanbieter werden die in der Öffentlichkeit bekanntesten Anbieter herangezogen: Microsoft Azure, Amazon Web Services und Google Cloud.

5.1 Arten

Es gibt verschiedene Arten, wie eine Cloud bereitgestellt wird. Insgesamt wird zwischen vier Methoden unterschieden:[18]

- **Private Cloud**

  Bei dieser Variante wird die Infrastruktur von der eigenen Firma betrieben, befindet sich innerhalb der eigenen Organisation und wird selbst oder von einem Drittanbieter gewartet. Es gibt mehrere Gründe, um eine Cloud in der eigenen Organisation zu betreiben. Zum einen besitzen viele Organisationen bereits eigene Infrastruktur und deshalb bauen viele auf diese Ressourcen auf und versuchen die Auslastung zu optimieren und zu maximieren. Ein weiterer wesentlicher Punkt sind Sicherheitsaspekte. Vielen ist die Datenübertragung kritischer und geheimer Dokumente über das Internet und das Speichern bei einem Drittanbieter zu unsicher. Auch die Kosten für den Transfer sämtlicher Daten über das Internet sind bei übermäßig großen Datenmengen zu bedenken. Es fallen auch Kosten für den Ausbau der eigenen Infrastruktur an, wenn diese Datenmengen übertragen werden sollen. Bei geschäftskritischer Infrastruktur wollen manche Organisationen die volle Kontrolle über die Hardware behalten und hinter der eigenen Firewall platzieren.

- **Community Cloud**

  Es schließen sich mehrere Organisationen zusammen und bauen zusammen eine Cloud-Infrastruktur auf. Die Ressourcen der Cloud werden geteilt sowie Richtlinien, Anforderungen, Werte und Anliegen. Bei dieser Variante der Cloud-Gemeinschaft erhalten die Teilnehmer ein gewisses Maß an Skalierbarkeit und demokratischer Ausgewogenheit.

- **Public Cloud**

  Dieses Modell der Bereitstellung wird in der Praxis am häufigsten angewendet. Verschiedene Dienste des Cloud-Computings werden öffentlich von einem Cloud-Anbieter angeboten und können von

[18] Vgl. Dillon/Wu/Chang (2010), S. 28.

Kundinnen und Kunden entgeltlich genutzt werden. Der Anbieter behält sämtliches Eigentum an der öffentlichen Cloud und definiert seine eigenen Richtlinien und Werte. Die Kosten für die Nutzung liegen auch uneingeschränkt bei dem Anbieter, wobei es meist verschiedene Kostenkalkulationen und Gebührenmodelle gibt. Prominente Anbieter von Public Clouds sind Google oder Amazon.

- **Hybrid Cloud**

Die Hybrid Cloud ist eine Kombination aus zwei oder mehreren Clouds, der oben genannten Bereitstellungsmethoden. Die Clouds bilden zwar abgegrenzte Einheiten, jedoch sind sie durch standardisierte oder proprietäre Technologien miteinander verbunden. Dadurch können Daten und Anwendungen zwischen den einzelnen Clouds ausgetauscht werden. Organisationen nutzen dieses Modell, um die eigenen Ressourcen zu optimieren und bei Leistungsdefiziten schnell Ressourcen zu erhalten. Dabei können wichtige Kernaktivitäten über die eigene Cloud abgearbeitet und periphere Geschäftsfunktionen auf eine öffentliche Cloud ausgelagert werden. Das hybride Modell setzt eine gewisse Standardisierung und Interoperabilität der verwendeten Clouds voraus.

5.2 Arten von Clouddiensten

Im oberen Kapitel ging es um die verschiedenen Bereitstellungsvarianten. In diesem Kapitel sollen nun die unterschiedlichen Services angeführt werden, welche von den Clouds zur Verfügung gestellt werden:[19]

1. **Software as a Service (SaaS)**

Bei SaaS stehen nur Applikationen zur Verfügung, welche vom Provider entwickelt oder zur Verfügung gestellt werden. Es gibt eine große Anzahl von stationären und mobilen Geräten die über eine Thin-Client-Schnittstelle, wie beispielsweise einen Browser, auf Services in der Cloud zugreifen. Dabei können die NutzerInnen nur Einstellungen bezüglich des Service vornehmen, sie haben jedoch keinen Zugriff auf die darunterliegende Infrastruktur wie Netzwerk, Server, Betriebssystem oder Speicher. Für Anwendungen kommen Enterprise-Services in Frage, wie Arbeitsfluss-Management, Lieferketten-Kommunikation, Desktop-Software oder Finanzsoftware, aber auch Web 2.0 Anwendungen wie Metadaten-Management, soziale Netzwerke, Blogs oder Wiki Services. SaaS eignet sich nicht für Echtzeitanwendungen oder wenn die verwendeten Daten nicht extern gehostet werden sollen. Merkmale, wann SaaS in Frage kommt, sind:

- Viele unterschiedliche Organisationen verwenden dasselbe Produkt (z.B. E-Mail)
- Die Anwendung muss einen mobilen Zugang zur Verfügung stellen.
- Es besteht nicht immer ein konstanter Bedarf der Anwendung. Wenn beispielsweise immer nur kurze Spitzen in der Auslastung auftreten (z.B. Rechnungsstellung und Gehaltsabrechnung).

2. **Platform as a Service (PaaS)**

Bei dieser Variante können von Kundinnen und Kunden programmierte oder erworbene Anwendungen gehostet werden. Voraussetzung dafür ist, dass die Programmiersprachen und Werkzeuge von dem

[19] Vgl. Marinescu (2018), S. 16 - 19.

Cloud-Hoster unterstützt wird. Die Interessenten bekommen nicht die komplette Kontrolle der Infrastruktur wie Speicher, Betriebssystem oder Netzwerk, sondern nur auf die Anwendung und eventuell über die Umgebung, auf der die Anwendung gehostet wird. Unter Konfigurationsmöglichkeiten der Umgebung fallen beispielsweise Sitzungsverhalten, Sandboxen, Geräteintegration oder Inhaltsverwaltung. PaaS unterliegt aber gewissen Einschränkungen. So ist diese Variante nicht gut für portable Software geeignet, wenn von der Cloud proprietäre Software verwendet wird. Es können auch keine Optimierungen der unterlagerten Hardware und Software vorgenommen werden, da kein Zugriff auf diese besteht. PaaS eignet sich bestens für Entwicklerinnen und Entwickler, wenn mehrere zusammenarbeiten und die Bereitstellungs- und Testdienste automatisiert ablaufen sollen.

3. Infrastructure as a Service (IaaS)

Die umfassendste Variante, bezogen auf die Konfigurationsmöglichkeiten, ist IaaS. Mit dieser Variante erhält der User Zugriff auf Speicher-, Netzwerk- und anderen grundlegenden Computerressourcen. Bei IaaS kann die Kundin oder Kunde eigene Software in der Cloud ausführen aber auch eigene Betriebssysteme betreiben. Ebenfalls kann den Interessenten auch eine eingeschränkte Kontrolle über verschiedenen Netzwerkkomponenten, z.B. Firewalls, erteilt werden. Beispiele für IaaS sind: Webserver, Computer-Hardware, Betriebssysteme, Datenspeicher, virtuelle Instanzen oder Internetzugänge. Ein Merkmal dieser Bereitstellungsvariante ist, dass Ressourcen verteilt sind und eine dynamische Skalierung angewandt werden kann. Die Hardware wird von mehreren Interessenten benutzt und nur verwendet, wenn sie tatsächlich benötigt wird. Das Bezahlmodell basiert darauf, dass nur die wirklich verwendeten Ressourcen bezahlt werden. Das bedeutet, dass z.B., wenn ein Computer nur zwei Mal im Monat für eine komplizierte Berechnung benötigt wird, muss nur die tatsächliche Laufzeit bezahlt werden. Dieses Modell hat erhebliche Vorteile, wenn der Bedarf an Ressourcen volatil ist und sich eine Organisation keine eigene Infrastruktur zulegen möchte. Auch die einfache Skalierung auf mehr Speicher oder Rechenleistung ist für schnell expandierende Organisationen ein sehr ansprechender Punkt.

4. Database as a Service (DBaaS)

DBaaS ist ein Cloud-Dienst, bei dem die Datenbank direkt auf der physikalischen Infrastruktur des Providers läuft. Diese Bereitstellungsvariante könnte auch PaaS zugeordnet werden, jedoch wurde durch die häufigen Einsätze ein eigener Begriff definiert. Im Gegensatz zur eigenen Hardware mit einer Datenbank bietet diese Methode einige Vorteile:

- Sofortige Skalierbarkeit
- Garantierte Leistung
- Immer die neuesten Technologien
- Sinkende Preise
- Failover-Funktionalität (schnelles Umschalten, wenn ein Gerät ausfallen sollte)

Einige der markantesten Merkmale dieser Bereitstellungsmethode sind:

- Der Service kann einfach und ohne große Konfigurationen selbst erstellt werden.

- Die Datenbanken sind Geräte- und standortunabhängig und können ohne Rücksicht auf die Hardware betrieben werden.
- Die verwendete Hardware kann einfach skaliert werden, bezogen auf die Auslastung.
- Es wird nur bezahlt, was auch wirklich an Ressourcen verwendet wird.
- Durch die Agilität passen sich die Anwendungen nahtlos an neue Technologien an.

DBaaS wird in einer Schichtenarchitektur betrieben, wobei drei Schichten definiert sind: Benutzerschnittstellenschicht, Anwendungsschicht und Datenbankschicht. Die Benutzerschnittstellenschicht regelt den Zugriff auf den Dienst über das Internet. Den Zugriff auf den Speicher und Softwaredienste regelt die Anwendungsschicht. Die Datenbankschicht bietet einen zuverlässigen Datenbankservice und auch weitere Services wie Backup-Management oder Plattenüberwachung.

In Abbildung 8 sind drei Bereitstellungsmethoden dargestellt. Daraus sind auch einige Anwendungsfälle ersichtlich, wie auch die Zuständigkeit der einzelnen Methoden. IaaS bietet die meisten Konfigurationsmöglichkeiten und SaaS zielt eigentlich nur auf Endkunden ab.

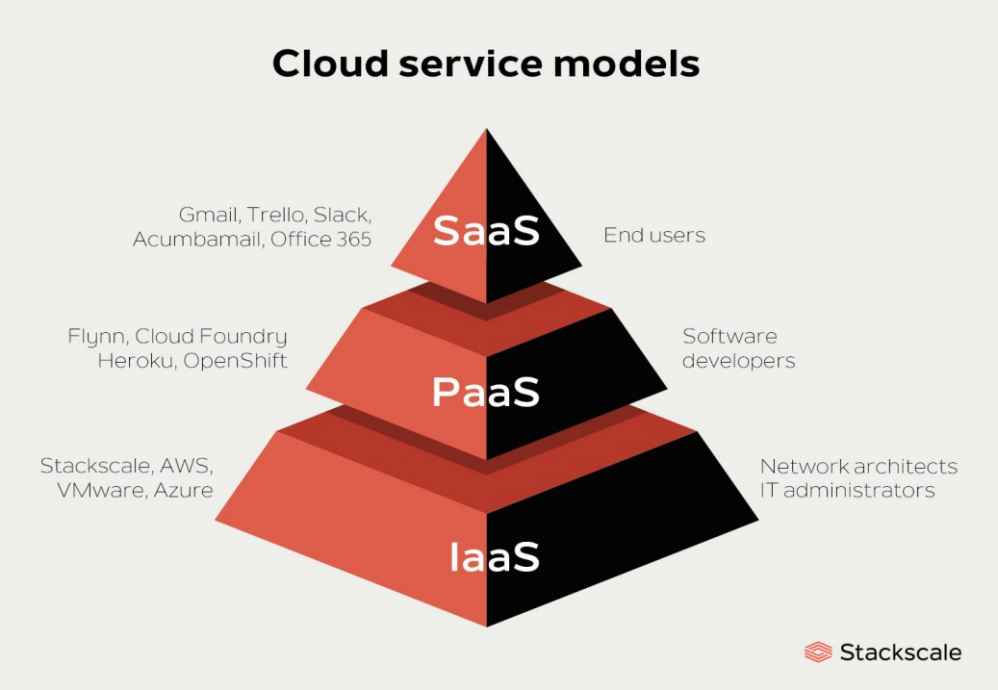

Abbildung 8: Cloud-Bereitstellungsmethoden, Quelle: Stackscale (2020), Online-Quelle [09.09.2020].

5.3 Cloud-Computing in der Industrie

Im bisherigen Kapitel sind die Arten von Cloud-Computing und die Unterscheidung zwischen den einzelnen Diensten erörtert worden. Ein Teilaspekt dieser wissenschaftlichen Arbeit ist auch die Einsetzbarkeit von Cloud-Computing in der Industrie zu erläutern, deshalb werden die Möglichkeiten in der Industrie in diesem Abschnitt behandelt. Dabei werden nicht Geschäftszweige wie Marketing oder Sales betrachtet, sondern der Bereich der Produktion und Produktionsmanagement.

Zur Erläuterung und zum besseren Verständnis über die Einsetzbarkeit von Cloud-Computing in der Industrie, wird nun eine einfache Automatisierungslösung betrachtet. Eine SPS liest und schreibt über

einen Feldbus digitale Ein- und Ausgänge. Die SPS ist über eine andere Schnittstelle mit einem Computer verbunden, auf der die Visualisierung des Prozesses abgebildet ist, mit der auch der Prozess gesteuert werden kann. Der Computer wiederum kommuniziert mit dem MES, um Informationen über einzelne Kundendaten zu erhalten. Dieser Aufbau kann in das Schema der Automatisierungspyramide überführt werden, welches bereits im Kapitel „Industrielle Kommunikation“ behandelt und in Abbildung 1 grafisch dargestellt wird. Zur genaueren Veranschaulichung sind die Komponenten in Abbildung 9 nochmals dargestellt.

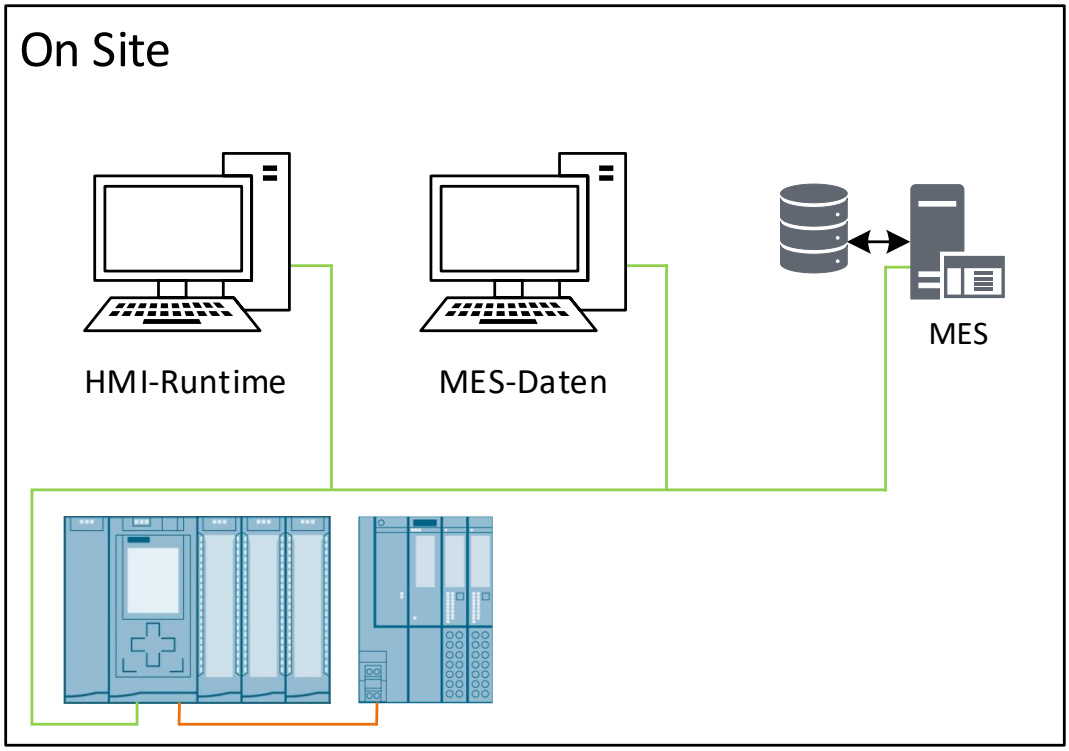

Abbildung 9: Herkömmliche Steuerungsarchitektur, Quelle: Eigene Darstellung.

Um die Einsetzbarkeit von Cloud-Computing in der Industrie zu eruieren, werden nun einige Gesichtspunkte betrachtet, die vor der Realisierung beachtet werden müssen.

Akzeptanz:

Um Cloud-Lösungen eines Drittanbieter einer Kundin oder einem Kunden zu verkaufen, bedarf es bei dieser oder diesem eine entsprechende Akzeptanz der Technologie. In der Industrie herrschen meist sehr konservative Ansichten, sodass neue Technologien nicht immer unmittelbar in der Masse eingesetzt werden. Bisherige Ansichten sind, dass Daten im Unternehmen bleiben sollen und dass, bezogen auf die Automatisierung, keine Verbindung nach draußen bestehen soll. Bezüglich der kompletten Isolation gegenüber dem Internet haben sich die Ansichten bereits etwas gelockert, denn für die Fernwartung ist es unabdingbar, dass eine Internetverbindung besteht. Begünstigt wird dieser Effekt aber auch durch Hardware für eine Fernwartung, die einen Internetzugang über ein Mobilnetz zur Verfügung stellt und so das Firmennetzwerk nicht miteinbeziehen muss. Damit kann trotz Internetzugang eine Trennung von Automatisierungsnetz und Intranet realisiert werden. Um Sicherheit der Daten in den Datencentern zu suggerieren, werden Datencenter auch in Ländern errichtet, in denen hohe Datenschutzrichtlinien

vorherrschen, z.B. Deutschland. All dies führt dazu, dass laut einer Studie von abas im Jahr 2017 nur mehr 28 % der Unternehmen Clouddienste kategorisch ausschließen, im Gegensatz zu 2015, mit 64 %.[20]

Realisierbarkeit:

Wenn ein Projekt mit Cloud-Computing umgesetzt werden soll, muss zuallererst geprüft werden, ob das Vorhaben realisierbar ist. Es wäre fatal anzunehmen, dass in der Industrie sämtliche Operationen in der Cloud realisiert werden können. Denn auch lokale Hardware vor Ort wird in nächster Zeit nicht vollkommen verdrängt werden. Zum einen können nicht Reaktionszeiten in Milli- oder Mikrosekunden-Bereich realisiert oder garantiert werden, denn die Übertragung über das Internet zu einem Rechenzentrum und wieder zurück benötigt längere Zeit und ist nicht deterministisch. Aber auch die Upload- und Downloadzeiten in unterschiedlichen Regionen können sehr variieren, sodass in bestimmten Regionen die Datenübertragung nicht garantiert werden kann. So gibt es bereits in der EU massive regionale Unterschiede bezüglich der *Next Generation Access* (NGA) Netzwerkabdeckung, wie Abbildung 10 zeigt. Ist ein Unternehmen

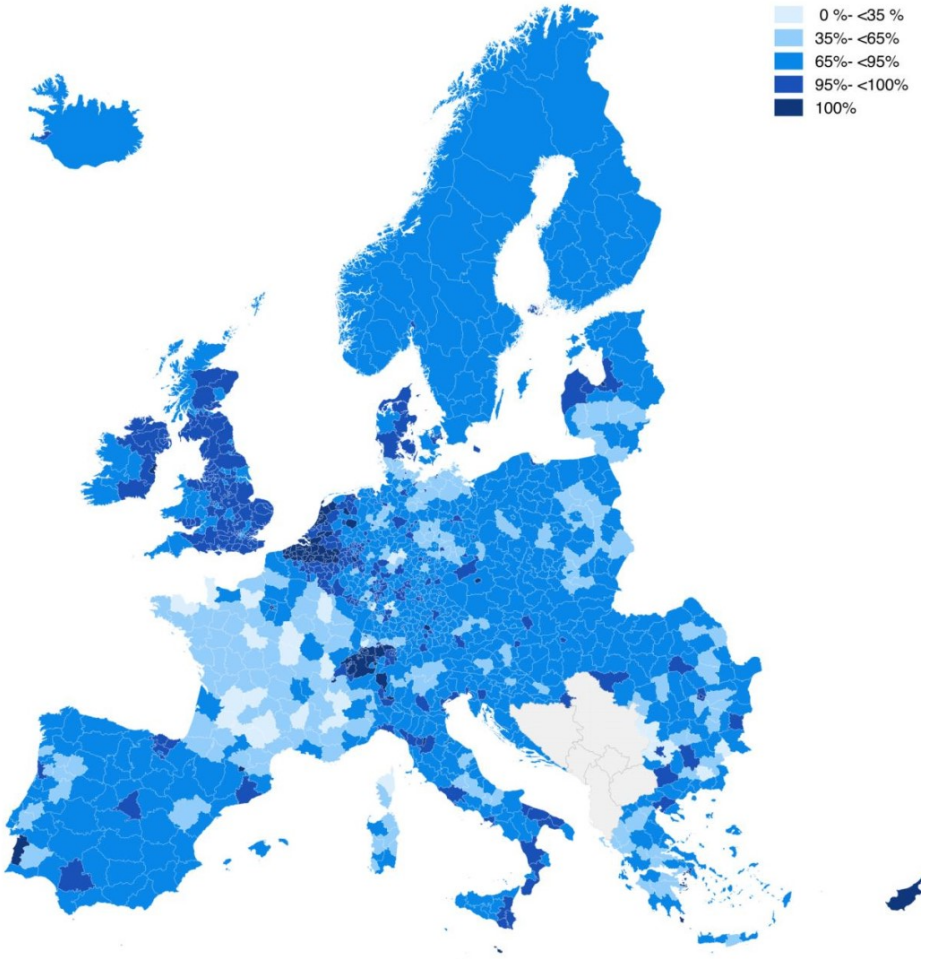

Abbildung 10: NGA Breitbandabdeckung der EU, Quelle: IHS Markit Ltd.; OMDIA; Point Topic (2020), S. 52.

international tätig und verkauft Anlagen, die eher an abgelegenen Orten errichtet werden, wie z.B. Offshore-Anlagen, Equipment für Verkehr oder Anlagen für Rohstofferzeugung, muss zuerst die Internetqualität geprüft werden. Ein weiterer Aspekt ob Clouddienste zum Einsatz kommen können, ist, ob Anlagenteile

[20] Vgl. abas Software GmbH (2017), Online-Quelle [11.09.2020].

eine Unterbrechung der Kommunikation verkraften können. Beispielsweise könnte eine Maschinensteuerung nicht komplett in die Cloud transferiert werden, denn bei Ausfall der Internetverbindung könnte es zu unsicheren Zuständen der Maschine kommen. Ist für eine Anwendung spezielle Hardware vonnöten, kann die Anwendung auch nicht als Clouddienst ausgeführt werden. Denn beim Cloud-Computing hat die Kundin oder der Kunde keinen Zugriff auf die Hardware und kann kaum Einfluss auf die eingesetzte Hardware nehmen. Bei der Kommunikation zwischen Clouddiensten und Peripherie ist auch Vorsicht geboten, denn nicht alle in der Automatisierungstechnik eingesetzten Datenbusse sind routingfähig.

Wissen:

Soll ein Projekt mit Cloud-Computing für Industrieanlagen realisiert werden, sind zusätzliche Kenntnisse vonnöten. Das trifft vor allem Netzwerkkenntnisse, denn sollen Anlagenteile dauerhaft mit dem Internet verbunden sein, müssen Netzwerkstrukturen anders aufgebaut werden und auch konfiguriert werden. Auch fundierte Kenntnisse von verschlüsselter Datenübertragung und Firewall-Einstellungen werden vorausgesetzt. Das bedeutet, dass ein viel stärkeres Augenmerk auf die IT-Infrastruktur gelegt werden muss.

Kosten:

Ein großer Aspekt bei der Realisierung ist natürlich der Kostenfaktor. Dabei muss jedoch anders kalkuliert werden als bei der Anschaffung eines Rechners. Bei Clouddiensten sind keine Anschaffungskosten fällig, denn bezahlt werden nur die konsumierten Ressourcen. Auch die Kosten für die Energieversorgung, welche bei leistungsstarker Hardware nicht zu vernachlässigen sind, sind bei den Kosten der Cloudanbieter bereits inkludiert. Wenn die Kosten für mehrere Jahre aufsummiert werden, sind Clouddienste meist preisgünstiger. In Tabelle 2 ist ein Preisvergleich zwischen den drei Cloud-Providern Amazon (AWS), Microsoft (Azure) und Google (GCP) dargestellt. Die Preise sind dargestellt für eine *Virtuelle Maschine* (VM) mit einem Linux Betriebssystem.

Type	virtuelle CPU's	Memory	AWS	Azure	GCP
General Purpose	2	8 GB	$ 0.0928	$ 0.0850	$ 0.1070
	4	16 GB	$ 0.1856	$ 0.1700	$ 0.2140
	8	32 GB	$ 0.3712	$ 0.3390	$ 0.4280
Compute Optimized	2	4 GB	$ 0.0850	$ 0.0850	$ 0.0813
	4	8 GB	$ 0.1700	$ 0.1690	$ 0.1626
	8	16 GB	$ 0.3400	$ 0.3380	$ 0.3253
Memory Optimized	2	16 GB	$ 0.1330	$ 0.1330	$ 0.1348
	4	32 GB	$ 0.2660	$ 0.2660	$ 0.2696
	8	64 GB	$ 0.5320	$ 0.5320	$ 0.5393

Tabelle 2: Preisvergleich Virtuelle Maschinen (Preise/Stunde), Quelle: ParkMyCloud (o. J.), Online-Quelle [12.09.2020] (leicht modifiziert).

Neue Möglichkeiten:

Eine Verschiebung von Funktionalität bringt nicht nur Aufwand mit sich, sondern es tun sich auch neue Möglichkeiten auf, die in herkömmlichen Systemen nur schwer realisierbar wären. So ist die einfache Skalierbarkeit von Speicher, Rechenleistung oder Anzahl von VM's ein riesiger Vorteil. Benötigt eine Anwendung beispielsweise mehr Speicherplatz, kann dies einfach mit ein paar Einstellungen realisiert werden und es wird nur ein höherer Mietpreis bezahlt. Bei eigener Hardware müsste eine neue Festplatte gekauft und eingebaut werden, was auch mit Wartezeiten und Aufwand für den Einbau verbunden ist. Die Verfügbarkeit der Daten ist auch eine Möglichkeit, einen Mehrwert für Interessenten zu generieren. Sind Daten bislang meist nur lokal verfügbar, können mit Cloudlösungen einfache Dashboards realisiert werden, die Echtzeitdaten von Anlagen darstellen. Mit dem Cloud-Computing können aber auch neue Funktionalitäten angeboten werden, welche mit konventioneller Hardware aus Kostengründen nicht verkaufbar gewesen wären. Gibt es beispielsweise Anwendungsfälle mit künstlicher Intelligenz, welche performantere Hardware nur azyklisch und für kurze Zeitintervalle benötigen, ist eine Cloudinstanz die perfekte Wahl. Es muss nicht die Hardware eines großen Servers gekauft werden, sondern es wird nur für die Zeitdauer der Problemstellung eine Cloudinstanz gestartet. Damit können die Kosten immens gesenkt senken.

Wird nun nochmals Abbildung 9 betrachtet, können nach Einbeziehung der oben genannten Gesichtspunkte für die Verwendung von Cloud-Computing in der Industrie folgende Schlüsse gezogen werden:

- Die Ein- und Ausgänge, sowohl digital als auch analog, müssen immer vor Ort bleiben, denn anders wäre die Erfassung von physikalischen Größen nicht realisierbar.
- Gewisse Steuerungsaufgaben können nicht in die Cloud ausgelagert werden, denn für unterschiedlichste Aufgaben bestehen Echtzeitanwendungen, welche mit einer Cloud nicht realisiert werden können.
- Die Visualisierung kann in die Cloud verlagert werden, denn sie dient der Adaptierung des Prozesses (Starten, Stoppen, Parameter ändern usw.) und muss deshalb keine Echtzeitanforderungen erfüllen. Die Visualisierung kann als Webserver ausgeführt sein, sodass jeder Browser den Prozess darstellen kann. Für Notsituationen müssen weiterhin vor Ort Sicherheitseinrichtungen für ein sicheres Abschalten vorhanden sein.
- Die SPS kommuniziert verschlüsselt mit dem Webserver und bekommt die Befehle und Produktdaten von diesem übermittelt.
- Ein ERP-System kann auch in die Cloud verlagert werden, wobei es dann auch möglich ist, extern darauf zugzugreifen. Das ERP-System kommuniziert dann direkt mit dem Webserver in der Cloud.
- Des Weiteren können in der Cloud weitere Funktionalitäten erstellt werden, welche vorher vielleicht nicht möglich gewesen wären. Diese Programme könnten direkt mit allen Komponenten kommunizieren.

Werden diese Aspekte in ein neues Schema eingebunden, kann dieses wie in Abbildung 11 aussehen. Dieses Schema ist nur ein Beispiel und kann nicht auf alle Anlagen angewendet werden. Für ein neues Schema einer Anlage muss immer eine gründliche Prüfung sämtlicher Aspekte stattfinden.

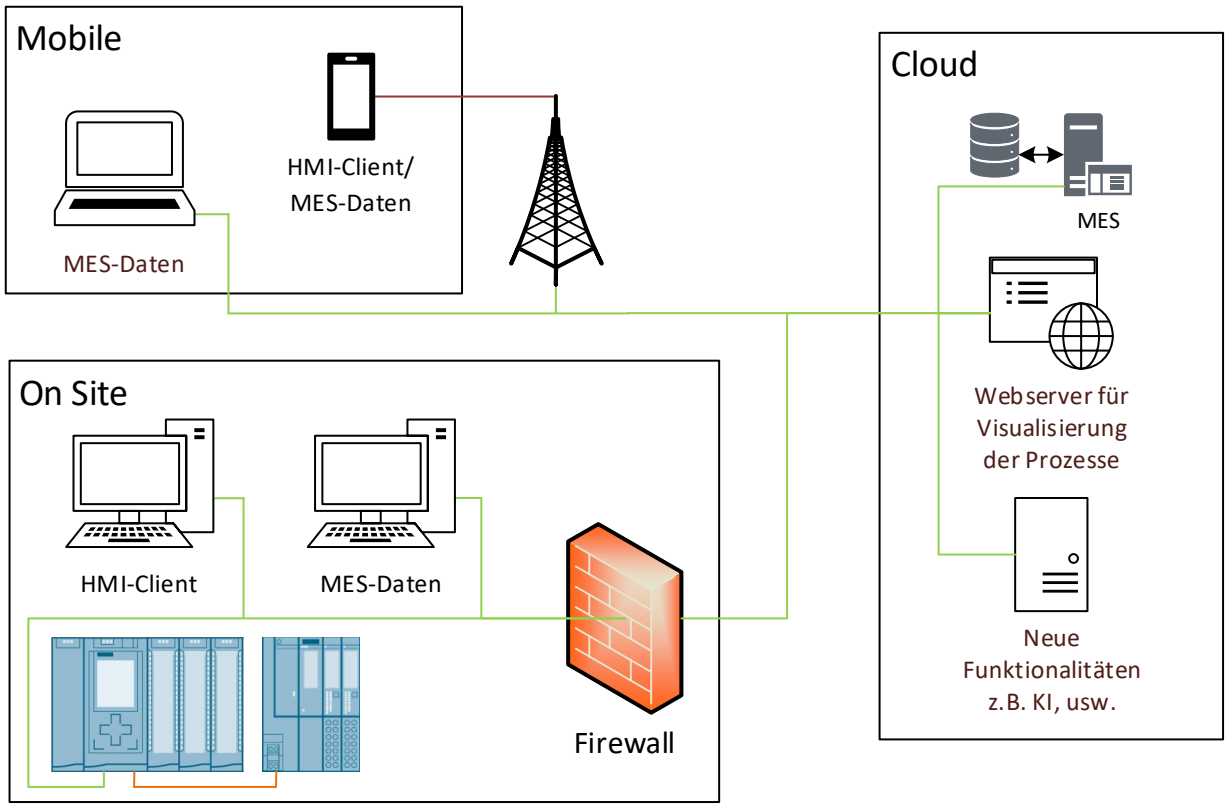

Abbildung 11: Mögliche Steuerungsarchitektur mit Cloud, Quelle: Eigene Darstellung.

6 KOMMUNIKATION ZWISCHEN ANWENDUNGEN

In diesem Kapitel werden unterschiedliche Varianten für den Datenaustausch zwischen Anwendungen erläutert. Wie in der Einleitung beschrieben, sollen die einzelnen Teile der Anwendung modular aufgebaut sein. Daher werden einige Möglichkeiten für den Datenaustausch angeführt, wobei später anhand dieser Varianten eine Auswahl getroffen wird.

Dieses Kapitel gliedert sich in zwei Unterkapitel:

- Der Kommunikation zwischen Anwendungen auf verschiedenen Systemen.
- Der Kommunikation zwischen Anwendungen auf demselben System.

6.1 Anwendungen auf verschieden Systemen

Bei der Kommunikation zwischen Anwendungen auf verschiedenen Systemen gibt es eine Vielzahl von Arten, wie *Data Distribution Service* (DDS), *Extensible Messaging and Presence Protocol* (XMPP) oder *Constrained Application Protocol* (CoAP). Im Zuge dieser Arbeit werden nur die bekannteren Arten: OPC UA, *Representational state transfer* (REST*) Application programming interface* (API) und *Message Queuing Telemetry Transport* (MQTT) betrachtet, da diese die bekanntesten sind und ihre Ansätze die meisten Problemstellungen abdecken.

6.1.1 OPC UA

Die Definition von OPC UA wird bereits im Kapitel 2.3 „Möglichkeiten zur Datenerfassung in der Automatisierungstechnik“ zitiert, hier soll nun die Funktion beschrieben werden und auf die Verwendung als Kommunikationsmittel zwischen zwei Applikationen eingegangen werden.

OPC UA wurde 2008 veröffentlicht und ist der direkte Nachfolger von *OPC Classic*. Eine grundlegende Verbesserung von OPC UA gegenüber *OPC Classic* ist die Plattform-Unabhängigkeit, denn *OPC Classic* verwendet Funktionalitäten des *Component Object Models* von Windows und ist daher nur mit diesem Betriebssystem verwendbar. OPC UA ist eine Server-Client-Architektur, bei der der Server Daten anbietet und verschiedene Clients diese lesen und beschreiben können. Es sind auch verschiedene Sicherheitsmechanismen für Verschlüsselung, Authentifizierung und Datenintegrität implementiert. Dabei gibt es für alle Mechanismen verschiedene Modi, die bei Server und Client implementiert sein können. Bei Kommunikationsstart werden diese abgeglichen und eines ausgewählt. Wird von Server und Client nicht ein gemeinsamer Sicherheitsmechanismus unterstützt, kann keine Verbindung erfolgen. OPC UA bietet auch eine Discovery Funktion, um OPC UA Server am lokalen PC oder im Netzwerk zu finden. Für das Abrufen von Daten gibt es zwei Möglichkeiten: Polling und Publish/Subscribe. Beim Polling kann ein Client auf Datenpunkte am Server zugreifen und diese nach Belieben lesen und schreiben, wobei mehrere Clients auf dieselben Datenpunkte gleichzeitig zugreifen können. Publish/Subscribe ist ein Mechanismus, welcher speziell für das Verteilen von denselben Datenpunkten auf mehrere Clients implementiert wurde. Der Server *published* (veröffentlicht) Datenpunkte und jeder Client, der einen Datenpunkt *subscribed* (abonniert) bekommt den Datenpunkt mitgeteilt. Des Weiteren können auch Methoden auf dem Server definiert werden, welche von Clients mit Parametern ausgeführt werden können. Für die Datenübertragung können mehrere Varianten verwendet werden: schnelle binäre Kommunikation oder universale

Datenformate wie JSON. Die Daten, die von einem OPC UA Server zur Verfügung gestellt werden, befinden sich im sogenannten Adressraum. Dieser ist immer hierarchisch aufgebaut, um eine einheitliche Darstellung zu erhalten und um das Auffinden von Daten zu erleichtern. OPC UA ist ein Multi-Schichtenmodell und ist so definiert, dass zukünftige Technologien und Methoden wie Sicherheitsmechanismen oder Datenübertragungsprotokolle integriert werden können.[21]

OPC UA soll als Datenaustauschstandard in der Automatisierungstechnik etabliert werden und ist bereits, wie in Kapitel 2.3 „Möglichkeiten zur Datenerfassung in der Automatisierungstechnik“ ausgeführt, in breiter Verwendung.

6.1.2 REST-API

In den folgenden drei Absätzen werden einige Basics von der REST-API erläutert:[22]

REST beschreibt eine Client-Server-Architektur, bei der Clients Daten anfragen, darstellen, verändern oder zurück an den Server übertragen. Bei dieser Architektur wird die Geschäftslogik am Server zentralisiert, welche dann von unterschiedlichen Clients genutzt werden kann. Die Übertragung basiert bei REST auf HTTP. Auf dem Server sind eindeutig identifizierbare Ressourcen definiert, welche von den Clients über HTTP-Methoden gelesen, erstellt, aktualisiert oder gelöscht werden können. Ressourcen sind serverseitige Objekte, welche sich eindeutig beschreiben lassen. Das können Kundendaten, Word-Dateien, PDF’s oder auch Bilder sein. Diese Ressourcen werden mit einem *Uniform Ressource Identifier* (URI) eindeutig identifiziert und bestehen aus einem Schema, einer Authority, einem Pfad sowie optionalen Query-Parametern. Das Schema definiert das Protokoll, also HTTP oder HTTPS für eine sichere Übertragung. Die Authority ist der Computer, worauf die API läuft. Der Pfad, zusammen mit den Query-Parametern, identifiziert das Objekt am Server.

Für die Interaktion zwischen Server und Client werden HTTP-Methoden verwendet. Diese Methoden geben an, was mit der referenzierten Ressource passiert.

- **GET**

  Diese Methode fordert eine Ressource vom Server an.

- **POST**

  Bei POST wird eine neue Ressource am Server angelegt, wobei dieser die URI selbst erzeugt. Daten der zu erzeugenden Ressourcen werden im Request-Body übertragen. Es gilt als Usus, dass die URI der erzeugten Ressource im Location-Header der Response zurückübertragen wird.

- **PUT**

  Die PUT-Methode erzeugt oder ersetzt eine Ressource. Daher muss die URI der Ressource vom Client mitgesendet werden. PUT-Requests müssen die vollständigen Daten einer Ressource enthalten und dürfen bei einem Update nicht nur die zu aktualisierenden Daten enthalten.

[21] Vgl. OPC Foundation (o. J.), Online-Quelle [16.09.2020].

[22] Vgl. Wirdemann (2019), S. 27 - 32.

- **DELETE**
  Bei DELETE wird ein Objekt, mit der vom Client an den Server übertragene URI, gelöscht.
- **PATCH**
  Mit PATCH können im Gegensatz zu PUT einzelne Daten einer Ressource upgedatet werden. Dabei wird nur die URI und die zu aktualisierenden Daten benötigt.
- **HEAD**
  HEAD liefert nur den HTTP-Header einer Ressource zurück. Damit kann ein Client prüfen, ob er die gesamte Ressource tatsächlich anfordern will. Dies findet meist bei Anwendungen mit großen Datenmengen statt.
- **OPTIONS**
  Mit OPTIOINS können die HTTP-Methoden einer Ressource abgefragt, oder *Cross-Origin Resource Sharing* (CORS)-Preflight-Requests durchgeführt werden.

Der Datenaustausch der Ressourcen passiert in einem gewissen Format, welches als Repräsentation der Ressource bezeichnet wird. Für Daten wird meist XML oder JSON verwendet. Um Daten erfolgreich auszutauschen, müssen sich Server und Client zuerst über ein Format einigen, das von beiden verstanden wird. Eine Ressource kann mehrere Repräsentationen besitzen, z.B. JSON und PDF.

Roy Fielding definierte in seiner Dissertation REST als Architekturstil für Hypermedia-Anwendungen und eine Sammlung von sechs Prinzipien, von ihm Constraints genannt:[23]

- **Client-Server**
  Anwendungen sollen als Client-Server-Architektur entworfen werden. Durch eine klare Trennung der Schnittstelle, Datenspeicherung und der Aufbereitung wird die Portabilität und Skalierbarkeit erhöht. Durch diese Architektur können die Komponenten unabhängig voneinander entwickelt werden.
- **Stateless**
  Stateless ist eine Einschränkung der Client-Server-Interaktion. Die Kommunikation muss zustandlos sein, das bedeutet, dass jede Anfrage vom Client an den Server jegliche Information enthält. Somit werden keine clientspezifischen Daten am Server gespeichert.
- **Cache**
  Der Server kann Ressourcen als cachebar markieren. Fragt der Client nun solch eine Ressource noch einmal ab, bekommt er die Ressource nur, wenn sie sich seit der letzten Abfrage geändert hat. Damit können Netzwerkressourcen erheblich geschont werden.
- **Uniform Interface**
  REST definiert eine einheitliche Schnittstellenarchitektur, wodurch die dahinterliegenden Dienste entkoppelt sind und immer mit dergleichen Syntax, Operationen, durchgeführt werden können. Diese Einheitlichkeit hat aber den Nachteil, dass nicht für jede Anwendung optimale Anfragen definiert werden können.

[23] Vgl. Fielding (2020), S. 76 - 85.

- **Layered System**
  REST-Anwendungen sind in Schichten aufgeteilt, um einzelne Funktionalitäten zu kapseln. Eine Schicht kann nicht über die eigene hinaussehen und es können beliebig viele hintereinandergeschaltet werden. Ein Nachteil besteht darin, dass die Verarbeitung von Overhead-Daten zu Latenzzeiten führen.
- **Code-on-Demand**
  Als letztes Prinzip wird Code-on-Demand angeführt. Dabei wird die Möglichkeit von Servern beschrieben, Code in Form von Applets oder Skripten den Clients zur Verfügung zu stellen und diesen dann lokal auszuführen. Dadurch kann die Zahl der vorimplementierten Funktionen eingeschränkt werden.

In Abbildung 12 ist ein einfacher Aufbau einer REST-API dargestellt. Einzelne Clients greifen über HTTP-Methoden auf die API zu. Hinter der API befindet sich eine Business-Logic, welche verschiedene Operationen durchführen kann. Die Daten, welche von den Clients abgefragt werden, liegen in einer Datenbank. In dieser Architektur greifen die Clients nicht direkt auf die Datenbank zu, sondern immer über den Server.

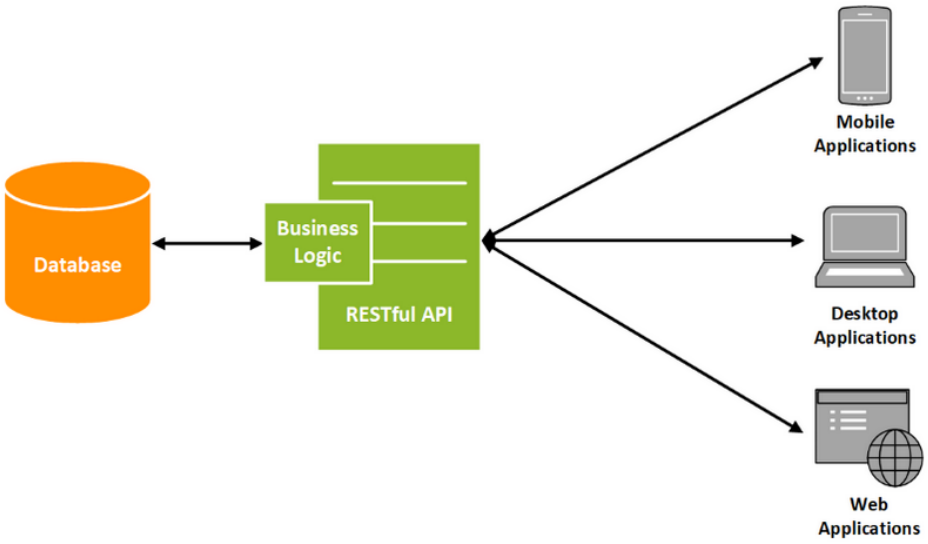

Abbildung 12: Schematische Darstellung einer REST-API: Quelle: De Roni (2019), Online-Quelle [17.09.2020].

6.1.3 MQTT

MQTT ist ein Machine-to-Machine-Protokoll und arbeitet als Publish/Subscribe-Mechanismus. Das Protokoll setzt bei der Übertragung auf TCP/IP auf und hat wenig Overhead bei der Kommunikation. Im Vergleich mit HTTP werden mehr Nutzdaten pro Nachricht gesendet. Wie bereits erwähnt, arbeitet MQTT mit dem Publish/Subscribe-Mechanismus, wobei für die Funktion Clients und ein Server (Broker) verfügbar sein muss. Der Broker dient nur als Vermittler zwischen den einzelnen Clients, denn die Clients kommunizieren nur über den Broker miteinander. Das bedeutet, dass die einzelnen Clients nichts voneinander wissen. Ist ein Client mit dem Broker verbunden, kann dieser Daten senden. Dabei *published* der Client einen Datenpunkt in einem Topic, welches als Messkanal angesehen werden kann. Ein Client

kann Datenpunkte in verschiedenen Topics *publishen*. Die Datenpunkte gelangen dann über das Topic zu dem Broker. Soll nun ein Topic zu einem anderen Client gesendet werden, *subscribed* (abonniert) der Client das Topic. Dafür sendet der Client das Interesse über ein Topic an den Broker. Bekommt der Broker nun über ein Topic einen Messwert, wird dieser über das Topic an alle Clients weitergeleitet, welche dieses abonniert haben. Für die Übertragung von Client zu Broker sowie Broker zu Client kann aus drei *Quality of Service* Level gewählt werden, welche angeben, wann eine Nachricht als erfolgreich übermittelt gilt. **Level 0 (At most once):** Der Nachrichtenerhalt wird nicht quittiert. Deshalb kann ein Ankommen der Nachricht nicht garantiert oder überprüft werden. **Level 1 (At least once):** Bei diesem Level wird beim Senden eine Bestätigungsbedingung hinzugefügt, sodass der Empfänger den Empfang der Nachricht quittieren muss. Ein Nachteil, der dadurch entsteht, ist, dass Nachrichten öfters gesendet werden und auch öfters beim Empfänger ankommen können. **Level 2 (Exactly once):** Level 2 stellt sicher, dass eine Nachricht genau einmal beim Empfänger ankommt. Dafür sind zwei Quittierungen nötig. Der Nachteil dieses Levels ist, dass hier der größte Overhead entsteht.[24]

Abbildung 13 zeigt exemplarisch, wie eine Temperatur von einem Sensor *published* und diese Temperatur von zwei Clients *subscribed* wird.

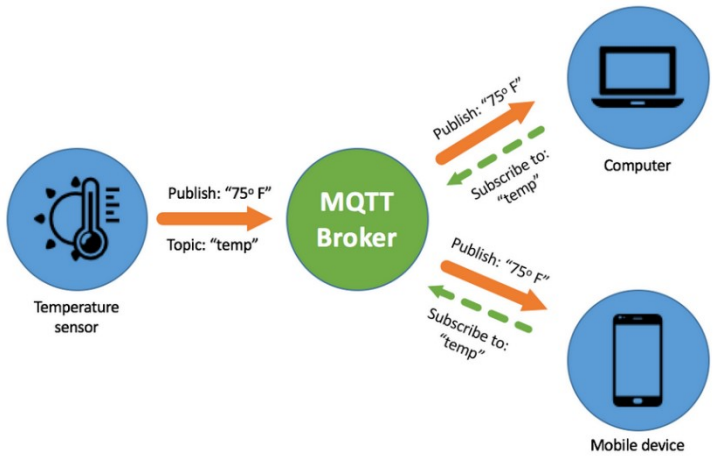

Abbildung 13: Funktionsprinzip MQTT, Quelle: Lahoz (2019), Online-Quelle [17.09.2020].

6.2 Anwendungen auf demselben System

In diesem Abschnitt werden Kommunikationsarten erläutert, welche für die Kommunikation auf demselben System verwendet werden können. Die Kommunikation zwischen Prozessen wird Interprozesskommunikation genannt. Es werden nun die verschiedenen Möglichkeiten nach Baun betrachtet:[25]

- **Gemeinsamer Speicher (Shared Memory)**
  Bei *Shared Memory* wird über einen gemeinsamen Speicher kommuniziert. Es wird ein Speicherbereich allokiert, auf den mehrere Prozesse zugreifen können. Ein Nachteil des *Shared*

[24] Vgl. Hillar (2017), S. 29 - 34, 69 f.

[25] Vgl. Baun (2017), S. 188 - 212.

Memory ist, dass die Prozesse den Zugriff auf den Speicherbereich selbst koordinieren und sicherstellen müssen, dass kein Bereich gleichzeitig beschrieben und gelesen wird.

- **Nachrichtenwarteschlangen (Message Queue)**
  Message Queues sind verkettete Listen, in denen Prozesse Nachrichten nach dem FIFO-Prinzip ablegen und auch abholen können. Im Gegensatz zu *Shared Memory*, müssen Prozesse den Zugriff auf die *Message Queue* nicht koordinieren.
- **Kommunikationskanäle (Pipes)**
  Pipes sind gepufferte Kommunikationskanäle zwischen zwei Prozessen und arbeiten auch nach dem FIFO-Prinzip. Im Gegensatz zur Message Queue legen *Pipes* keine Listen an, sondern übertragen einen gepufferten Stream.
  Pipes werden wiederum in zwei Kategorien eingeteilt:
  - Anonyme Pipes
    Diese Art von *Pipes* kann nur von eng verwandten Prozessen angewandt werden, denn die Referenz, die bei der Erzeugung entsteht, muss beiden Prozessen bekannt sein. *Anonyme Pipes* sind immer unidirektional, was bedeutet, dass nur in eine Richtung kommuniziert werden kann.
  - Bekannte Pipes
    Mit *bekannten Pipes* können auch nicht eng miteinander verwandte Prozesse miteinander kommunizieren und es ist auch eine bidirektionale Kommunikation möglich. Eine *bekannte Pipe* wird durch einen Eintrag im Dateisystem repräsentiert, sodass jeder Prozess, der den Namen der *Pipe* kennt, imstande ist, mit ihr zu interagieren.
- **Sockets**
  Mit *Sockets* können Prozesse unterschiedlicher Computer und auch Betriebssysteme miteinander kommunizieren. Jeder Prozess kann beim Betriebssystem *Sockets* anfordern und über diesen Daten senden und empfangen. Auf einem Computer wird die Adressierung der einzelnen Prozesse mittels Portnummern realisiert. Es ist eine verbindungslose- oder verbindungsorientierte Kommunikation möglich.

Diese Varianten sind im Gegensatz zu den in Kapitel 6.1 angeführten Varianten meist schneller, denn sie benötigen meist weniger Overhead. Die Sockets-Variante kann auch dazu verwendet werden, um unterschiedliche Computer miteinander zu verbinden, jedoch auch um systeminterne Verbindungen herzustellen. Dadurch ist diese Variante auch langsamer als die Übrigen.

7 LIZENZEN

Im Internet findet sich viel Source-Code, welchen EntwicklerInnen in ihren Applikationen verwenden können. Dieser Source-Code kann auf eigenen Seiten der EntwicklerInnen oder Organisationen sowie auf speziellen Websites für Versionsverwaltung, z.B. GitHub heruntergeladen werden. Es muss bei diesen Source-Codes aber beachtet werden, dass das Urheberrecht immer bei den jeweiligen Entwickelnden bleibt. Die EntwicklerInnen von Source-Codes kann aber anderen Personen oder Unternehmen ein Nutzungsrecht des Codes einräumen. Um dieses Nutzungsrecht juristisch zu regeln, wurden Lizenzen für Open-Source-Code erstellt. Damit können EntwicklerInnen oder Unternehmen Anwendungen oder Source-Codes unter eine Lizenz stellen, um die Nutzung des Codes zu regeln.

Wie in Abbildung 14 dargestellt, kann frei beziehbare Software oder Source-Code in verschiedene Kategorien eingeordnet werden. Eine Beschreibung liefert Grassmuck:[26]

- **Free Software** nimmt das Copyright und Urheberrecht der AutorInnen in Anspruch und definiert im Lizenztexten zugleich spezifische Nutzungsfreiheiten. Es gibt unterschiedliche Ausführungen, doch die wesentlichen Details, die geregelt sind, betreffen: Beifügung des Quellcodes zum Binärcode der Software, ursprüngliche Software zu ändern und diese zu verbreiten und Kopien vom Original anzufertigen und zu verbreiten.
- **Open-Source-Software d**ient meist als Synonym für Free Software.
- **Freeware** ist eine Copyright-geschützte Software, die vom Veröffentlicher kostenlos zur Verfügung gestellt wird. Diese Freeware wird oft ohne Lizenzbedingungen und ohne Quellcode veröffentlicht. Die Software kann frei weitergegeben werden.
- **Public-Domain-Software** untersteht keinem Copyright und ist somit, mit oder ohne Source-Code, gemeinfrei. Dies kann unterschiedliche Gründe haben, z.B. wenn der Autor oder die Autorin darauf verzichtet oder die Software gesetzlich nicht schützbar ist. Somit kann die Software von jedem ohne Auflagen genutzt werden, wobei auch die Beanspruchung eines Copyrights von Dritten zulässig ist.

Obwohl in der Literatur Free Software und Open-Source-Software meist als Synonym verwendet wird, definiert GNU, für die auch die erste GPL-Lizenz geschrieben wurde, Free Software geringfügig anders. Es besteht auf den unterschiedlichen Werten der beiden Typen. Laut ihrer Auffassung geht es der Free Software Bewegung für die Freiheit der BenutzerInnen von Computern, wohingegen die Open-Source-Idee mehr auf den praktischen Nutzen für die Verteilung von Software abzielt.[27]

In Abbildung 14 sind die soeben genannten Arten dargestellt. Es sind auch weitere Details dargestellt und einige Beispiele für die verwendeten Arten angeführt. Aus der Abbildung ist auch ersichtlich, dass bei Free Software als auch bei Open-Source-Software ein Entgelt für die Nutzung der Software verlangt werden kann.

[26] Vgl. Grassmuck (2004), S. 278 f.

[27] Vgl. Stallmann (2020), Online-Quelle [21.09.2020].

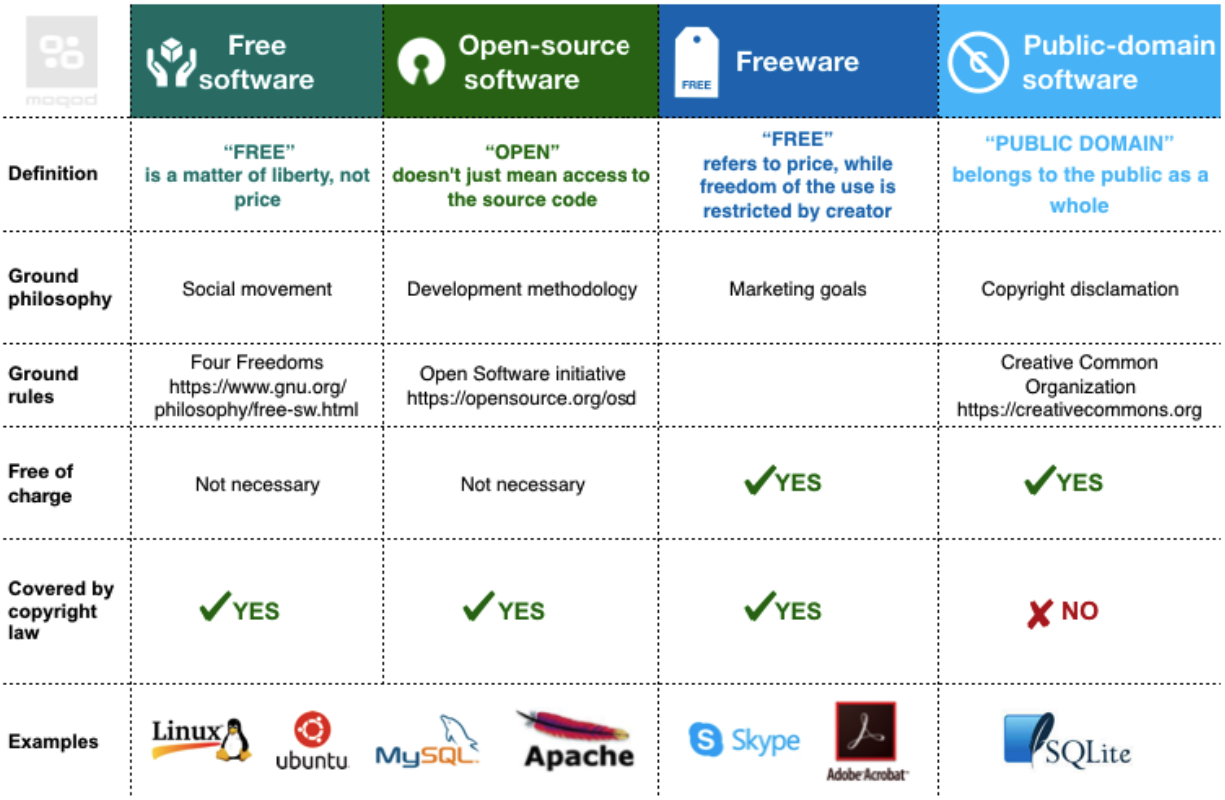

Abbildung 14: Vergleich: Free Software, Open-Source-Software, Freeware und Public-Domain-Software, Quelle: Stribos (2019), Online-Quelle [20.09.2020].

Da sich diese Arbeit auf die Entwicklung einer Applikation bezieht, sind Lizenzen für Source-Codes eher relevant. Deswegen sollen nun die zehn Definitionen für eine Lizenz von der Open-Source-Initiative angeführt werden. Auf eine Unterscheidung zwischen Free Software und Open-Source-Software wird hier nicht weiter eingegangen, denn auch GPL-Lizenzen wurden von der Open-Source-Initiative aufgenommen. Ist die Überschrift nicht eindeutig genug, folgt eine kurze Beschreibung:[28]

1. **Freie Neuverteilung**
   Eine Lizenz darf bei der Verwendung der Software keine Einschränkungen bezüglich des Verkaufens oder Verschenkens beinhalten. Es dürfen keine Lizenzgebühren oder sonstige Gebühren beim Weiterverkauf der ursprünglichen Lizenz verlangt werden.
2. **Quell-Code**
   Das vertriebene Programm muss den Source-Code enthalten und die Verbreitung im Source-Code und in kompilierter Form erlauben. Ist der Source-Code nicht im Produkt enthalten, muss es eine gut publizierte Möglichkeit geben, ihn für nicht mehr als angemessene Reproduktionskosten zu erhalten. Vorzugsweise durch kostenloses Herunterladen im Internet.
3. **Abgeleitete Werke**
   Eine Lizenz muss abgeleitete Werke und Modifikationen erlauben. Auch die Verbreitung unter den gleichen Bedingungen, wie die Lizenz der Originalsoftware, muss zugelassen sein. Absichtlich verschleierter Code ist nicht zulässig.

[28] Vgl. Open Source Initiative (2007), Online-Quelle [21.09.2020].

4. **Integrität des Quellcodes des Autors**
   Die Weitergabe des Quellcodes in modifizierter Form darf nur eingeschränkt werden, wenn eine Weitergabe des Quellcodes mit „Patch-Dateien“ zum Zweck der Modifikation eines Programmes zur Erstellungszeit erlaubt ist. Des Weiteren kann verlangt werden, dass abgeleitete Werke andere Namen und Versionsnummern erhalten müssen.
5. **Keine Diskriminierung von Personen oder Gruppen**
6. **Keine Diskriminierung von Branchen**
7. **Verteilung der Lizenz**
   Die Rechte, welche mit dem Programm verbunden sind, müssen auch für all jene gelten, an die das Programm weitergegeben wird.
8. **Lizenz darf nicht produktspezifisch sein**
9. **Lizenz darf andere Software nicht einschränken**
10. **Lizenz muss technologieneutral sein**

In Abbildung 15 sind einige Lizenzen mit einzelnen Eigenschaften angeführt. Die erste Eigenschaft, *Type*, ist meist die Wichtigste. Sie gibt an, ob die Lizenz freizügig oder mit einem Copyleft versehen ist. Copyleft bedeutet, dass von einem Source-Code, welcher unter Copyleft steht, abgeleiteter oder damit implementierter Source-Code wieder unter dieser oder einer kompatiblen Lizenz stehen muss. Dabei gibt es auch verschiedene Varianten, was als Implementation gilt. So muss beispielsweise Source-Code, welcher GPL-Bibliotheken linkt unter die GPL-Lizenz gestellt werden. Werden hingegen LGPL Bibliotheken gelinkt, kann dieser Source-Code auch proprietär vertrieben werden. Freizügige Lizenzen erlauben die Implementierung in proprietärer Software meist mit geringen Pflichten, z.B. Nennung der Autorin oder Autors.

	APACHE	BSD	MIT	GPL Free as in Freedom	LGPL Free as in Freedom	AGPL Free as in Freedom
Type	**Permissive**	**Permissive**	**Permissive**	**Copyleft**	**Copyleft**	**Copyleft**
Provides copyright protection	✓ TRUE	✓ TRUE	✓ TRUE	✓ TRUE	✓ TRUE	✓ TRUE
Can be used in commercial applications	✓ TRUE	✓ TRUE	✓ TRUE	✓ TRUE	✓ TRUE	✓ TRUE
Provides an explicit patent license	✓ TRUE	✗ FALSE	✗ FALSE	✗ FALSE	✗ FALSE	✗ FALSE
Can be used in proprietary (closed source) projects	✓ TRUE	✓ TRUE	✓ TRUE	✗ FALSE	✗ FALSE partially	✗ FALSE for web
Popular open-source and free projects	Kubernetes Swift Firebase	Django React Flutter	Angular.js JQuery, .NET Core Laravel	Joomla Notepad++ MySQL	Qt SharpDevelop	SugarCRM Launchpad

Abbildung 15: Übersicht über ausgewählte Lizenzen, Quelle: Stribos (2019), Online-Quelle [20.09.2020].

8 ENTWURF SYSTEM-LAYOUT

In diesem Kapitel wird das grundlegende Layout des Systems für die Datenerfassung und Speicherung in Online-Datenbanken erarbeitet. Die Entwicklung des Systems erfolgt unter der Berücksichtigung des V-Modells, um einen strukturierten Projektablauf sicherzustellen. Zuerst werden Spezifikationen angeführt, um mit diesen eine Systemarchitektur zu entwerfen. Im Anschluss werden weitere Funktionen des Systems definiert. In den darauffolgenden Kapiteln werden die einzelnen Systemkomponenten entwickelt.

8.1 Spezifikationen

In der Einleitung wird bereits das Ziel dieser Arbeit angeführt, hier soll nun eine genaue Spezifikation des zu entwickelnden Systems definiert werden. Die grundlegende Funktion des Systems ist die Erfassung von Daten mit unterschiedlichen Protokollen und die Speicherung dieser in Online-Datenbanken. Im Zuge dieser Arbeit kommen nur Geräte als Datenquelle infrage, welche über eine Ethernet- oder WLAN-Schnittstelle verfügen. Es sind keine Protokolle definiert, mit denen Daten gesammelt werden, denn der Fokus bei den Datenquellen liegt bei der einfachen Implementierung neuer Protokolle. Deshalb werden im weiteren Verlauf zwei Protokolle für die Datensammlung ausgewählt und in diesem Projekt integriert. Bei den Datenbanken sollen mehrere Typen getestet werden und das zukünftige Hinzufügen zusätzlicher Datenbanken soll mit geringem Aufwand möglich sein.

Deswegen soll das System modular aufgebaut werden, sodass für jeden Typen, ob Protokoll oder Datenbank, ein eigenes Modul zur Verfügung steht. Dahingehend sollen einheitliche Schnittstellen definiert werden, um bei zukünftigen Neuentwicklungen eine einfache Realisierung zu ermöglichen.

Diesen modularen Aufbau soll auch eine Systemarchitektur unterstützen, bei der die unterschiedlichen Programmteile auf unterschiedlichen Computersystemen lauffähig sind. Somit kann erreicht werden, dass die Datensammlung näher an den Geräten erfolgt. Die Daten sollen direkt nach dem Erfassen zwischengepuffert und mit größeren Datenpaketen weitergeleitet werden, um Netzwerkressourcen zu schonen.

Ein weiterer wichtiger Punkt ist die Skalierbarkeit. Das System soll für kleine sowie für große Datensammlungen geeignet sein. Mit dem modularen Aufbau wird diesem Aspekt bereits Rechnung getragen.

Plattformunabhängigkeit wird in der Industrie immer wichtiger und deshalb soll das System auch auf den Betriebssystemen Windows und Debian laufen. Obwohl Microsoft mit seinem Betriebssystem Windows noch eine Art Monopolstellung hat, werden vor allem Linux-Betriebssysteme in der Industrie immer wichtiger.

Wie bereits im Kapitel „Sicherheit bei der Datenübertragung“ erläutert wird, werden immer mehr Sicherheitsanforderungen an Automatisierungssysteme gestellt. Deshalb muss das zu entwerfende System auch über eine verschlüsselte Datenübertragung verfügen.

Zusammenfassung der **Spezifikation:**

- Erweiterbar
- Modularer Aufbau
- Ermöglichung eines verteilten Aufbaus
- Pufferung von Daten
- Skalierbar
- Plattformunabhängig
- Sichere Datenübertragung

8.2 System-Architektur

Unter Berücksichtigung der oben genannten Spezifikationen, soll nun eine System-Architektur entworfen werden.

Als Voraussetzung für detaillierte Planungen muss zuerst die Entwicklungsumgebung und die verwendete Technologie definiert werden. Da keine genaue Technologie spezifiziert ist, sondern nur eine Plattformunabhängigkeit erreicht werden soll, wird hier Visual Studio 2019 als Entwicklungswerkzeug definiert und .NET Core als Framework. Visual Studio 2019 ist ein Produkt von Microsoft für die Softwareentwicklung, mit dem verschiedene Sprachen verwendet werden können. Beispiele hierfür sind Python, C, C++, JavaScript oder C#, welche für die Realisierung dieses Projektes verwendet wird. .NET Core wird ausgewählt, da die Laufzeitumgebung des Frameworks auf Windows und auf Linux installiert und somit betrieben werden kann. Es ist eine Open-Source-Entwicklung und stellt bereits viele Funktionalitäten bereit. Die Anwendung benötigt keine grafische Bedienoberfläche, denn im Betrieb soll das System im Hintergrund arbeiten. Deshalb wird es als Konsolen-Programm ausgeführt.

Gemäß der Anforderung eines modularen Aufbaus wird in diesem System mit drei Modulen gearbeitet:

- **Data-Collector**
  Diese Applikation soll die Daten von Geräten holen, wobei für jedes Protokoll eine eigene Applikation geschrieben wird. Mit dieser Architektur können neue Protokolle einfach implementiert werden. Der Data-Collector soll komplett autonom laufen, sodass für den normalen Betrieb keine Verbindung zum Manager benötigt wird. Das bedeutet, dass eine Möglichkeit für die Datenpufferung vorgesehen wird. Die Daten werden über einen Mechanismus der Manager-Applikation angeboten, welcher im weiteren Verlauf definiert wird.
- **DB-Connector**
  Der DB-Connector soll die Daten in eine Online-Datenbank transferieren. Dabei soll pro Online-Datenbank eine Applikation entwickelt werden. Das ist nötig, denn für die Überführung der Daten in unterschiedliche Online-Datenbanken sind meist unterschiedliche Mechanismen erforderlich. Wird für jeden Type eine eigene Applikation erstellt, können diese Mechanismen für jede Datenbank angepasst und auch gewartet werden. Auch bei dieser Applikation soll ein Datenpuffer angelegt werden, um bei Ausfall der Internetverbindung keine Daten zu verlieren.

- **Manager**

  Pro System soll es eine Manager-Applikation geben, welche als Bindeglied zwischen den beiden anderen Applikationen dienen soll. Sie soll einerseits die gelesenen Daten eines Gerätes vom Data-Collector einsammeln und diese über den Data-Connector an eine Online-Datenbank senden. Der Manager soll auch das Handling für die Konfigurationskontrolle übernehmen. Die Applikation soll mehrere Operationen gleichzeitig ausführen können, wodurch Multithreading benötigt wird.

Durch diese Architektur des Systems werden einige Punkte der Spezifikationen erreicht. Durch die Aufsplittung der einzelnen Funktionen in separate Programme können neue Protokolle oder Datenbankanbindungen einfach realisiert und implementiert werden. Somit ist auch der modulare Aufbau gegeben und einzelne Teile können auf verteilten Rechnern laufen. Das bringt auch eine Skalierbarkeit mit sich, denn werden mehr Daten von verschiedenen Geräten gebraucht, müssen nur einzelne Module hinzugefügt werden.

Die Konfiguration der Applikationen soll mittels Konfigurationsdateien erfolgen. Eine genaue Beschreibung erfolgt im nächsten Punkt dieses Kapitels, aber es soll bereits vorgegriffen werden, dass eine Überprüfung dieser Konfigurationsdateien stattfinden soll.

Um den Anforderungen in der Automatisierungstechnik gerecht zu werden, muss die Kommunikation zwischen den einzelnen Applikationen verschlüsselt stattfinden.

In Abbildung 16 ist nun die grundsätzliche System-Architektur dargestellt. Der Datenfluss erfolgt von links nach rechts. Die Data-Collectors sammeln die Daten von den verschiedenen Geräten und stellen diese, der Manager-Applikation zur Verfügung. Wie aus der Darstellung zu entnehmen ist, gibt es für jedes Gerät einen eigenen Data-Collector, wobei auch der gleiche Data-Collector mehrfach verwendet werden kann. Der Manager gibt die Daten nach Erhalt an den gewünschten DB-Connector weiter. Die DB-Connectors organisieren die Verbindung zu der ausgewählten Online-Datenbank und transferieren die Daten.

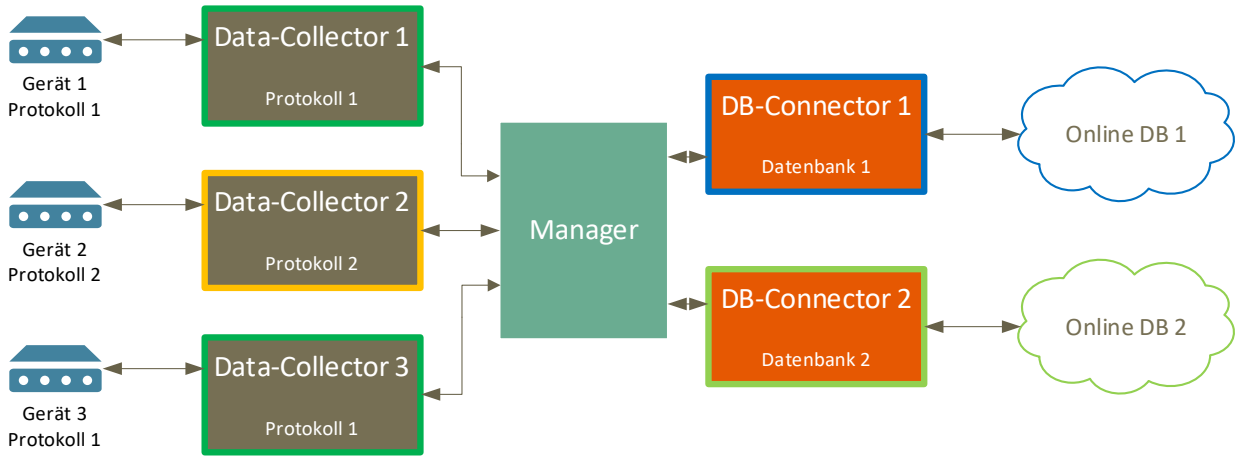

Abbildung 16: Grundsätzliche System-Architektur, Quelle: Eigene Darstellung.

8.3 Konfigurationsdateien

Mittels Konfigurationsdateien soll eine einfache Konfiguration des Systems ermöglicht werden. Von einer grafischen Bedienung wird Abstand genommen, da die Datensammlung im Hintergrund ablaufen soll.

Die Konfigurationsdateien werden im JSON-Format erstellt. JSON ist ein Datenaustauschformat, welches unter anderem in Webapplikationen eingesetzt wird. Es können hierarchische Strukturen aufgebaut werden und es unterstützt unterschiedliche Typen: Boolean, Zahlen, Strings, Arrays oder Objekte. Das Datenformat ist leicht leserlich und benötigt weniger Syntax wie XML.

Bei Betrachtung des Gesamtsystems wird ersichtlich, welche veränderbaren Eigenschaften verfügbar sein müssen:

- Quellen von Daten (IP-Adresse, Port, …)
- Adresse einer Variablen am Gerät (Name- oder Adressdefinitionen)
- Eigenschaften einer Variablen (Datentype)
- Abtastzeit der Variablen
- Allgemeine Einstellungen der einzelnen Applikationen (Logger, Pfade, …)
- Angaben, welche Data-Collectors verwendet werden und Adressdaten (IP-Adresse, Port, …)
- Online-Datenbank-Adresse, wohin die Daten transferiert werden sollen
- Definitionen, welche Variable vom Data-Collector, auf welchen Datenpunkt in der Online-Datenbank gespeichert werden soll
- Etwaige Authentifizierungsdaten, um einen sicheren Datenaustausch zu gewährleisten

Um eine gewisse Unabhängigkeit der einzelnen Applikationen zu erreichen, sollen verschiedene Konfigurationsdateien definiert werden. Dabei soll eine Konfigurationsdatei immer einen funktionellen Verbund bilden und die einzelnen Datenpunkte müssen im gesamten System eindeutig referenzierbar sein.

Somit werden drei Konfigurationsdateien definiert:

1. Conf_DataCollector:
   Für jeden Data-Collector soll eine eigene Konfigurationsdatei angelegt werden. Diese soll vom Data-Collector und der Manager-Applikation geladen werden. Durch eine Operation soll validiert werden, ob beide Applikationen die gleiche Datei verwenden. Durch diese Art der Konfiguration ist es auch möglich, Einträge hinzuzufügen, welche nur von einer Applikation genutzt werden. Das bedeutet, dass Einträge in der Konfigurationsdatei für beide Applikationen definiert werden, andere nur für den Data-Collector oder Manager. Die Einträge, auf welche beide Applikationen zugreifen, müssen in allen Konfigurationsdateien enthalten sein, andere können je nach Anforderung des Protokolls hinzugefügt werden.
2. Conf_DBConnector:
   Für jede Datenbankanbindung wird eine eigene Konfigurationsdatei benötigt. Darin werden Pfade angegeben, die für die Ausführung der einzelnen DB-Connectors benötigt werden. Die genaue Art der Komponente (DB-Connector), wird im Kapitel 8.4.2 Manager und DB-Connector definiert. In der Konfigurationsdatei wird auch definiert, in welche Online-Datenbank die Daten transferiert werden. Dafür werden Zieladresse und Authentifikationsdaten der Online-Datenbank benötigt. Es

muss eine genaue Zuordnung der gesammelten Datenpunkte zu einem Datenpunkt in der Online-Datenbank hergestellt werden können. Um diese Anforderung zu erfüllen, wird in dieser Konfigurationsdatei, für jeden Datenpunkt, der in diese Online-Datenbank transferiert werden soll, der Name des Data-Collectors und der Variable angeführt. Diese Konfigurationsdatei soll von Manager und DB-Connector verwendet werden.

3. Conf_Manager:
   In der Konfigurationsdatei Conf_Manager wird definiert, welche Komponenten geladen werden. Dafür wird in dieser Konfigurationsdatei je eine Liste für den Speicherort der anderen verwendeten Konfigurationsdateien (Conf_DataCollector und Conf_DBConnector) angelegt.

Die genaue Definition, welche Einträge in den Konfigurationsdateien enthalten sind, kann erst nach der Definition der Kommunikation zwischen den Applikationen erstellt werden. Es sind noch einige Tatsachen ungeklärt, weshalb die Definition der Einträge erst später vorgenommen wird.

8.4 Kommunikation zwischen Applikationen

Die grundsätzliche System-Architektur ist definiert. Der nächste Punkt, welcher in diesem Unterkapitel behandelt wird, ist die Kommunikation zwischen den einzelnen Komponenten. Dafür sollen die Anforderungen begutachtet und die richtige Lösung dafür ausgewählt werden. Eingangs dieses Entscheidungsprozesses soll erwähnt werden, dass nicht zwischen allen Applikationen die gleiche Kommunikationsart stattfinden muss. Hier soll neben den Kommunikationsarten die Schnittstellen definiert werden, sodass bei Erweiterungen von Modulen nur diese implementiert werden müssen. Ein wichtiges Augenmerk wird auch auf die Verschlüsselung von Daten gelegt.

8.4.1 Data-Collector und Manager

Wie in den Spezifikationen angeführt, ist es sinnvoll, das System so zu entwerfen, dass der Data-Collector auf einem anderen Computer als der Manager ausgeführt werden kann. Gründe dafür sind Skalierbarkeit, Modularität und Erweiterbarkeit. In Kapitel 6.1 werden bereits drei unterschiedliche Arten für diese Kommunikation behandelt:

- **OPC UA**
- **REST-API**
- **MQTT**

Aus diesen Varianten soll nun mittels einer Entscheidungsmatrix (Tabelle 3) die richtige Variante gefunden werden. Es sind die drei Varianten angeführt und einzelne Kriterien für die Bewertung. Für jedes Kriterium werden Punkte zwischen Eins und Drei vergeben, wobei Drei die Bestnote ist und für jedes Kriterium mehrfach die gleiche Note vergeben werden kann.

Alle Varianten sind routbar weshalb bei diesem Punkt alle die Bestnote bekommen. MQTT ist gegenüber OPC UA schneller und benötigt auch weniger Overhead.[29] OPC UA kann Daten schneller lesen, wenn

[29] Vgl. Drolshagen (2015), Online-Quelle [25.09.2020], S. 96.

Arrays oder Strukturen gelesen werden. Die Verbreitung von REST-APIs ist am größten und auch die IT-Freundlichkeit mit Kommunikation über Port 80 oder 443 hat Vorteile. Bei OPC UA und MQTT ist die Übertragung standardisiert, wohingegen bei REST-API komplette Freiheit besteht. Auch die Schnelligkeit ist ein wichtiger Punkt, wobei hier REST-API OPC UA schlägt.[30] Aufgrund der Auswertung fällt somit die Wahl für die Kommunikation auf **REST-API**.

		Varianten		
		OPC UA	REST API	MQTT
Kriterien	Routbar	3	3	3
	Schnelligkeit	1	2	3
	Sicherheit	3	3	2
	User Freundlich	2	3	1
	Standardisierte Übertragung	3	1	3
	Verbreitung	2	3	1
	Overhead	1	2	3
	Ressourcenbedarf	1	2	3
	IT-Freundlichkeit	2	3	2
Summe		18	22	21

Tabelle 3: Entscheidungsmatrix: Kommunikation Data-Collector und Manager, Quelle: Eigene Darstellung.

Schnittstellenarchitektur:

Nachdem die Wahl der Kommunikation feststeht, muss nun die Architektur definiert werden. Wie in Abschnitt 6.1.2 erläutert, ist REST-API eine Client-Server-Architektur. Bei der Definition, welche der Applikationen (Data-Collector, Manager) Server oder Client sein soll, sind folgende Punkte zu beachten:

- Es können pro System mehrere Data-Collectors definiert werden.
- Pro System gibt es immer nur einen Manager, welcher die Daten von den Data-Collectors bezieht.
- Der Manager soll die Verbindung initialisieren.

Nach Evaluierung dieser Punkte wird definiert, dass die Data-Collectors die Server-Funktion übernehmen und der Manager die Client-Funktion. Die Daten-Collectors sammeln die Daten, wie es ihnen mittels Konfigurationsdateien vorgeschrieben wird, speichert diese und wenn der Manager Daten benötigt, fragt er als Client die Daten ab. Der Manager benötigt dabei nur die Server Daten wie Adresse, Port und einen Endpunkt zum Abfragen der Daten.

In Abbildung 17 ist der systematische Aufbau grafisch dargestellt. In der Abbildung befinden sich drei Data-Collectors mit je einem REST-API-Server. Die Data-Collectors können unabhängig vom Manager gestartet werden, denn sie speichern die Daten, solange sie nicht abgefragt werden. Will ein Manager die auf einem Data-Collector gespeicherten Daten abfragen, sendet er einen Request und bekommt die am Data-Collector gespeicherten Daten zurück.

[30] Vgl. Rinaldi (2019), Online-Quelle [25.09.2020].

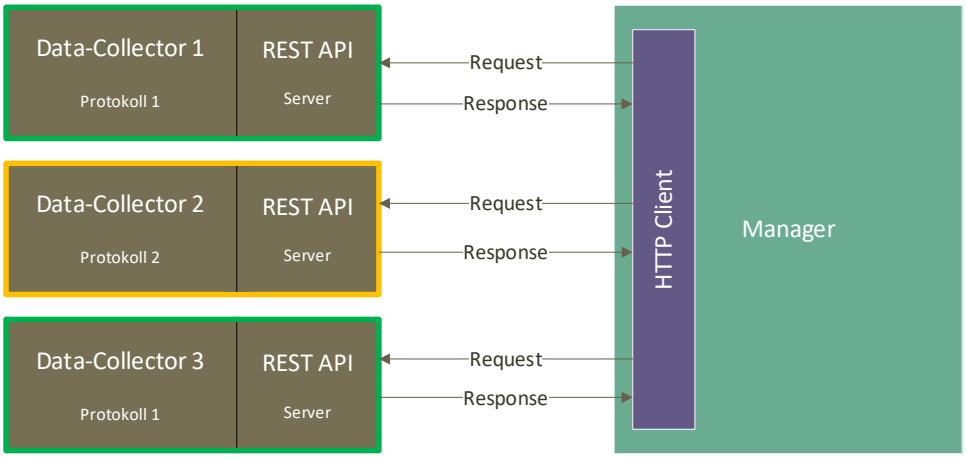

Abbildung 17: Kommunikation Data-Collectors und Manager, Quelle: Eigene Darstellung.

Schnittstellendefinition:

Wenn verschiedene Applikationen entwickelt werden (Data-Collectors) ist es unbedingt notwendig, eine einheitliche Schnittstelle festzulegen. Damit kann vermieden werden, dass bei späteren Implementierungen keine Änderungen an einem bestehenden Programm vorgenommen werden müssen. Deshalb sollen alle bekannten Eventualitäten miteinbezogen werden.

Bei der späteren Realisierung sollen deshalb folgende Abfragen implementiert werden:

- Hash der Konfigurationsdatei “Conf_DataCollector”
  Mit dieser Abfrage kann geprüft werden, ob ein Data-Collector und die Manager-Applikation die gleiche Konfigurationsdatei verwenden. Der Data-Collector soll einen Hash-Code über seine verwendete Konfigurationsdatei bilden und per API zur Verfügung stellen. Bei erstmaliger Verbindung zwischen Data-Collector und Manager, fragt der Manager den Hash-Code des Data-Collectors ab. Auch der Manager generiert mit derselben Hash-Funktion den Hash-Wert und prüft, ob die Werte ident sind. Bei Aufruf des Hash-Codes per REST-API wird ein Dictionary zurückgegeben, welches einen Eintrag als Schlüssel-Wert beinhaltet. Der Schlüssel ist ein String mit „Hash“ und der Wert ist ein 95-Byte langer String, wobei jedes drittes Zeichen ein Separator ist. Die Abfrage lautet:
  - https://<Data-Collector IP>:<Data-Collector Port>/api/Hash
- Gesammelte Werte des Data-Collectors
  Für jede Variable soll eine eigene Abfrage zur Verfügung gestellt werden. Um beiden Applikationen die Adressierung der Variablen bekannt zu machen, wird dies mit der Konfigurationsdatei realisiert. Da jede Variable pro Data-Collector eindeutig sein muss, wird eine Abfrage für Daten implementiert, wobei als Parameter der Variablen-Name mitgesendet wird. Als Antwort wird ein Array von Datenpunkten erstellt, wobei jeder Datenpunkt den Wert als *Double* und den Messzeitpunkt als *DateTime*-Format zurückgibt. Es gilt eine maximale Begrenzung der Datenpunkte von 1000 zu implementieren, sodass eine Antwort nicht zu groß wird. Sind keine Datenpunkte vorhanden wird ein leeres Array zurückgegeben. Eine Abfrage lautet:
  - https://<Data-Collector IP>:<Data-Collector Port>/api/Variables?Var=<Variablen-Name aus Konfigurationsdatei>

Verschlüsselung:

Um die Kommunikation zu verschlüsseln, wird TLS verwendet. Die Funktionsweise wird bereits in Kapitel 3.2 erläutert. Zur Authentifizierung und Verschlüsselung soll ein selbst erstelltes Zertifikat mit Public- und Private-Key verwendet werden. Dieses Zertifikat mit Public- und Private-Key wird dann von allen Data-Collectors (REST-API) verwendet. Da der Manager den Private-Key nicht benötigt, wird diesem nur das Zertifikat, welches den Public-Key enthält, mitgeteilt. Um bei einem Kommunikationsaufbau die Authentizität des Data-Collectors gegenüber dem Manger zu bestätigen, schickt der Data-Collector das Zertifikat. Der Manager besitzt auch das Zertifikat und kann prüfen, ob es ident ist. Da das reine Schicken des Zertifikates nicht ausreicht, um die Authentizität zu bestätigen (das Zertifikat könnte abgefangen und von einem anderen Server benutzt werden), wird ein mit dem Public-Key verschlüsseltes Geheimnis vom Manager übertragen, welches nur mit dem Private-Key der Data-Collectors entschlüsselt werden kann. Mit diesem Geheimnis wird dann die Kommunikation synchron verschlüsselt.

Konfigurationseinträge:

Für die Kommunikation zwischen Manager und Data-Collector müssen gewisse Parameter bekannt sein. Da die Konfigurationsdatei *Conf_DataCollector* beiden Applikationen bekannt ist, werden diese Parameter für die Kommunikation in ihr hinterlegt.

- IP-Adresse des Data-Collectors "*IPAddress_DataCollector_API"*
  Diese IP-Adresse muss auf einer Schnittstelle des Systems eingestellt sein, auf dem der Data-Collector betrieben wird. Die REST-API horcht auf dieser Adresse auf Anfragen des Managers. Der Manager benötigt diese, um die Anfragen an den Data-Collector zu stellen.
- Port des Data-Collectors "*Port_DataCollector_API"*
  Auf diesen Port hört der Data-Collector bei Anfragen und der Manager muss auf diesem Port die Anfragen stellen. Es ist darauf zu achten, dass dieser Port nicht von einer Firewall blockiert wird.
- Namen der Variablen "*Name"*
  Um die Anfragen zu unterscheiden, werden die Daten pro Messpunkt separat ausgegeben. Für das Referenzieren auf einen Messpunkt wird der Name des Messpunkts verwendet. Dieser befindet sich im Unterpunkt Variables, welches ein Array aus allen Messpunkten ist.

```
{
  "General": {
    "IPAddress_DataCollector_API": "192.168.0.1",
    "Port_DataCollector_API": "8000"
    },
  "Variables": [
    {
      "Name": "VarName1"
    },
    {
      "Name": "VarName2"
    }
  ]
}
```

Abbildung 18: Parameter in Conf_DataCollector für die Kommunikation, Quelle: Eigene Darstellung.

8.4.2 Manager und DB-Connector

Für die Kommunikation zwischen Manager und DB-Connector bestehen nicht die identen Anforderungen, wie bei der Kommunikation zwischen Data-Collector und Manager. Deshalb werden in diesem Abschnitt die Anforderungen neu evaluiert und eine geeignete Technologie ausgewählt.

Anforderungen:

- Verschiedene DB-Connectors müssen verwendbar sein.
  Laut der Definition gibt es einen Manager, der verschiedene DB-Connectors ansprechen kann. Welche DB-Connectors verwendet werden, wird erst bei Laufzeit, mittels Einträge in der Konfigurationsdatei *Conf_DBConnector*, entschieden. Mit dieser Variante können die verschiedenen DB-Connectors einfach ausgetauscht und verwaltet werden.
- Schnelle Datenübertragung
  Für die Kommunikation zwischen den beiden Komponenten (Manager und DB-Connector) soll eine Technik verwendet werden, welche einen hohen Datendurchsatz gewährleistet. Beim Manager kommen alle Daten von allen Data-Collectors zusammen, weshalb eine schnelle Weiterleitung der Daten unabdingbar ist.
- Sichere Kommunikation
  Der Datenaustausch zwischen den beiden Komponenten soll so gestaltet werden, dass Angreifer keinen Zugriff auf die Daten erhalten.
- Nicht Anforderung: Verteilt
  Es ist keine Anforderung des Systems, dass sich die beiden Komponenten auf unterschiedlichen Systemen befinden. Wie in der Systemarchitektur festgelegt ist, werden alle Daten von den Data-Collectors gesammelt. Eine neuerliche Aufteilung der Daten ist dann nicht mehr sinnvoll. Deshalb können sich Manager und die DB-Connectors auf demselben System befinden.

Möglichkeiten:

In Kapitel 6 sind einige Möglichkeiten für die Kommunikation zwischen Anwendungen angeführt. Die Funktionsweisen der einzelnen Möglichkeiten werden hier nicht nochmals angeführt, sondern können dem referenzierten Kapitel entnommen werden. Aus dem Kapitel ergeben sich folgende Möglichkeiten:

- OPC UA
- REST-API
- MQTT
- Gemeinsamer Speicher
- Nachrichtenwarteschlangen
- Kommunikationskanäle
- Sockets

Eine weitere Möglichkeit ist die Verwendung von Dynamic Link Librarys (DLL). Eine DLL ist keine Kommunikationsart, sondern eine Möglichkeit zum Laden von Code zur Laufzeit. DLLs besitzen Funktionen, welche von den aufrufenden Programmen benutzt werden können.

Auswahl:

OPC UA, Rest-API, *MQTT* und *Sockets* werden nicht berücksichtigt, da sie für die Kommunikation zwischen verschiedenen Computern konzipiert sind. Deshalb haben sie mehr Overhead, welcher für diese Art nicht notwendig ist und wodurch auch langsamere Kommunikationsgeschwindigkeiten resultieren. Die weiteren Interprozesskommunikationsarten werden auch nicht verwendet. Sie wären aufgrund ihrer Geschwindigkeit und der Fähigkeit für die Nutzung verschiedener DB-Connectors grundsätzlich verwendbar, jedoch ist die Realisierung der Schnittstelle aufwendiger. Deshalb wird hiermit DLL gewählt. Jeder DB-Connector wird eine eigene DLL und kann zur Laufzeit geladen werden. Es muss auch auf keine Sicherheitsaspekte geachtet werden, denn die Kommunikation findet im selben Prozess statt.

Ausführung:

Wie bereits angeführt wird, erhält der Manager eine Konfigurationsdatei (*Conf_Manager*) in der die zu verwendenden anderen Konfigurationsdateien (*Conf_DataCollector* und *Conf_DBConnector*) eingetragen werden. Dabei soll sich in der *Conf_DataCollector* ein Eintrag befinden, wo sich die DLL des jeweiligen DB-Connectors befindet. Mit DLLs ist es auch möglich, den Code einer DLL mehrfach im Manager zu verwenden, wodurch auch Speicherplatz gespart wird. Um verschiedene DLLs auf dieselbe Weise nutzen zu können, wird eine einheitliche Schnittstelle definiert. Wird diese komplett in die DB-Connector DLL implementiert, können auch zukünftig entwickelte DB-Connectors verwendet werden.

Schnittstelle:

Für die Kommunikation zwischen den beiden Komponenten (Manager und DB-Connector) werden vier Funktionen definiert. Damit bleibt die Kommunikation übersichtlich und ein neuer DB-Connector muss nur diese vier Funktionen beinhalten.

- **InitCheck**
  Diese Funktion wird als Erstes aufgerufen. Dabei werden diverse Parameter übergeben (Speicherort der Konfigurationsdatei Conf_DBConnector, Speicherort der Puffer-Datenbank, Hash-Wert der Konfigurationsdatei Conf_DBConnector vom Manager). Bei der Methode *InitCheck(…)* werden alle erforderlichen Objekte initialisiert und geprüft, ob sie für die Ausführung bereit sind. Tritt ein Fehler auf, wird ein Wert kleiner Null zurückgegeben, anderenfalls Eins.
- **GetIdxOfVars**
  Um speicheroptimierter zu arbeiten, wird vom Manager jedem Messpunkt eine eindeutige ID zugewiesen. Diese ID wird in dieser Funktion der DLL mitgeteilt. Übergeben wird die ID, der Name des Data-Collectors und der Name der Variable. Mit der ID muss nicht bei jedem Messwert der String des Data-Collectors und der Variable mitgesendet werden, sondern lediglich ein UINT mit der ID.
- **StartPostVarsToOnlineDB**
  Mit dieser Funktion wird das Transferieren der Daten in die Online-Datenbank gestartet. Diese Funktion soll aufgerufen werden, wenn alle Initialisierungen und Konfigurationen abgeschlossen sind.
- **ReceiveValues**
  Mit dieser Funktion werden der DLL die Messwerte übergeben. Als Parameter müssen die ID, der

Zeitpunkt der Messung und der Messwert übergeben werden. Intern kann mittels der ID die Zuweisung auf die Einträge in der Konfigurationsdatei geschlossen werden.

In Abbildung 19 ist die Einbindung von zwei DLLs visuell dargestellt. Der Manager lädt zur Laufzeit die gewünschten DLLs und kann mit ihnen, mit den Befehlen der definierten Schnittstelle, kommunizieren. Hierbei ist auch zu sehen, dass mehrere DLLs gleichzeitig geladen werden können, wobei auch eine DLL mehrmals geladen werden kann.

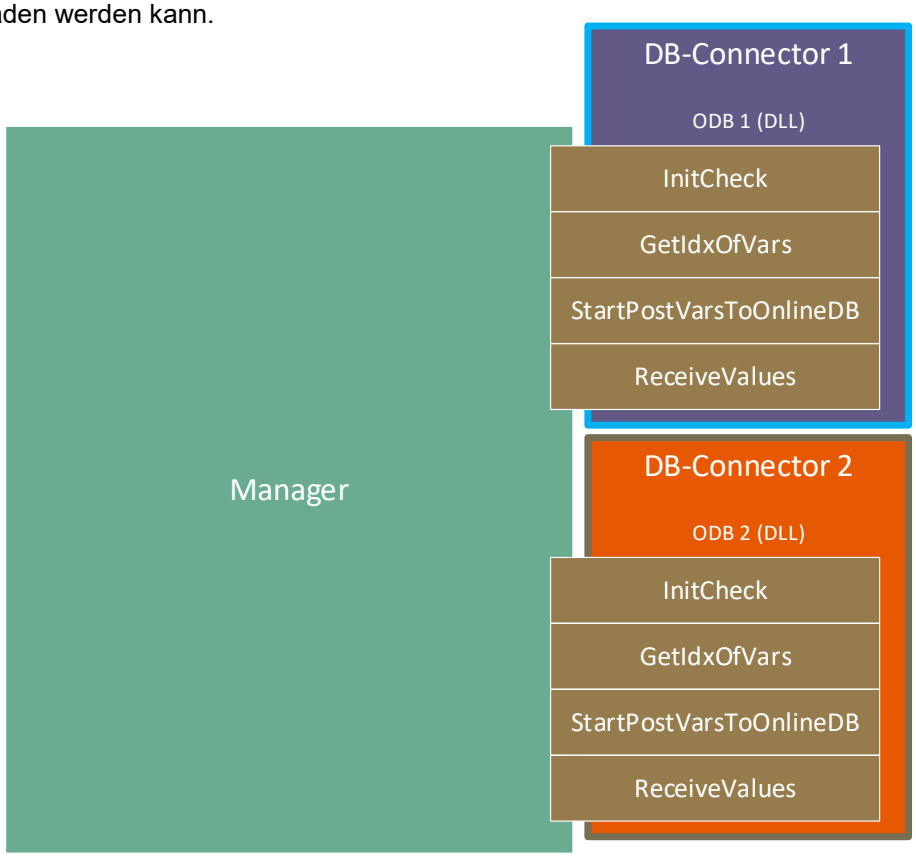

Abbildung 19: Einbindung von DB-Connectors (DLL) im Manager, Quelle: Eigene Darstellung.

8.4.3 Zu Online-Datenbanken

Auf die Verbindung zwischen den DB-Connectors und den Online-Datenbanken kann in dieser Phase noch nicht eingegangen werden, denn die zu verwendenden Online-Datenbanken stehen noch nicht fest. Jede Anbindung einer Online-Datenbank kann individuell passieren, weshalb erst bei der Implementierung darauf eingegangen wird.

8.5 Realisierung

In den nachfolgenden Kapiteln werden die einzelnen Module des Systems genau definiert und anschließend umgesetzt. Begonnen wird mit den Data-Collectors, gefolgt von den DB-Connectors und abschließend der Manager. Zu Beginn jedes Kapitels werden, falls nötig, die Technologien ausgewählt. Anschließend wird die grundlegende Architektur festgelegt und abschließend erfolgt die Umsetzung.

9 DATA-COLLECTORS

In diesem Kapitel erfolgt die genaue Projektierung und Realisierung von zwei Data-Collectors. Eingangs werden die zu entwerfenden Protokolle ausgewählt, danach die grundsätzliche Architektur definiert und anschließend die einzelnen Komponenten realisiert.

9.1 Auswahl von Protokollen

Da im Zuge dieser Arbeit ein System erstellt wird, welches mit verschiedenen Protokollen von unterschiedlichen Geräten Daten erfasst, werden nun zwei Protokolle ausgewählt, die realisiert werden.

9.1.1 OPC UA

Das erste Protokoll, welches realisiert wird, ist OPC UA. Es wird im Zuge dieser Arbeit bereits zwei Mal angeführt. In Kapitel 2.3 wird auf die Verbreitung des Protokolls eingegangen woraus hervorgeht, dass OPC UA als Standard für den Kommunikationsaustausch zwischen Systemen erachtet wird. Darauf kann auch geschlossen werden, wenn auf Websites bekannter Automatisierungshardware-HerstellerInnen wie Siemens[31], Rockwell[32] oder B&R[33] nach OPC UA recherchiert wird. So ist mittlerweile fast bei jeder SPS ein OPC UA Server standardmäßig integriert. Durch die Verbreitung und Plattformunabhängigkeit dieses Protokolls ist es prädestiniert für ein System zur Datenerfassung von verschiedenen Geräten. OPC UA lässt sich im OSI-Modell in Layer fünf bis sieben eingliedern und kann deshalb jeweils einen anderen Unterbau für die Datenübertragung verwenden. Deshalb sind auch Übertragungen über Drahtlosnetzwerke möglich. OPC UA ist für den Datenaustausch von Datenpunkten zwischen Geräten entworfen worden und deshalb gibt es eine Fülle an Definitionen und Methoden, welche für die Realisierung genutzt werden können.

9.1.2 TCP/IP

Als zweite Variante soll eine Datenerfassung mittels TCP/IP Verbindung entwickelt werden. TCP/IP befindet sich auf OSI-Layer vier und drei und auf diese Protokolle bauen viele andere auf, wie HTTP, FTP oder OPC UA. Fast jedes Gerät, welches eine Ethernet-Schnittstelle besitzt, kann auch über TCP/IP kommunizieren. Da dieses Protokoll auf verschiedene Übertragungsmedien aufsetzen kann, ist auch hier eine Übertragung sowohl per Kabel als auch kabellos möglich. Ein Nachteil dieses Protokolls ist, dass es rein für die Übertragung konzipiert ist und damit keine Definitionen besitzt, wie Datenpunkte ausgetauscht werden können. Deshalb muss bei der Realisierung ein Schema definiert werden, wie die Datenpunkte von einem Gerät zu dem Data-Collector übertragen werden.

[31] Vgl. Siemens AG (o. J.), Online-Quelle [04.10.2020].

[32] Vgl. Rockwell Automation (2018), Online-Quelle [04.10.2020].

[33] Vgl. B&R (o. J.), Online-Quelle [04.10.2020].

9.2 Architektur

In diesem Unterkapitel wird die generelle Architektur des Data-Collectors definiert. Diese soll bei allen Data-Collectors annähernd ident sein, um die Wartung aber auch die Entwicklung neuer Data-Collectors zu erleichtern.

Ein Data-Collector kann als Summe zweier Teile gesehen werden:

- **Datensammler**
  Dieser ist bei jedem Protokoll unterschiedlich, denn in ihm sind die Funktionalitäten für das jeweilige Protokoll hinterlegt.
- **Webserver**
  Der Webserver beinhaltet die Zwischenspeicherung der Daten und die REST-API. Diese Funktionalität ist bei allen Data-Collectors ident.

Um eine strikte Trennung der beiden Teile zu gewährleisten, werden diese auch als verschiedene Applikationen entwickelt. Dabei soll der Datensammler, mit den Protokoll-Funktionalitäten, den Webserver aufrufen. Durch definierte Schnittstellen muss der Webserver nur ein einziges Mal entwickelt werden und kann von allen Datensammlern aufgerufen werden. Die Datensammler werden mit „.Net Core 3.1“ entwickelt. Für den Webserver wird „ASP.NET Core 3.1“ verwendet, was ein Framework von Microsoft ist und plattformübergreifend eingesetzt werden kann. Es ist aber speziell für Webanwendung und APIs entworfen worden und bietet standardmäßig Funktionalitäten wie Routing, Authentifikationsfunktionen oder Datenbankzugriff bei HTTP-Requests an.

9.2.1 Aufbau

In Abbildung 20 ist der generelle Aufbau der Architektur dargestellt. Gestartet wird der Data-Collector über eine .exe-Datei. Der grundsätzliche Aufbau der beiden Applikationen wird anschließend erörtert.

Datensammler:

1) **Lesen der Konfigurationsdatei**
   Der Datensammler öffnet die Konfigurationsdatei, welche im selben Verzeichnis wie die .exe-Datei gespeichert sein muss. Daraus werden alle relevanten Parameter gelesen, die im Programm verwendet werden.
2) **Webserver starten**
   Unter anderem enthält die Konfigurationsdatei den Pfad, wo der Webserver abgelegt ist. Dieser muss in einem separaten Thread gestartet werden, um den Datensammler nicht zu blockieren.
3) **Hash-Wert prüfen**
   Beim Lesen der Konfigurationsdatei wird von dieser auch ein Hash-Wert gebildet. Dieser Wert wird mit dem Webserver verglichen. Stellen die Applikationen einen Unterschied fest, wird mit dem Programmablauf nicht fortgefahren und es wird ein Logfile-Eintrag generiert.
4) **Objekte für Variablen erstellen**
   Mit den Informationen aus der Konfigurationsdatei kann für jede Variable, die gelesen werden soll, ein Objekt generiert werden. Dieses Objekt beinhaltet verschiedene Operationen, welche auf jede

Variable angewendet werden. Des Weiteren findet in diesem Objekt die Datenübertragung zum Webserver statt.

5) **Protokollfunktionalität**

   Nach Erstellung der Objekte, für jede Variable, werden protokollspezifische Funktionalitäten angewendet. Diese managen die Datensammlung und verweisen die erhaltenen Werte an die Variablen-Objekte, wo die Werte an den Webserver gesendet werden.

Webserver:

1) **Lesen der Konfigurationsdatei**

   Auch der Webserver liest direkt nach dem Starten die Konfigurationsdatei aus, welche sich auch im selben Verzeichnis wie die .exe-Datei befinden. Des Weiteren wird beim Auslesen auch der Hash-Wert der Konfigurationsdatei gebildet und für den späteren Abgleich abgespeichert.

2) **Konfigurieren des Webservers**

   Direkt nach dem Auslesen der Konfigurationsdatei, müssen einzelne Parameter des Webservers definiert werden. Viele davon können im laufenden Betrieb nicht mehr geändert werden.

3) **Hash-Wert prüfen**

   Ist der Server konfiguriert, vergleicht der Webserver den erhaltenen Hash-Wert vom Datensammler mit dem eigenen. Stimmen diese Werte nicht überein, wird die Abarbeitung des Programmes beendet.

4) **Daten empfangen**

   Sind alle Vorbereitungen erfolgreich abgeschlossen, wartet der Webserver auf Daten vom Datensammler. Dabei gibt es auch für jede Variable ein eigenes Objekt, in dem die Daten empfangen werden. Von diesem Objekt werden die Daten in eine Queue und von dieser dann in eine Datenbank geschrieben. Jeder empfangener Variablenwert besitzt auch einen eindeutigen Zeitstempel, um eine genaue zeitliche Zuordnung zu gewährleisten.

5) **Daten hosten**

   Sobald der Webserver bereit ist, können über die REST-API Anfragen zu Variablen gestellt werden. Aber auch der Hash-Wert der Konfigurationsdatei kann für den späteren Abgleich abgefragt werden. Bei einer Abfrage einer Variablen werden die Daten dieser Variable aus der Datenbank abgefragt und im JSON-Format an den Client zurückgesendet. Es ist ein maximales Limit für die Anzahl der Variablen hinterlegt, sodass ein Request nicht zu lang wird. Sind die Werte ausgegeben, werden diese Daten aus der Datenbank gelöscht. Bekommt der Webserver eine leere Liste, sind keine Daten mehr am Webserver vorhanden.

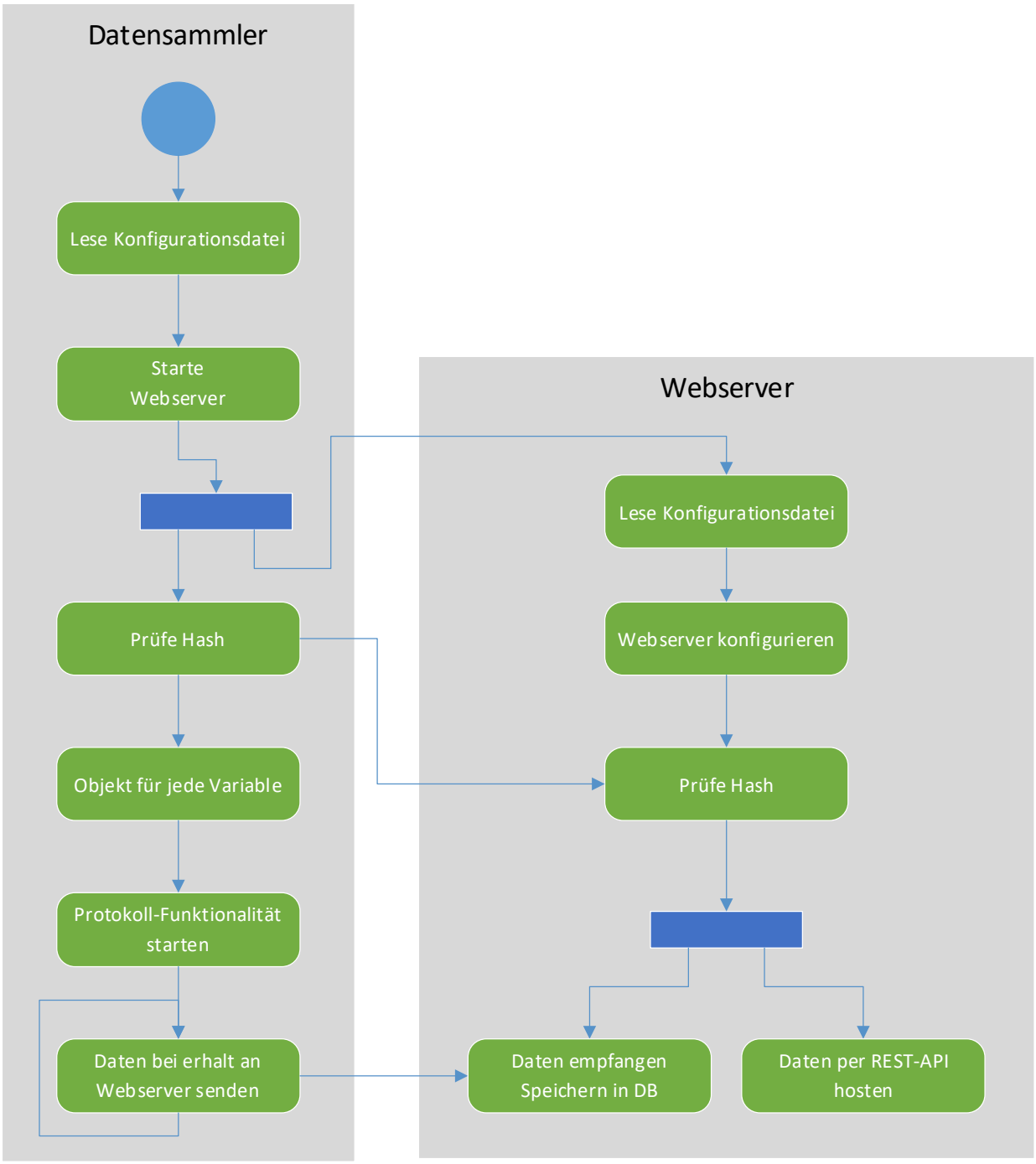

Abbildung 20: Genereller Aufbau Data-Collector, Quelle: Eigene Darstellung.

9.2.2 Schnittstelle

Aufgrund der Entscheidung, für den Data-Collector zwei unterschiedliche Applikationen (Datensammler und Webserver) zu entwickeln, muss eine Technik für die Kommunikation zwischen den beiden Applikationen gefunden werden. In Kapitel 6.2 werden bereits die Möglichkeiten für die Interprozesskommunikation angeführt, welche nun für die Auswahl mit einbezogen werden.

Eine weitere Möglichkeit, um Daten zwischen den beiden Applikationen auszutauschen ist die Daten direkt über eine Datenbank zu teilen. Der Datensammler könnte als ein Client des DBS die erhaltenen Daten in die Datenbank schreiben, der Webserver diese auslesen und löschen.

Getroffene Auswahl:

Nach Eruierung der erwähnten Möglichkeiten, wird die *bekannte Pipe* für die Kommunikation verwendet. Bei *Shared Memory* müsste ein Zugriffsverfahren implementiert werden, um nicht gleichzeitig schreibend und lesend auf Speicherbereiche zuzugreifen. Dieses Verfahren sind bei den anderen Techniken jedoch standardmäßig implementiert. Von der *Message Queue* wird Abstand genommen, da sie eher für Nachrichtenübertragung vorgesehen ist. Die *Socket*-Kommunikation wäre auch eine Möglichkeit, da sich die beiden Applikationen immer am selben Computer befinden, wäre diese Technik zu aufwendig für den Nutzen. Das Ablegen der Daten direkt in einer Datenbank und das spätere Lesen werden verworfen, da bei dieser Variante auch immer ein DBS installiert werden müsste, wodurch die Wartung und Installation des Systems komplizierter würde. Die *anonymen Pipes* scheiden aus, da bekannte Pipes mehr Funktionalitäten besitzen und jede Variable aufgrund ihres Namens und Namen des Data-Collector eindeutig im System referenziert werden kann. Abbildung 21 ist die Kommunikation zwischen Datensammler und Webserver dargestellt. Im Datensammler wird in jedem Objekt der Variable eine Pipe initialisiert. Der Name der Pipe setzt sich aus dem Präfix „VarPipe“, den Namen des Data-Collectors und dem Variablennamen zusammen. Damit sind die Pipes am Computer eindeutig referenzierbar und der Webserver besitzt dieselben Informationen, denn diese basieren ausschließlich auf Einträgen aus der Konfigurationsdatei *Conf_DataCollector*.

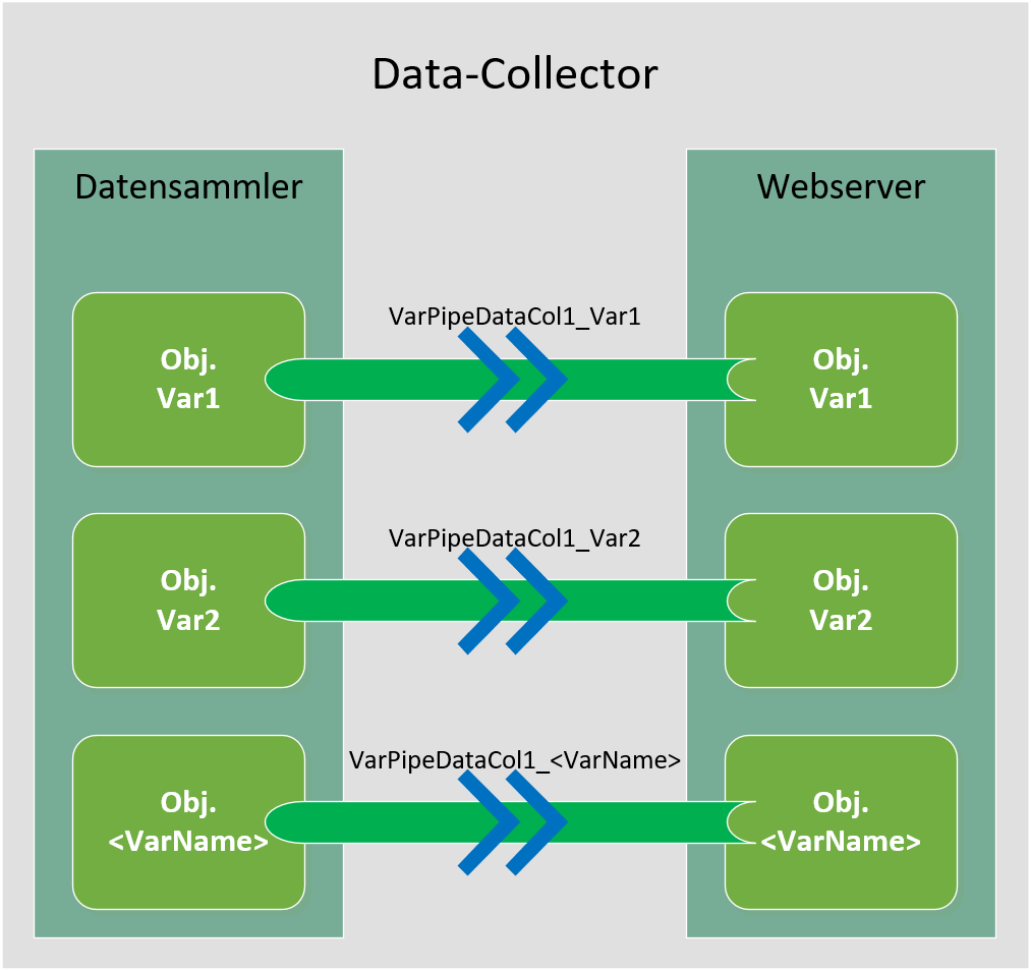

Abbildung 21: Kommunikation Datensammler und Webserver, Quelle: Eigene Darstellung.

9.3 Webserver (REST-API)

Bevor die Realisierung der einzelnen Protokolle behandelt wird, wird vorher der Webserver mit der REST-API erörtert. In der Arbeit wird nicht der Source-Code dargestellt, denn das würde den Umfang sprengen. Stattdessen werden die wichtigsten Funktionalitäten angeführt und prägnant zusammengefasst. Der generelle Aufbau wird bereits in Abbildung 20 dargestellt und anschließend kurz beschrieben.

Lesen der Konfigurationsdatei:

Für das Lesen der Konfigurationsdatei wird eine eigene Klasse erstellt, die diese Funktionalität beinhaltet. Sie soll generisch aufgebaut werden, da diese Funktionalität in allen Programmteilen benötigt wird. Der erste Schritt ist das Laden der Datei. Diese befindet sich, wie bereits erwähnt, immer im gleichen Verzeichnis wie die .exe-Datei und hat auch immer denselben Namen: „Conf_DataCollector“. Für das Lesen wird die C#-Bibliothek „System.Text.Json“ verwendet, welche Funktionalitäten für die Transformation von einer JSON-Datei in die C#-Struktur bietet. Für die Transformation wird eine Klasse angelegt, welche die gleiche Struktur wie die JSON-Datei besitzt. Ein Objekt dieser Klasse wird der Transformationsfunktion übergeben. Befinden sich die gleichen Bezeichnungen in JSON-Datei und Klasse, wird der Wert der JSON-Datei dem Wert des Objektes zugewiesen. Mit dieser Realisierung ist es möglich, Konfigurationsdateien zu verwenden, welche nicht für ein einzelnes Programm zugeschnitten sind.

Beim Lesen der Konfigurationsdatei ist der gesamte Dateninhalt der Konfigurationsdatei bereits geladen und deshalb wird hier der Hash-Wert der Datei berechnet. Für das Erstellen des Hash-Wertes wird die Bibliothek „System.Security.Crypography“ verwendet, welche verschiedene Klassen für die Kryptographie beinhaltet. Als Hash-Funktion wird SHA256 gewählt, wodurch auch in allen anderen Implementationen diese Hash-Funktion verwendet werden muss. Der Hash-Algorithmus gehört zur Familie der *Secure Hash Algorithm-2* (SHA) und wird vom *National Institute of Standards and Technology* (NIST, dem Herausgeber der SHA) empfohlen, denn der Vorgänger SHA-1 ist gebrochen worden.[34]

Konfiguration des Webservers:

Nach dem Lesen der Konfigurationsdatei wird der Webserver konfiguriert. Die wichtigste Einstellung, die vorgenommen wird, ist die Definition der *Uniform Resource Locators* (URL). Dabei werden das verwendete Protokoll, die IP-Adresse und der Port definiert. Bei diesem Webserver wird nur ein Zugang für Clients eingerichtet, weshalb auch nur eine URL definiert wird. Als Protokoll wird immer HTTPS definiert, wohingegen IP-Adresse und Port aus der Konfigurationsdatei gelesen werden. Eine Beispiel-URL, welche am Development Computer verwendet wird, lautet:

- https://192.168.0.200:5000

Mit dieser Adresse kann der Webserverdienst über das Netzwerk von Clients angesprochen werden.

Eine Anforderung an das System ist eine sichere Datenübertragung. Obwohl Data-Collector und Manager vorrangig in abgeschirmten Netzwerken betrieben werden sollen, muss die Kommunikation der beiden Applikationen verschlüsselt sein. Deshalb wird bei der Konfiguration des Webservers ein Zertifikat

[34] Vgl. National Institute of Standards and Technology (2015), Online-Quelle [16.10.2020].

ausgewählt, mit dem sich der Webserver authentifizieren und einen sicheren Übertragungskanal einrichten kann. Der Entwurf des Systems sieht vor, dass jeder Webserver das gleiche Zertifikat verwendet. Bei der Realisierung wird auf die Verwendung verschiedener Zertifikate, welche vom selben Root-Zertifikat unterzeichnet sind, verzichtet. Die Funktionalität ist komplett integriert und für verschiedene Zertifikate müssten nur andere Pfade in den Konfigurationsdateien angegeben werden.

Die Erstellung eines Zertifikats kann auf mehrere Arten erfolgen. Der beste Weg ist ein Zertifikat bei einer Zertifizierungsstelle zu erwerben. Dabei müssen Daten an die Zertifizierungsstelle übermittelt werden, welche in das Zertifikat einfließen und auf Richtigkeit überprüft werden. Danach wird das Zertifikat von der Zertifizierungsstelle mit ihrem Zertifikat zertifiziert, wodurch dieses Zertifikat von den Betriebssystemen als sicher eingestuft wird. Zertifikate für Unternehmen sind mit Kosten verbunden, jedoch gibt es auch Zertifizierungsstellen, wie *Let's Encrypt,* die zeitlich begrenzte Zertifikate kostenlos ausstellen. Es können auch selbst signierte Zertifikate lokal am Rechner erstellt werden. Hierfür gibt es eigene Programme, wie OpenSSL, die auch in Applikationen implementierbar sind. Microsoft bietet mit PowerShell eine Möglichkeit, ohne Installation Zertifikate zu erstellen.

Wie in Abbildung 22 ersichtlich, wird das Zertifikat mit der PowerShell erstellt. Dafür sind lediglich vier Eingaben notwendig und das Zertifikat ist erstellt und in einem Verzeichnis exportiert. Exportiert werden das Zertifikat als .crt-Datei und das passwortgeschützte Zertifikat mit Private-Key als .pfx-Datei. Bei der Erstellung des Zertifikats, mit dem ersten Befehl, können verschiedene Parameter übergeben werden, welche im Zertifikat angezeigt werden.

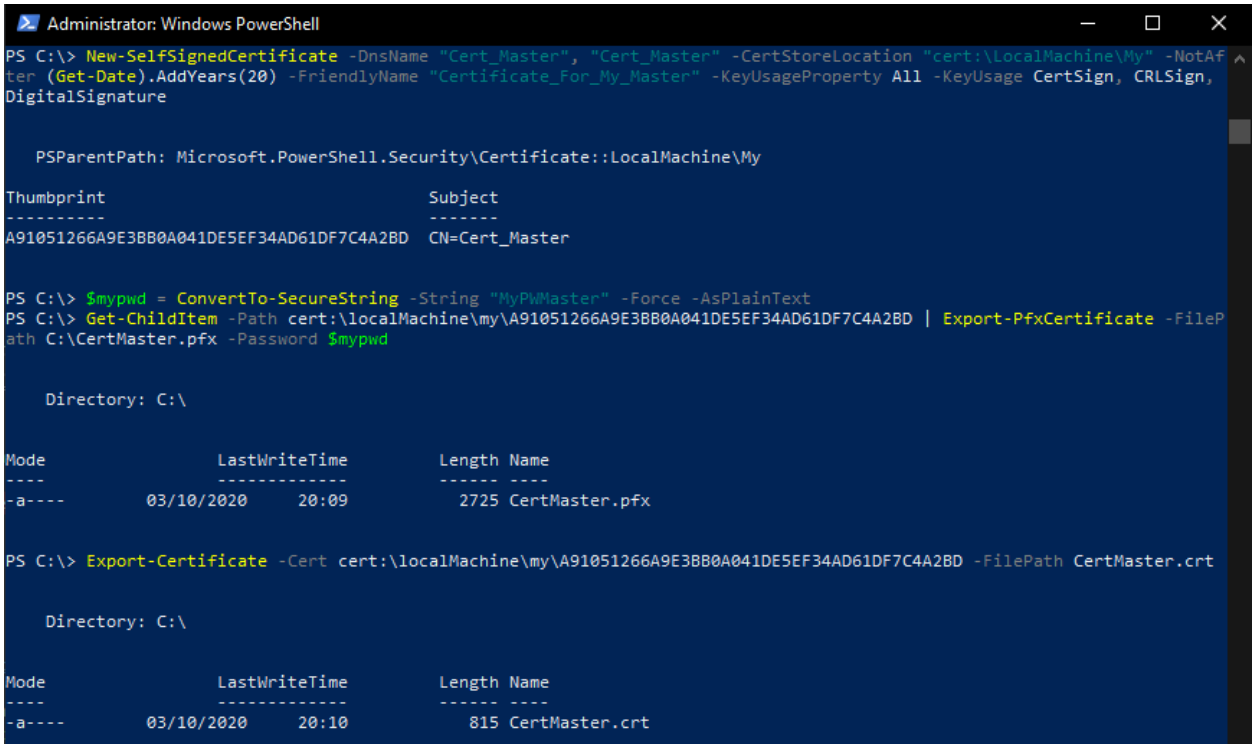

Abbildung 22: Erstellung des Zertifikats für Webserver, Quelle: Eigene Darstellung.

Abbildung 23 zeigt die erstellte .crt-Datei, wobei der Name ersichtlich ist. Des Weiteren ist aus der Abbildung ersichtlich, dass das Betriebssystem dem Zertifikat nicht vertraut. Grund dafür ist, dass das Zertifikat selbst signiert ist und es nicht in den „Trusted Root Certification Authorities" hinzugefügt ist.

Nach Erstellung der Zertifikatsdateien, wird dem Webserver die .pfx-Datei als zu verwendendes Zertifikat zugewiesen. Danach sendet der Webserver bei jedem Client-Request dieses Zertifikat als Authentifizierung mit. Der Client, in diesem Fall der Manager, würde ohne Zutun diese Verbindung nicht aufbauen, da das Zertifikat nicht vertrauenswürdig ist. Dafür wird implementiert, dass der Manager die .crt-Datei besitzt und dadurch eine Überprüfung durchführen kann, ob der Server tatsächlich den Privat-Key (.pfx-Datei) kennt.

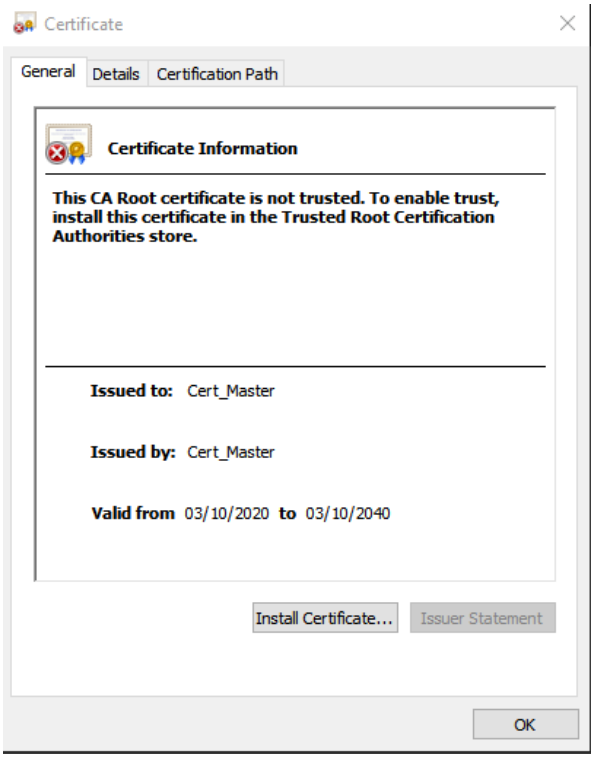

Abbildung 23: Zertifikat für Webserver, Quelle: Eigene Darstellung.

Hash-Wert prüfen:

Bevor der Webserver Daten vom Datensammler empfängt, wird geprüft, ob beide Applikationen die gleichen Konfigurationsdateien benutzen. Damit werden Inkonsistenzen zwischen den beiden Applikationen verhindert. Für die Prüfung wird eine eigene Klasse angelegt, in der die Funktionalitäten implementiert werden. Um die Hash-Werte der Konfigurationsdateien von Datensammler und Webserver zu vergleichen, muss zuerst eine Verbindung zwischen den beiden Applikationen aufgebaut werden. Dies geschieht, wie beschrieben, mit *benannten Pipes*. Der Pipe-Name setzt sich aus dem Präfix „HashPipe“ und dem Namen des Data-Collectors zusammen. Initiator der Pipe ist der Datensammler, welcher die Pipe erstellt und seinen Hash-Wert in die Pipe schreibt. Der Webserver wartet so lange, bis sich eine Nachricht in der Pipe befindet. Diese wird danach ausgelesen und mit dem eigenen Hash-Wert verglichen. Stimmen die beiden Werte nicht exakt überein, wird der Webserver geschlossen und ein Eintrag in der Log-Datei generiert. Wie bereits bei der Hash-Wert-Bildung beschrieben ist, wird die Hash-Funktion SHA256 verwendet.

Datenhandling:

Sind alle Vorbereitungen und Überprüfungen abgeschlossen, kann damit begonnen werden, Daten vom Datensammler zu empfangen und bis zur Abfrage abzuspeichern. Dafür wird eine Klasse entworfen, von der, für jede Variable, ein Objekt abgeleitet wird. Beim Erstellen der Objekte werden alle Eigenschaften bezüglich einer Variablen den einzelnen Objekten zugewiesen.

Für das Lesen der Werte aus der Pipe wird ein eigener Thread für jedes Objekt generiert. Diesem Thread wird eine Methode zugewiesen, welche ständig auf neue Einträge in der Pipe wartet. Ist ein neuer Eintrag vorhanden, muss dieser von einem Byte-Array in ein C#-Objekt umgewandelt werden, um wieder dasselbe Format wie im Data-Collector zu erhalten. Die daraus resultierende Messung (Messzeitpunkt und Messwert) wird in eine statische threadsichere Queue geschrieben.

Um die einzelnen Messungen aller Messpunkte zu speichern, wird zusätzlich ein statischer Thread gestartet. Diesem Thread wird eine Methode zugewiesen, welche dauernd auf Einträge in der Queue wartet. Beim Auftauchen eines neuen Eintrages wird dieser gelesen und aus der Queue entfernt. Anschließend wird die Messung mit Verweis auf die Herkunft (Variablenname) in eine Puffer-DB geschrieben.

Datenbanksystem:

Als DBS für die Speicherung der Daten wird SQLite verwendet. SQLite bietet den großen Vorteil, dass kein DBMS auf dem Betriebssystem installiert werden muss. Für die Verwendung wird nur eine Bibliothek benötigt, welche alle Funktionalitäten besitzt. Die Daten werden aber trotzdem als Binärdatei abgelegt. Um nicht SQL-Befehle nutzen zu müssen, wird in der Implementation zusätzlich das „Entity Framework“ verwendet. Mit diesem kann die Struktur der DB automatisch generiert werden und die Daten in der DB können wie eine Datensammlung verwendet werden.

In Abbildung 24 wird der schematische Ablauf grafisch dargestellt. Zuerst wird für jede Variable ein eigenes Objekt initialisiert. In jedem Objekt dieser Klasse wird in einem eigenen Thread auf neue Einträge in der Pipe vom Datensammler gehorcht. Ist ein neuer Eintrag vorhanden, wird dieser in die Queue geschrieben. Die Queue wird auch in einem eigenen Thread ausgewertet, und wenn ein neuer Eintrag vorhanden ist, in die Datenbank geschrieben.

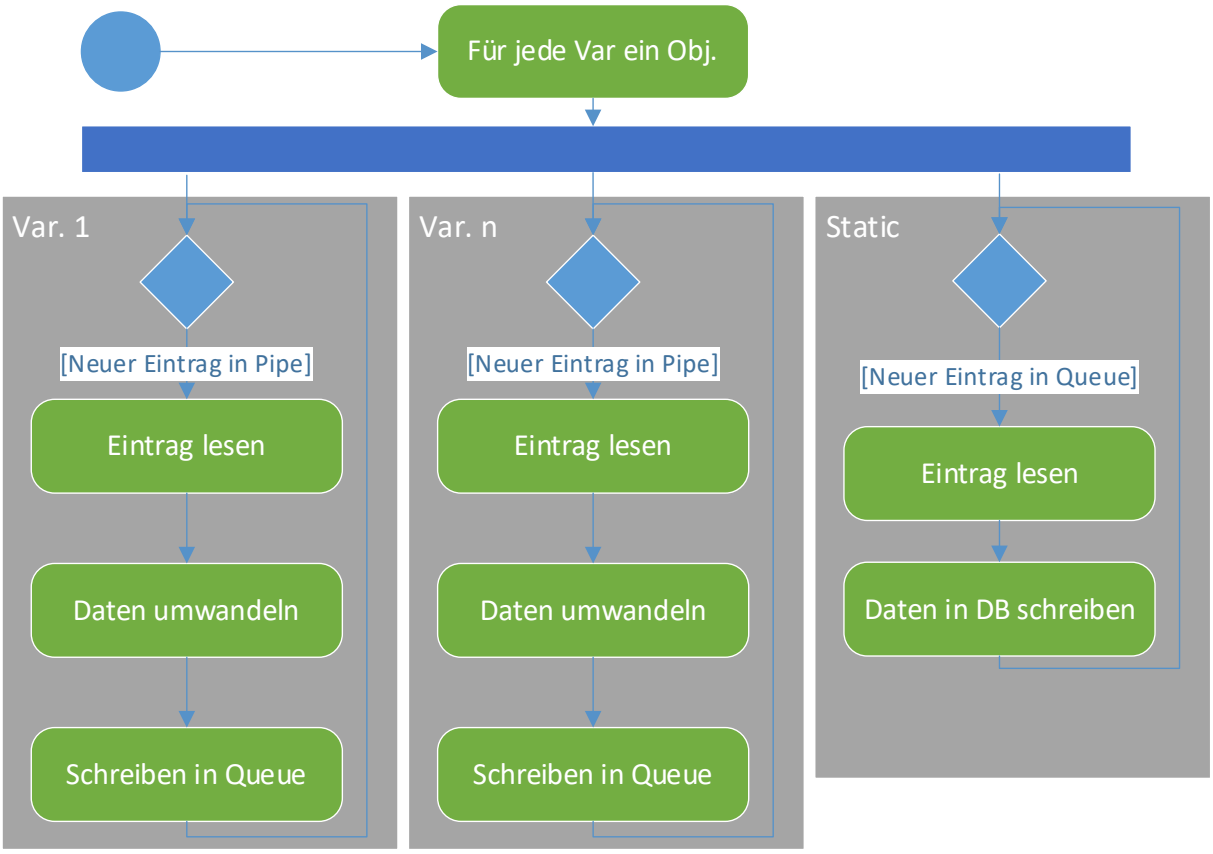

Abbildung 24: Ablauf Datenhandling, Quelle: Eigene Darstellung.

REST-API:

Der letzte Teil für die Realisierung des Webservers ist die REST-API. Durch die Verwendung von „ASP.NET Core“ wird die Implementierung der API erheblich vereinfacht. Für die Erstellung einer neuen URL muss ein neuer Controller in Visual Studio erstellt werden. Dabei wird der Name der zu erstellenden URL angegeben und das Grundgerüst wird automatisch generiert. Für die Kommunikation mit dem Manager werden zwei URLs erstellt:

- Hash-Wert
  Bei dieser URL kann der Hash-Wert der Konfigurationsdatei abgefragt werden. Ist der Webserver verfügbar und der Hash-Wert wird abgefragt, ist sichergestellt, dass Datensammler und Webserver die gleiche Konfigurationsdatei verwenden. Der Hash-Wert wird in einem JSON-Dictionary ausgegeben, welches als Key „HashKey“ beinhaltet und als Wert den Hash-Wert (Abbildung 25).
- Lesen von Messungen
  Um nicht für jede einzelne Variable eine eigene URL erstellen zu müssen, wird nur eine einzige erstellt, welche aber über einen Parameter „Variable“ verfügt. Will somit ein Client Daten einer Variable abfragen, sendet er einen HTTPS GET-Befehl und gibt dabei den Namen der Variable in der URL mit an. Der Name, der eingetragen werden muss, ist der gleiche, wie in der Konfigurationsdatei *Conf_DataCollector*. Trifft eine Abfrage am Webserver ein, liest er alle Messungen aus der Datenbank aus (außer es sind mehr als 1000, dann nur die ältesten 1000) und sendet diese an den Anfragesteller zurück. Vor dem Senden werden die Daten noch in das JSON-

Format konvertiert. Ist die Datenausgabe erfolgreich, werden die gesendeten Daten aus der Datenbank gelöscht. Das Rückgabeformat ist ein Strukturarray, welches die Messzeitpunkte und Messwerte beinhaltet (Abbildung 26).

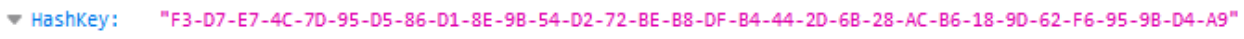

Abbildung 25: Beispiel: Hash-Wert über REST-API, Quelle: Eigene Darstellung.

Abbildung 26: Beispiel: Messungen einer Variablen über REST-API, Quelle: Eigene Darstellung.

9.4 Protokoll OPC UA

In diesem Abschnitt wird die Realisierung des Protokolls OPC UA behandelt. Um die Wartbarkeit und zukünftige Neuentwicklungen zu vereinfachen, werden die Funktionalitäten, bezüglich der Datenerfassung eines Protokolls, in einem eigenen Programm realisiert. Eine einheitliche Schnittstelle bezüglich der Kommunikation mit dem Webserver ist in diesem Kapitel bereits definiert.

Für die Datensammlung über OPC UA wird ein OPC UA Client implementiert. Dieser verbindet sich zu einem OPC UA Server und liest die gewünschten Datenpunkte aus. Von welchem OPC UA Server, welche Datenpunkte gelesen werden, ist in der Konfigurationsdatei *Conf_DataCollector* hinterlegt. Zur Implementierung des OPC UA Clients wird der OPC UA Stack von der OPC Foundation verwendet. Die Verwendung dieses Stacks steht unter einer Dualen-Lizenz. Für *OPC Foundation Corporate Member* unterstehen der Stack der RCL-Lizenz und für alle anderen der GPL 2.0.[35] Das bedeutet, dass Nicht-Mitglieder der *OPC Foundation Corporate Member* die entwickelte Software auch unter der GPL 2.0 stellen müssen. Der OPC UA Stack (OPCFoundation.NetStandard.Opc.Ua.Symbols Version: 1.4.363.49) wird mit dem NuGet-Package-Manager in der Applikation integriert.

Lesen der Konfigurationsdatei:

Der erste Schritt, der bei den Datensammlern ausgeführt wird, ist das Lesen der Konfigurationsdatei. Dieser Schritt läuft gleich ab wie beim Webserver, weshalb dieser hier nicht im Detail behandelt wird. Es werden

[35] Vgl. OPC Foundation (o. J.), Online-Quelle [11.10.2020].

jedoch nur die Einträge ausgelesen, welche für die Programmausführung benötigt werden. Einerseits sind das gleiche Einträge, welche auch der Webserver ausliest, andererseits aber auch Einträge, die speziell für die Funktionalität des OPC UA Clients benötigt werden.

Folgende Einträge werden aus der Konfigurationsdatei für den Datensammler (OPC UA) gelesen:

- **Namen des Data-Collectors**
  Zur eindeutigen Namensgebung der Pipes.
- **URL zu OPC UA Server**
  Adresse vom OPC UA Server, von dem die Daten gelesen werden.
- **Pfad der .exe-Datei des Webservers**
- **Name der Variablen**
  Eindeutige Namen pro Data-Collector.
- **Update Intervall der Variablen**
  Definiert, wie oft eine Variable vom Server gelesen werden soll.
- **Option für Verwendung eines Zertifikates für die Kommunikation mit OPC UA Server**
- **Adresse der Variablen am OPC UA Server**

Wie bereits beim Webserver wird beim Lesen der Konfigurationsdatei der Hash-Wert der Datei gebildet und gespeichert.

Webserver starten:

Nachdem der Pfad der .exe-Datei für den Webserver bekannt ist, wird der Webserver gestartet. Um den Datensammler nicht zu blockieren, wird die .exe-Datei in einem separaten Prozess gestartet.

Hash-Wert prüfen:

Ist der Webserver gestartet, erfolgt der Abgleich des erstellten Hash-Wertes mit dem Hash-Wert des Webservers. Der grundlegende Ablauf ist bereits beim Webserver beschrieben. Es wird im Datensammler eine Klasse entworfen, welche eine Pipe öffnet. Diese setzt sich aus dem Präfix „HashPipe“ und den Namen des Data-Collectors zusammen. Nachdem die Pipe geöffnet ist, schreibt der Datensammler den Hash-Wert in die Pipe. Der Datensammler erhält keine Rückmeldung, ob der Webserver die gleiche Konfigurationsdatei enthält. Die Programmausführung wird somit fortgeführt, obwohl der Webserver sich selbst beenden könnte. Dieses Konzept wird gewählt, um, trotz fehlendem Webserver, Funktionalitäten des Datensammlers prüfen zu können.

Klasse für Variablenfunktionalitäten:

Methoden und Eigenschaften, welche auf jede Variable in der gleichen Art und Weise angewendet werden, sind in einer Klasse definiert. Nach Prüfung des Hash-Wertes wird somit für jede Variable ein eigenes Objekt instanziiert. Dieses Konzept soll bei allen Protokollen angewendet werden.

Eine Methode, welche immer gleich implementiert werden muss, ist die Kommunikation mit dem Webserver. Diese Methode wird in der Klasse einem eigenen Thread zugewiesen. Die Methode öffnet bei Aufruf eine Pipe, welche sich aus dem Präfix „VarPipe“, dem Namen des Data-Collectors und dem Namen der Variablen zusammensetzt. Nach erfolgreicher Generierung der Pipe wird darauf gewartet, bis in einer

threadsicheren Queue eine neue Messung (Messwert und Messzeitpunkt) geschrieben wird. Diese Messung wird ausgelesen und über die Pipe an den Webserver gesendet. Die Realisierung, wie die Messungen in die Queue kommen, kann bei jedem Protokoll unterschiedlich sein.

OPC UA spezifisch wird in der Klasse eine Methode implementiert, die dem OPC UA Client zugewiesen wird. Dieser ruft bei erhalt einer neuen Messung einer Variablen, diese Methode auf. Die Methode erhält vom OPC UA Client als Übergabeparameter unter anderem den Messwert und den Messzeitpunkt, welche sofort in die threadsichere Queue geschrieben werden.

OPC UA Client:

Alle Funktionalitäten bezüglich des OPC UA Clients sind in dieser Klasse zusammengefasst. Zur Konfiguration des OPC UA Clients gibt es zwei unterschiedliche Methoden. Die OPC UA Bibliothek entnimmt einer XML-Datei, welche bei der Implementierung der Bibliothek erstellt wird, OPC UA spezifische Einstellungen. So ist darin definiert, welches Zertifikat der Client für die Kommunikation mit dem OPC UA Server benutzt. Des Weiteren werden die Verzeichnisse definiert, wo sich die akzeptierten und abgelehnten Zertifikate der OPC UA Server befinden. Generelle Einstellungen, wie z.B. mit welchem OPC UA Server sich der Client verbinden soll, werden dem Objekt bei der Instanziierung übergeben und stammen aus der Konfigurationsdatei.

Der Verbindungsaufbau mit dem OPC UA Server erfolgt in folgenden Schritten:

1. Eine neue Instanz eines OPC UA Clients wird generiert. Dazu wird die XML-Datei, welche sich im selben Verzeichnis wie die .exe-Datei befinden muss, ausgelesen und deren Einstellungen abgespeichert.
2. Das auf in der XML-Datei referenzierte Zertifikat des Clients wird geladen. Ist dieses Zertifikat nicht verfügbar, wird ein neues, mit den definierten Eigenschaften, erzeugt und als vertrauenswürdig gekennzeichnet. Des Weiteren ist auch möglich, in der Konfigurationsdatei *Conf_DataCollector* den Eintrag *UseCertificateOPCUA* auf *false* zu setzen. Dann wird der Client versuchen, eine Verbindung ohne Sicherheitsmaßnahmen aufzubauen. Dies sollte aber nur für Testzwecke benutzt werden.
3. Der Client fragt bei dem in der Konfigurationsdatei *Conf_DataCollector* definierten OPC UA Server die verfügbaren Endpoints an. Diese beschreiben welche Sicherheitspolicies der Server unterstützt. Zusätzlich wird das Zertifikat des OPC UA Servers dem Client übermittelt.
4. Es wird überprüft, ob dem Zertifikat des OPC UA Servers vertraut wird. Einem Zertifikat wird vertraut, wenn es sich im *trusted*-Ordner des OPC UA Clients befindet. Soll der Client dem Server automatisch vertrauen, kann in der XML-Datei der Eintrag *autotrust* auf *true* gesetzt werden. Dann wird beim erstmaligen Verbindungsaufbau das Zertifikat automatisch in den *trusted*-Ordner kopiert.
5. Wird dem Server vertraut, wählt der Client eine Sicherheitspolicy aus. Unterstützen Server und Client nicht eine gemeinsame Sicherheitspolicy, kann keine Verbindung aufgebaut werden.
6. Nach Auswahl der Sicherheitspolicy versucht der Client mit dem Server eine Session aufzubauen und sendet sein Zertifikat mit. Der Server lässt nur eine Session zu, wenn der Server dem Zertifikat vertraut. Dies kann über Kopieren von Zertifikaten in einen *trusted*-Ordner passieren oder der

Server vertraut einem Root-Zertifikat, mit dem das Zertifikat unterschrieben ist. Dieser Schritt kann aber entfallen, wenn der Server eine Session ohne jegliche Sicherheitsmechanismen zulässt.

Sind all diese Schritte erfolgreich durchlaufen worden, besteht zwischen Server und Client eine aktive Session.

Um nun Daten vom Server lesen zu können, wird für jedes Zeitintervall eine eigene Subskription erstellt. In jeder Subskription wird für jede Variable, welche das Zeitintervall besitzt, eine neue Abfrage angelegt. Dieser Abfrage wird der Name der Variable, die Adresse und die Methode, welche im Objekt der Variable definiert ist (schreiben in die threadsichere Queue), zugewiesen. Danach werden alle Subskriptionen auf die aktive Session verwiesen.

Die Subskriptionen werden gemäß ihrer Zeitintervalle zyklisch aufgerufen. Dabei werden die Werte, welche die Subskription beinhalten, aktualisiert. Des Weiteren wird bei Aktualisierung eines Wertes die Methode aufgerufen, auf die referenziert wird. In dieser Methode wird die Messung (Messwert und Messzeitpunkt) in die Queue geschrieben, aus der die Messung an den Webserver gesendet wird (wie oben beschrieben).

In Abbildung 27 ist ein Auszug der Log-Datei dargestellt. Daraus ist der gesamte Ablauf der Applikation bis zum Sammeln der Daten ersichtlich. Bei diesem Snapshot werden drei Variablen von einem Simulations-OPC UA Server gelesen, wobei dafür zwei Subskriptionen angelegt sind. Es ist auch ersichtlich, dass Zertifikate für die Kommunikation verwendet werden und der Endpoint *Basic256* angewendet wird. Am Ende des Eintrages sind die erfassten Daten mit den unterschiedlichen Zeitstempel ersichtlich.

```
22:30:58  Application Started                                                                                    Info
22:30:58  Begin Read Json                                                                                        Info
22:30:58  Hash= 36-E2-0A-49-E5-E7-1D-26-0C-66-A9-D6-0A-FB-98-43-A8-43-86-60-2D-93-00-88-F9-34-BE-C2-14-59-BD-26 Info
22:30:58  UrlOPCServer = opc.tcp://Bernhard-Surface:26543/OPCUA/SimulationServer                                 Info
22:30:58  Name = DataCollector1                                                                                  Info
22:30:58  UseCertificateOPCUA = True                                                                            Info
22:30:58  Found Variables = 3                                                                                    Info
22:30:58  VarName1 - int - 1000 - ns=5;s=Sinusoid1                                                               Info
22:30:58  VarName2 - int - 2000 - ns=5;s=Counter1                                                                Info
22:30:58  VarName3 - int - 1000 - ns=5;s=Random1                                                                 Info
22:30:58  Start WebAPI                                                                                           Info
22:30:59  CreatePipe: VarPipeDataCollector1_VarName1                                                             Debug
22:30:59  CreatePipe: VarPipeDataCollector1_VarName2                                                             Debug
22:30:59  Start OPC Client                                                                                       Info
22:30:59  CreatePipe: VarPipeDataCollector1_VarName3                                                             Debug
22:31:00  1 - Create an Application Configuration.                                                               Info
22:31:04  2 - Discover endpoints of: opc.tcp://Bernhard-Surface:26543/OPCUA/SimulationServer                     Info
22:31:05      Selected endpoint uses: Basic256                                                                   Info
22:31:05  3 - Create a session with OPC UA server.                                                               Info
22:31:06  5 - Create a subscription with publishing interval of 1000 ms                                          Info
22:31:06  6 - Add a list of items to the subscription.                                                           Info
22:31:06  7 - Add the subscription to the session.                                                               Info
22:31:06  5 - Create a subscription with publishing interval of 2000 ms                                          Info
22:31:06  6 - Add a list of items to the subscription.                                                           Info
22:31:06  7 - Add the subscription to the session.                                                               Info
22:31:07  VarName1: 1.9021130047561248, 20:31:06, Good                                                          Info
22:31:07  VarName3: 0.8533842470358814, 20:31:06, Good                                                          Info
22:31:08  VarName1: 1.989043782539744, 20:31:07, Good                                                           Info
22:31:08  VarName3: 0.10779522840046163, 20:31:07, Good                                                         Info
22:31:08  VarName2: 89, 20:31:08, Good                                                                          Info
22:31:09  VarName1: 1.9890437977149242, 20:31:08, Good                                                          Info
22:31:09  VarName3: 0.01534730142092755, 20:31:08, Good                                                         Info
22:31:10  VarName1: 1.902113049618437, 20:31:09, Good                                                           Info
22:31:10  VarName3: 0.2926910972606911, 20:31:09, Good                                                          Info
22:31:10  VarName2: 91, 20:31:10, Good                                                                          Info
22:31:11  VarName1: 1.7320508292927856, 20:31:10, Good                                                          Info
22:31:11  VarName3: 0.20597637389995282, 20:31:10, Good                                                         Info
22:31:12  VarName1: 1.4862896722274184, 20:31:11, Good                                                          Info
22:31:12  VarName3: 0.015619611194311434, 20:31:11, Good                                                        Info
```

Abbildung 27: Auszug Log-Datei OPC UA Client, Quelle: Eigene Darstellung.

9.5 Protokoll TCP/IP

Als zweites Protokoll wird TCP/IP integriert. Diese Integration fußt auf dem TCP-Protokoll, welches sich auf OSI Layer vier befindet und ein reines Datenübertragungsprotokoll ist. Da TCP keine Möglichkeit für die Repräsentation der Daten vorsieht, muss die Integration in höheren Ebenen geschehen. Meist setzen andere Protokolle wie OPC UA, HTTP oder FTP darauf auf, aber in dieser Realisierung werden die Datenpakete durch die Konfigurationsdatei definiert, wodurch beide Kommunikationspartner die genaue Bitposition jeder Variable wissen. Beim Datenaustausch über TCP/IP sendet das Gerät, welches die Daten an den Data-Collector liefert, kontinuierlich Telegramme mit den verschiedenen Messwerten.

Für den Datenaustausch wird definiert, dass jede Messung immer denselben Platz im Telegramm bekommt und nur der Messwert übertragen wird. Der Messzeitpunkt wird vorerst vernachlässigt, denn aufgrund der kurzen Übertragungszeit kann die Zeit auch vom Data-Collector hinzugefügt werden. Welchen Platz eine Variable bekommt, muss in der Konfigurationsdatei mittels Nummer definiert werden. Pro Messung werden im Telegramm immer vier Bytes reserviert, wobei die nicht genutzten Bits vernachlässigt werden. Um den Data-Collector den Datentyp der verwendeten Variablen mitzuteilen, gibt es dafür einen Eintrag in der Konfigurationsdatei. Des Weiteren wird definiert, dass die ersten zehn Bytes für etwaige zukünftige Konfigurationen frei bleiben und die Messwerte somit erst ab Byte zehn beginnen. Abbildung 28 zeigt wie vier Messwerte im Big-Endian-Format auf das Byte-Array geschrieben sind. Hat ein Messwert vier Bytes, werden alle vier Bytes beschrieben. Ist der Messwert ein zwei-Byte-Integer, dann wird der Messwert den hinteren beiden Bytes zugewiesen und bei einem Bool nur das letzte Bit. Um das Little-Endian-Format zu nutzen, muss in der Konfigurationsdatei der Parameter *ByteSwapping* auf *false* gesetzt werden.

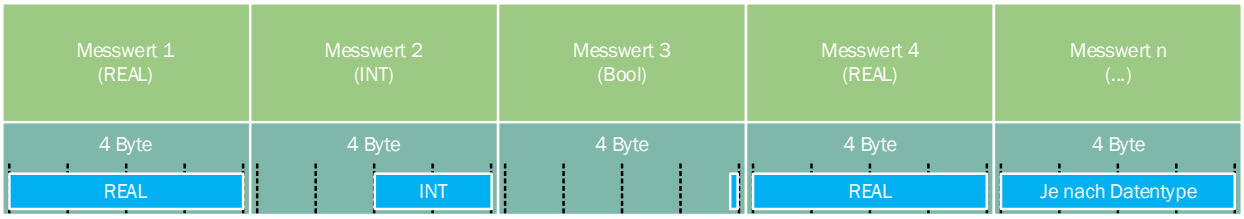

Abbildung 28: Data-Collector TCP, Interpretation Messwerte, Quelle: Eigene Darstellung.

Nun folgt eine Beschreibung der einzelnen Programmteile, wobei oft auf die Implementierung von OPC UA referenziert wird. Denn viele Teile sind ident aufgebaut.

Lesen der Konfigurationsdatei:

Das Lesen und Bilden des Hash-Wertes der Konfigurationsdatei *Conf_DataCollector* ist ident aufgebaut, wie bei OPC UA. Es werden nur unterschiedliche Daten ausgelesen, weil für das Protokoll TCP/IP andere Parameter benötigt werden:

- **Namen des Data-Collectors**
  Zur eindeutigen Namensgebung der Pipes.
- **IP-Adresse für Datenerhalt**
  Mit diese IP-Adresse horcht der Data-Collector auf das Eintreffen neuer Telegramme.

- **Port Nummer für Datenerhalt**
  Auf diese Port-Nummer muss das Gerät, von dem die Daten gesammelt werden, die Telegramme senden. Der Data-Collector hört nur auf diesen Port.
- **Pfad der .exe-Datei des Webservers**
- **Name der Variablen**
  Eindeutige Namen pro Data-Collector.
- **Update Intervall der Variablen**
  Definiert, in welchem Zeitabstand eine Variable vom Telegramm gespeichert werden soll
- **Position der Variable im Telegramm**

Webserver starten:

Nachdem der Pfad der .exe-Datei für den Webserver bekannt ist, wird dieser gestartet. Um den Datensammler nicht zu blockieren, wird die .exe-Datei in einem separaten Prozess gestartet.

Hash-Wert prüfen:

Der Hash-Wert wird gleich geprüft wie beim OPC UA Client. Es wird eine Pipe geöffnet und an den Webserver gesendet. Dieser vergleicht den Hash-Wert der eigenen Konfigurationsdatei mit dem erhaltenen Wert. Gibt es einen Unterschied, wird der Webserver beendet.

Klasse für Variablenfunktionalitäten:

Auch bei diesem Protokoll gibt es eine Klasse für die Funktionalitäten, die alle Variablen betreffen. Die Methode für den Datenaustausch (Pipe) ist ident, wie bei OPC UA. Aus einer threadsicheren Queue werden die Messungen gelesen und an den Webserver gesendet.

Für das Schreiben der Messungen in die Queue wird eine eigene Methode entwickelt. Diese wird aufgerufen, wenn ein Messwert mit dem Telegramm empfangen wird. Als Parameter wird der Messwert als Byte-Array übergeben. In der Methode wird als Erstes, falls notwendig, der Byte-Swap angewandt. Danach wird der Datentyp über den Parameter *Datatype* in der Konfigurationsdatei bestimmt. Aufgrund des Datentyps erfolgt dann die dezidierte Umwandlung des Byte-Arrays in das Datenformat *double.* Des Weiteren wird noch vor dem Schreiben in die Queue überprüft, ob seit dem letzten Schreiben in die Queue das definierte Zeitintervall abgelaufen ist. Ist das Zeitintervall noch nicht überschritten, wird der Wert zwischengespeichert, um bei Empfang des nächsten Wertes zu prüfen, ob das Zeitintervall dann abgelaufen ist. Ist das Zeitintervall dann noch immer nicht abgelaufen, wird der alte Wert verworfen und der neue Wert zwischengespeichert. Erst bei Überschreiten des Zeitintervalls, wird der letzte Wert in die Queue geschrieben. In Abbildung 29 ist der schematische Ablauf der Funktion dargestellt.

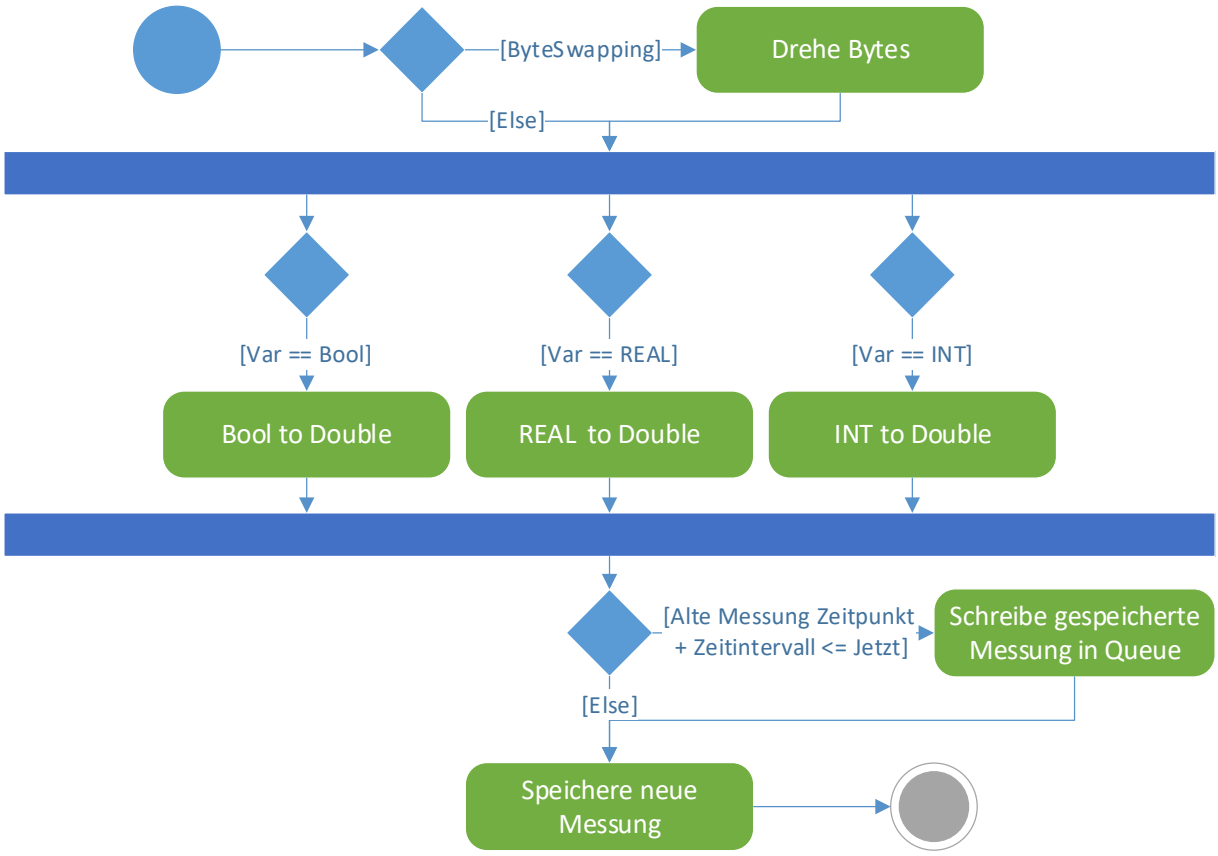

Abbildung 29: Data-Collector TCP, Ablauf Speicherung einer Messung, Quelle: Eigene Darstellung.

Empfangen der Telegramme:

Für das Empfangen der Telegramme wird eine eigene Klasse entwickelt und es wird die C#-Bibliothek „System.Net.Sockets“ importiert. Diese wird instanziiert, nachdem alle anderen Konfigurationen abgeschlossen sind. Bevor die Telegramme empfangen werden können, muss ein TCP-Listener konfiguriert werden, wobei IP-Adresse und Port-Nummer, aus der Konfigurationsdatei, zugewiesen werden. Nach dem Starten des TCP-Listener wartet dieser auf das Eintreffen von Telegrammen auf der zugewiesenen Schnittstelle und Port. Trifft ein neues Telegramm ein, geschieht folgendes:

- Die ersten zehn Bytes werden übersprungen.
- Die restlichen Bytes werden in vierer Schritten aufgerufen.
- Gibt es zu einer Vierergruppe (Bytes) einen Eintrag mittels Position in der Konfigurationsdatei, werden die Bytes der Methode für das Schreiben in die Queue übergeben.
- Ist das gesamte Telegramm durchforstet, wartet der TCP-Listener auf ein neues Telegramm.

Abbildung 30 zeigt den Start des Datensammlers TCP/IP. Bei dieser Konstellation sind sechs Variablen mit unterschiedlichen Updateintervallen definiert. Das Device sendet alle zwei Sekunden ein neues Telegramm mit den Werten. Aus der Log-Datei ist ersichtlich, dass nicht jede Messung an den Webserver geschrieben wird, sondern nur nach Ablauf des Zeitintervalls.

```
00:45:52  Application Started                                                    Info
00:45:52  Json Hash = 74-B3-73-01-AC-EF-88-B1-34-82-22-40-F6-FA-47-46-09-5B-
                                                  75-68-A9-51-FE-35-D8-88-0A-8A-CA-A0-A2-97 Info
00:45:52  Port_ForCollection = 8500                                             Info
00:45:52  ByteSwapping = True                                                  Info
00:45:52  IPAddress_ReceiveData = 192.168.1.8                                  Info
00:45:52  Name = DataCollector_TCP1                                            Info
00:45:52  Found Variables = 6                                                  Info
00:45:52  VarName1 - int - 2000 - 1                                            Info
00:45:52  VarName2 - int - 4000 - 2                                            Info
00:45:52  VarName3 - reAl - 1500 - 5                                           Info
00:45:52  VarName4 - bool - 3000 - 4                                           Info
00:45:52  VarName10 - Real - 10000 - 8                                         Info
00:45:52  VarName8 - int - 500 - 7                                             Info
00:45:52  Start WebAPI                                                         Info
00:45:54  Start listening                                                      Info
00:45:54  Config TCP Listener                                                  Info
00:45:56  Connected!                                                           Info
00:45:56  Received New Message.                                                Info
00:45:56  Write to Pipe: VarName1 - 00:45:54 - 22827                           Debug
00:45:56  Write to Pipe: VarName4 - 00:45:54 - 1                               Debug
00:45:56  Write to Pipe: VarName3 - 00:45:54 - 37793136902144                  Debug
00:45:56  Write to Pipe: VarName10 - 00:45:54 - 2                              Debug
00:45:56  Write to Pipe: VarName8 - 00:45:54 - 100                             Debug
00:45:56  Write to Pipe: VarName2 - 00:45:54 - -22702                          Debug
00:45:58  Connected!                                                           Info
00:45:58  Received New Message.                                                Info
00:45:58  Waiting for a connection...                                          Info
00:45:58  Write to Pipe: VarName8 - 00:45:56 - 201                             Debug
00:45:58  Write to Pipe: VarName3 - 00:45:56 - 46110332682240                  Debug
00:46:00  Connected!                                                           Info
00:46:00  Received New Message.                                                Info
00:46:00  Waiting for a connection...                                          Info
00:46:00  Write to Pipe: VarName4 - 00:45:58 - 3                               Debug
00:46:00  Write to Pipe: VarName1 - 00:45:58 - 23226                           Debug
00:46:00  Write to Pipe: VarName3 - 00:45:58 - 56314168344576                  Debug
00:46:00  Write to Pipe: VarName8 - 00:45:58 - 302                             Debug
```

Abbildung 30: Auszug Log-Datei TCP/IP, Quelle: Eigene Darstellung.

10 DB-CONNECTORS

In diesem Kapitel werden die Datenbank-Connectors realisiert. Es werden anfangs zwei DB-Connectors ausgewählt und diese anschließend in eine DLL implementiert. Die DB-Connectors erhalten die Daten vom Manager und übergeben diese weiter an die jeweilige Online-Datenbank. In Kapitel 8.4.2 wird bereits festgelegt, dass die DB-Connectors als DLL ausgeführt werden und welche Methoden für die Kommunikation implementiert werden müssen.

10.1 Auswahl Datenbanken

Im Zuge dieser Arbeit werden zwei Online-Datenbanken in das System integriert, welche nun in diesem Abschnitt ausgewählt werden. In Kapitel 4.1 werden bereits die infrage kommenden Datenbankmodelle erläutert. Wie aus jenem Kapitel hervorgeht, eignen sich relationale Datenbanken eher weniger für die Speicherung für Zeitreihen. Bei Relationen zwischen den einzelnen Tabellen benötigen größere Abfragen mehr Zeit. Objektrelationale Datenbanken können hingegen bereits Arrays speichern, weshalb diese schon besser für das Speichern von Zeitreihen infrage kommen. Bei dokument-orientierten Datenbanken werden die Datensätze in Dokumenten gespeichert, welche mit einem eindeutigen Schlüssel referenziert werden. Diese Architektur eignet sich auch nicht perfekt zum Speichern von zeitabhängigen Werten. Als letzte Datenbank wird das Time-Series-Datenbankmodell untersucht. Dieses Modell eignet sich perfekt für diese Aufgabe, da dieses Modell speziell dafür entwickelt wurde. Bei diesen Datenbanken werden die Daten nach der Zeit sortiert abgelegt, sodass Abfragen über die Zeit schneller durchgeführt werden können.

Nach Evaluierung, der verschiedenen Datenbankmodelle, wird eine Datenbank als *Time-Series-Datenbankmodell* realisiert. Dafür wird die in Kapitel 4.1.4 beschriebene Datenbank *InfluxDB* näher betrachtet, jedoch auch die verschiedenen Online-Datenbanken von *Microsoft-Azure*, *Google-Cloud* und *Amazon-Web-Services*. Microsoft-Azure bietet mit Azure-Time-Series-Insights und der Erweiterung *Timescale DB* für *PostgreSQL* zwei Möglichkeiten für Zeitreihenverarbeitung. Google bietet noch keine dezidierte Time-Series-Datenbank und setzt dabei auf NoSQL-Lösungen wie der Datenbank *Firestore*, *Firebase* oder *Bigtable*. Bei diesen Datenbanken fehlen jedoch Möglichkeiten für die Zeitreihenanalyse.[36] Amazon-Web-Services bietet mit der *Amazon-Timestream* eine Datenbank für das Speichern von Zeitreihen.

Für die Time-Series-Datenbank wird die **InfluxDB** realisiert. Sie bietet eine einfache Umsetzung, ist die am weitesten verbreitete Time-Series-Datenbank laut dem Ranking in Kapitel 4.2 und bietet einen freien Testaccount mit folgenden Merkmalen:

- 30 Tage Datenspeicher
- 1000 kb/s Lesen
- 17 kb/s Schreiben
- 10.000 gespeicherte Messungen

Für diesen Account muss auch keine Kreditkarte angegeben werden.

[36] Vgl. Stauffer (2020), Online-Quelle [03.11.2020].

Da Time-Series-Datenbanken noch nicht so weit verbreitet sind und um einen Vergleich mit einem anderen Datenbankmodell zu erhalten, wird eine Datenbank nicht als Time-Series-Datenbank realisiert. Stattessen wird eine Wide-Column-Datenbank realisiert, in der in einem Table verschiedene Datentypen pro Zelle erlaubt sind. Im Falle dieses Projektes ist das sinnvoll, denn es werden verschiedene Datentypen gesammelt, z.B. Bool, Int oder Real. Damit kann der Speicherbedarf reduziert werden. Mit wenigen Adaptionen könnte dieses Schema aber auch auf relationale Datenbanken angewendet werden. Für die Auswahl werden die großen Datenbankhoster (Google, Microsoft und Amazon) betrachtet. Microsoft Azure bietet dafür zwei unterschiedliche Datenbanken an: *Table Storage* und *Cosmos DB*. Google-Cloud und Amazon-Web-Services bieten hingegen nur eine Datenbank: *Bigtable* (Google) und *DynamoDB* (Amazon).

Für die Realisierung wird die **Microsoft Azure Cosmos DB** gewählt. Ausschlaggebend dafür ist, dass es viel Beispielcode für die Implementierung gibt. Microsoft Azure bietet auch eine Vielzahl von weiteren Online-Datenbanken, weshalb sich die Auseinandersetzung mit dieser Plattform auszahlt. Sollte in Zukunft ein anderes Datenbankmodell entwickelt werden, gibt es bei Microsoft Azure mit hoher Wahrscheinlichkeit einen Vertreter davon. Auch bei Azure gibt es für eine begrenzte Zeit Möglichkeiten für die kostenlose Evaluierung der Online-Services, jedoch müssen dafür Kontodaten hinterlegt werden.

10.2 Generelle Architektur

Da mehrere DB-Connectors entwickelt werden, muss zuerst eine generelle Architektur definiert werden. Die generelle Architektur soll bei allen Data-Connectors gleich aufgebaut sein, um die Wartung und zukünftige Entwicklungen zu erleichtern. Wie bereits definiert ist, werden die DB-Connectors als DLL ausgeführt. Dazu werden in Visual Studio 2019 neue „.Net Core 3.1“ Projekte angelegt und anschließend in den Einstellungen auf *Class Library* umgestellt. Dadurch wird beim Kompilieren keine .exe-Datei erstellt, sondern eine DLL.

Definitionen:

Da Messwerte über eine Methode übergeben werden, wird definiert, dass jede Variable eine eindeutige ID bekommt. Das soll dazu führen, dass nicht bei jedem Messwert sämtliche Informationen des Data-Collectors mitübergeben werden müssen. Der Manager soll bei der Initialisierung für jede Variable eine ID vergeben und diese über eine separate Methode den DB-Connectors mitteilen. Somit wird bei der Übermittlung der Messwerte Rechenleistung gespart, denn es wird nur ein UINT pro Messung mitgesendet, statt zwei Strings (Data-Collector und Variablenname).

Um bei Ausfall der Internetverbindung keine Daten zu verlieren, wird auch eine Datenbank für das Zwischenspeichern der Daten implementiert. Diese soll alle empfangenen Daten des Managers zuerst speichern und erst nach erfolgreichem Transferieren an die Online-Datenbank löschen. Somit wird eine hohe Datensicherheit gewährleistet.

Um neue Implementierungen zu erleichtern, wird eine Basisklasse für die Funktionalitäten der verschiedenen Online-Datenbanken errichtet. Diese soll alle Grundfunktionalitäten beinhalten, welche für den Erhalt der Daten aus der Puffer-Datenbank erforderlich sind oder andere Methoden, welche immer ident sind. Diese Klasse soll immer als Basisklasse für jene Klasse dienen, welche die Funktionalitäten für das Übermitteln der Daten in die Online-Datenbanken beinhaltet.

Klassen:

Im Data-Connector sind immer die folgenden Klassen vorhanden:

- **Interface**
  In der Klasse Interface sind alle Methoden hinterlegt, welche für die Kommunikation mit dem Manager benötigt werden. Die Klasse wird vom Manager instanziiert und es werden die jeweiligen Methoden aufgerufen.
- **ReadJson_DBConnector**
  In dieser Klasse werden die Einträge der Konfigurationsdatei *Conf_DBConnector* gelesen. Welche Datei gelesen wird, wird bei der Initialisierung der Klasse festgelegt. Da unterschiedliche DB-Connectors unterschiedliche Parameter benötigen, muss diese Klasse bei jedem DB-Connector adaptiert werden. Die Klasse ist im Grunde gleich aufgebaut, wie die Klasse für das Lesen der Konfigurationsdatei für den Data-Collector. Für das Transformieren der JSON-Konfigurationsdatei wird die C#-Bibliothek „System.Text.Json“ verwendet und der Hash-Algorithmus ist SHA256.
- **VariableList**
  Die Klasse *VariableList* ist eine Sammlung von allen Datenpunkten, die der jeweilige DB-Connector an die Online-Datenbank senden soll. In ihr befindet sich die Zuweisung zwischen dem eindeutigen Index der Messung und Eigenschaften der Messung (Data-Collector Name und Tag Name). Der Inhalt der Liste wird bei Aufruf der Methode *GetIdxOfVars(…)* vom Manager gefüllt. Dieser generiert die ID und übergibt die dazugehörigen Parameter (Data-Collector Name und Tag Name). Beim Einfügen der Einträge in die Liste wird zusätzlich überprüft, ob diese Einträge auch in der Konfigurationsdatei *Conf_DBConnector* vorhanden sind.
- **BufferDB**
  Diese Klasse beinhaltet alle Methoden für das Zwischenspeichern der Messungen. Dafür wird, wie auch beim Datensammler, die SQLite Datenbank, mit der Bibliothek „*System.Data.SQLite“*, verwendet. Der Pfad für die Datenbankdatei wird mit der Initialisierungsmethode *InitCheck(…)* übergeben. Dies ist nötig, da der Manager eine DLL öfter laden kann und dadurch kein statischer Pfad realisiert werden kann. Die Datenbank enthält für jede Messung einen eigenen Table, welcher angelegt wird (wenn nicht vorhanden), wenn auch die Klasse *VariableList* befüllt wird. Der Name der Tables setzt sich aus dem Data-Collector Namen und dem Tag Namen zusammen und ist somit im gesamten System eindeutig. Befüllt wird die *BufferDB* mit der Schnittstellenmethode *ReceiveValues(…)*. Diese wird vom Manager aufgerufen und übergibt die ID, Zeitpunkt der Messung und den Messwert. Für das Auslesen der Messungen ist auch eine eigene Methode integriert, welche mit der ID und der Anzahl der zu lesenden Messungen aufgerufen wird. Abgerufen werden immer die ältesten Messungen und als Rückgabewert ist eine Liste von den gelesenen Messungen definiert.
- **BaseOnlineDB**
  Die Klasse *BaseOnlineDB* dient als Basisklasse für die Klasse *OnlineDB*. In ihr sind fundamentale Objekte und Methoden definiert, welche von jeder Online-Datenbank benötigt werden. Dazu zählen die Referenz zur Konfigurationsdatei-Klasse, eine Referenz zur *BufferDB* und diverse Methoden.

- **OnlineDB**

  In der Klasse *OnlineDB* werden alle Funktionalitäten für den Transfer der Messungen in die jeweilige Online-Datenbank integriert. Diese muss deshalb bei jeder Online-Datenbank angepasst werden. Wie bereits beschrieben, erbt diese Klasse von der Klasse *BaseOnlineDB*. Um die Variablen in der Online-Datenbank mit den richtigen Parametern zu versorgen, wird in der Klasse *OnlineDB* eine Liste der Variablen mit den zu verwendenden Parameter erstellt. Diese wird zur selben Zeit generiert, wie die generelle Liste in *VariableList.* Aus dieser Liste werden beim Transfer der Daten in die Online-Datenbank die jeweiligen Parameter entnommen und angefügt.

Ablauf:

In Kapitel 8.4.2 sind die einzelnen Befehle der Schnittstelle bereits angeführt. Die Hauptsteuerung geht vom Manager über die Befehle der Schnittstelle aus. In diesem Abschnitt soll der komplette Ablauf in der DLL angeführt werden.

- **Initialisieren der DLL**

  Zum Initialisieren der DLL ist eine eigene Methode angelegt. Mit ihr werden diverse Parameter übergeben, die für den weiteren Programmablauf nötig sind:

  - Pfad der Konfigurationsdatei Conf_DBConnector
  - Pfad der Puffer-Datenbank (BufferDB)
  - Hash-Wert der Conf_DBConnector, die vom Manager verwendet wird

  Als erster Schritt wird beim Initialisieren die *Conf_DBConnector* gelesen und auch der Hash-Wert gebildet. Anschließend werden die Klassen *BufferDB* und *OnlineDB* initialisiert. Danach erfolgt der Vergleich der beiden Hash-Werte, wobei die Methode vorzeitig abgebrochen wird, wenn die Werte unterschiedlich sind. Abschließend versucht die *BufferDB* die Datenbankdatei zu öffnen oder zu erstellen. Ist dieser Versuch erfolgreich, ist die Initialisierung erfolgreich abgeschlossen und es wird Eins zurückgegeben, andernfalls ein Wert kleiner Null.

- **Verwaltung der Indizes**

  Sind alle Schritte der Initialisierung mittels der Methode *InitCheck(…)* abgeschlossen, werden die Indizes für die einzelnen Variablen erstellt. Der Manager erstellt für jede Variable im gesamten System eine eindeutige ID und übergibt diese mit Data-Collector-Name und Tag-Name über die Methode *GetIdxOfVars(…)* an die DLL. Die Kombination aus Data-Collector-Name und Tag-Name ist im gesamten System eindeutig und wird deshalb auch vor der Indexerstellung vom Manager überprüft. Nach Übergabe der Parameter (ID, Data-Collector-Name und Tag-Name) wird in der Klasse *VariableList* eine Liste angelegt, welche diese Parameter beinhaltet, plus dem Tabellen-Namen der Variable, der sich aus dem Data-Collector-Name und Tag-Name zusammensetzt. Nach Generierung der Tabellennamen der Variable wird überprüft, ob dieser in der *BufferDB* vorhanden ist und gegebenenfalls angelegt. Bei Erstellung der Liste werden zusätzlich die einzelnen Parameter eines Tags im *Conf_DBConnector* ausgelesen. Dies ist nötig, um eine weitere Liste für das Transferieren der Messungen in die Online-Datenbanken zu ermöglichen. In der Konfigurationsdatei *Conf_DBConnector* sind für jede Messung Parameter hinterlegt, wie sie in der Online-Datenbank abgelegt werden sollen. So kann je nach Umsetzung der Anbindung zur Online-Datenbank, beispielsweise pro Variable, eine eigene Tabelle und Bezeichnung vergeben werden.

Dadurch entfällt die Abhängigkeit vom Data-Collector-Namen und Tag-Namen. Diese Parameter werden mit der ID in einem Dictionary an die Klasse *OnlineDB* übergeben, welche die Daten wiederum in einer eigenen Liste speichert. Mit dieser Liste können bei jedem Upload der Messungen die richtigen Parameter ergänzt werden.

- **Verwalten der Messungen**

  Wie bereits in den Anforderungen definiert, werden die Messungen in einer SQLite-Datenbank zwischengespeichert. Somit wird die Datensicherheit erhöht und Schwankungen des Internetzugangs können überbrückt werden. Die Daten werden mit der Methode *ReceiveValues(…)* vom Manager, mit dem Parametern ID, Zeitpunkt der Messung und dem Messwert, übergeben. Der Parameter Messwert ist *dynamic* womit die Genauigkeit des Messwertes immer genau erhalten bleibt. Die übergebenen Parameter werden danach wiederum der Klasse *BufferDB* übergeben, welche die Messung in die SQLite-Datenbank schreibt. Bevor die Daten in die Datenbank geschrieben werden, wird mittels des Indizes der Table-Name von der Klasse *VariabelList* eruiert. Für das Abfragen und Löschen der Daten wird in der Klasse *BufferDB* eine Methode generiert, welche eine abgefragte Menge von Messungen zurückgibt. Diese Methode wird in der Basisklasse *BaseOnlineDB* aufgerufen.

- **Senden der Daten an die Online-Datenbank**

  Die genaue Implementierung für das Senden der Daten in die Online-Datenbank kann nicht definiert werden, da jede Anbindung den Anforderungen der jeweiligen Online-Datenbank entsprechen muss. Hier werden nur die Grundabläufe beschrieben, die in allen Realisierungen gleich implementiert sein sollen. Wird vom Manager die Methode *StartPostVarsToOnlineDB()* aufgerufen, wird damit begonnen Daten aus der *BufferDB* zu entnehmen und an die jeweilige Online-Datenbank zu transferieren. Die Funktionalität für das Transferieren in die Online-Datenbank ist in einem eigenen Thread untergebracht, wodurch eine Unabhängigkeit zur Schnittstelle erreicht wird. Die Upload-Intervalle können somit selbst bei der Programmierung bestimmt werden.

In Abbildung 31 ist der schematische Ablauf dargestellt. Zu Beginn ruft der Manager die *InitCheck(…)*-Methode der Schnittstelle auf. Danach werden mit der Methode *GetIdxOfVars(…)* alle Variablen übergeben und diverse Listen für eine schnellere Abarbeitung erzeugt. Sind diese Schritte abgeschlossen, können mit der Methode *ReceiveValues(…)* kontinuierlich Messwerte übergeben werden, welche in die Datenbank abgespeichert werden. Mit der Methode *StartPostVarsToOnlineDB()* wird ein eigener Thread für das Transferieren in die Online-Datenbank gestartet. In diesem Thread werden die Daten aus der *BufferDB* ausgelesen und in die Online-Datenbank transferiert.

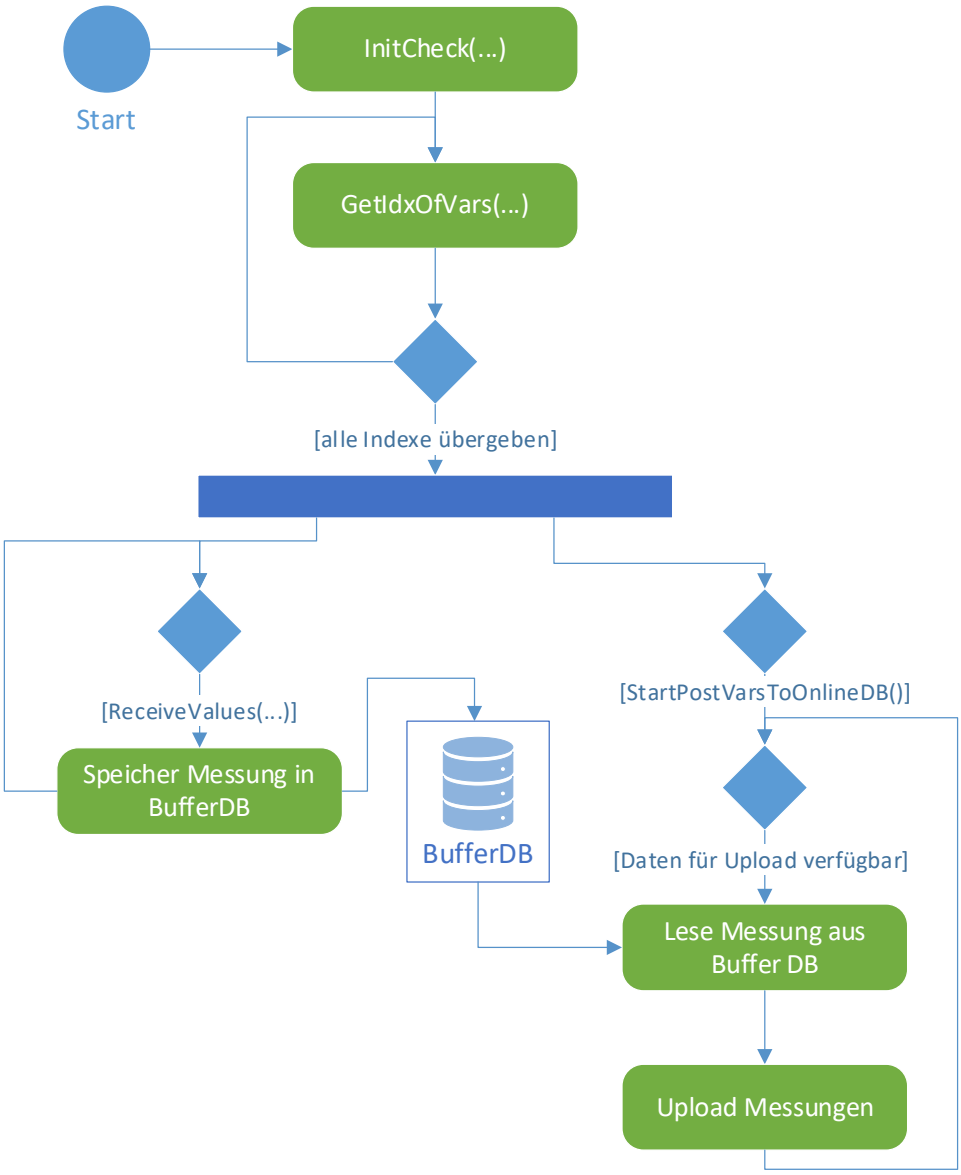

Abbildung 31: Schematischer Ablauf der DB-Connectors, Quelle: Eigene Darstellung.

10.3 Realisierung der Online-Datenbank-Anbindung

Nachdem die generelle Funktionalität definiert ist, folgt nun die Anbindung der beiden Online-Datenbanken.

10.3.1 InfluxDB Cloud

Für das Speichern der Daten in der InfluxDB Cloud wird ein Account bei InfluxDB angelegt (https://cloud2.influxdata.com/signup). Bei der Einrichtung des Accounts kann zwischen unterschiedlichen Cloud-Arten gewählt werden. Für diese Realisierung wird das kostenlose Modul gewählt, mit den technischen Daten, die in Kapitel 10.1 angeführt sind. Ein weiterer Auswahlpunkt ist der Cloudanbieter, welcher die Datenbank hostet. InfluxDB stellt nur die Technologie zur Verfügung, physisch werden die Daten aber von einem Cloudanbieter gehostet. Zur Auswahl stehen AWS, Microsoft Azure und Google-Cloud. Diese Realisierung behält die Default-Einstellung AWS bei.

Konfiguration der Online-Datenbank:

Nach der Registrierung wird für die Messdaten ein *Bucket* angelegt. Ein *Bucket* ist bei dieser Online-Datenbank vergleichbar mit einer eigenen Datenbank. Ein Account kann somit mehrere „Datenbanken“ verwalten. Das erstellte *Bucket* erhält die Bezeichnung „MyMasterBucket“. Des Weiteren muss dem *Bucket* eine Aufbewahrungsrichtlinie zugewiesen werden (Richtlinie wie lange die Daten gespeichert bleiben sollen, Kapitel 4.1.4). Dieses *Bucket* muss dem Datenbank-Client mitgeteilt werden, um die Daten in dieses *Bucket* zu speichern.

Um Zugriff auf ein *Bucket* zu erhalten, muss ein *Token* angelegt werden. Ein *Token* ist ein Schlüssel, den der Client für das Transferieren der Daten benötigt. Bei dem *Token* können die Zugriffsrechte für einzelne *Buckets* definiert werden. Für das *Bucket* „MyMasterBucket“ ist ein *Token* „Token For My Master 1“ angelegt.

Mit diesen Einstellungen ist die Datenbank für das Empfangen von Daten eingerichtet.

Konfigurationsdatei Einträge (Conf_DBConnector):

Bevor mit der Implementierung begonnen wird, werden die benötigten Parameter der Konfigurationsdatei definiert. Um Daten an die InfluxDB zu senden, benötigt der Client folgende Daten:

- **Bucket**
  Wird im vorherigen Abschnitt angelegt.
- **Token**
  Wird im vorherigen Abschnitt angelegt.
- **ORG (UserIn)**
  Hier muss die E-Mail-Adresse eines gültigen Users oder Userin des Accounts angegeben werden. Die Besitzerin oder der Besitzer eines Accounts kann mehrere UserInnen hinzufügen.
- **URL**
  Ist die Internetadresse, die für den Zugriff auf die Online-Datenbank benötigt wird.

All diese Parameter werden in der InfluxDB-Konfigurationsdatei im Feld *General* eingefügt.

Für die Speicherung der einzelnen Messungen in der Online-Datenbank werden pro Messung zwei Einträge hinzugefügt.

- **Measurement**
  Ist eine Sammlung von Messungen und kann wie eine Tabelle im relationalen Datenbankmodell angesehen werden.
- **Tag**
  Pro Variable können mehrere *Tags* vergeben werden. In dieser Konfiguration wird nur ein *Tag* erstellt, denn für die Realisierung ist das hinreichend genug. Mit dem *Tag* kann einer Messung ein Name gegeben werden, wobei bei dieser Realisierung der Name einer Messung immer derselbe sein wird.

Diese Einträge werden jeder Variable hinzugefügt und müssen vor Verwendung ausgefüllt werden. Die Kombination aus *Measurement* und *Tag* darf pro Konfigurationsdatei nur ein einziges Mal vergeben werden und wird dementsprechend auch geprüft.

DLL Implementation:

Für die Implementation der Kommunikation zwischen DLL und InfluxDB wird das NuGet-Package „InfluxDB.Client“ installiert, welches alle Funktionalitäten für die Kommunikation beinhaltet. Wie bereits definiert, wird die komplette Realisierung der Funktionalität in der Klasse *OnlineDB* realisiert.

Bei der Instanziierung der Klasse *OnlineDB* werden die Einträge der Konfigurationsdatei *Conf_DBConnector* übergeben. Mit den Einträgen aus dem Feld *General* wird ein InfluxDB-Client initialisiert, wofür die Einträge URL und Token benötigt werden.

Um alle Messungen der Online-Datenbank zuweisen zu können wird die Methode der Basisklasse *BaseOnlineDB* überschrieben, die in der Schnittstellen-Methode *GetIdxOfVar(…)* aufgerufen wird. In der Methode werden alle Parameter, die für die Zuweisung eines Messpunktes in der Online-Datenbank benötigt werden, in ein lokales Dictionary (VarList) der Klasse *OnlineDB* geschrieben. Aus diesem Dictionary werden die Informationen gelesen, wenn Daten in die Online-Datenbank transferiert werden.

Für den Upload der Daten wird eine Methode entwickelt, die dem Thread zugewiesen und mit der Methode *StartPostVarsToOnlineDB()* gestartet wird. Diese Methode läuft in einer Endlosschleife, bis die Applikation beendet wird. Die Methode durchläuft ständig alle Einträge in der lokalen *VarList* und versucht bei jedem Durchlauf 100 Messungen einer Variablen aus der *BufferDB* zu holen. Mit den Messungen aus der *BufferDB* wird eine Liste erstellt, die einem speziellen Format der InfluxDB entspricht. Pro Eintrag in der Liste wird angegeben:

- Measurement (von Konfigurationsdatei)
- Tag (von Konfigurationsdatei)
- Field (mit dem Namen Value und als Wert den gemessenen Messwert vom Data-Collector)
- Timestamp (Zeitpunkt der Messung in UTC von Data-Collector)

Ist die Liste vollständig, werden die Daten mittels Befehl vom bereits initialisierten Client an die Datenbank transferiert. Für das Schreiben werden der Methode das *Bucket,* der *User* und die Liste der Messungen übergeben. Es würde auch ein Befehl für das Schreiben nur eines Messwertes zur Verfügung stehen, jedoch ist die Performance erheblich besser, wenn pro Methode mehrere Messungen geschrieben werden.

Online Daten:

Abbildung 32 zeigt eine Darstellung einiger Beispieldaten des *Buckets MyMasterBucket*. Die Daten werden mit der DLL in die Online-Datenbank transferiert, wobei das gesamte System bereits funktioniert. Diese Ansicht wird im Account der Online-Datenbank zur Verfügung gestellt. Zu sehen sind auch einige Werkzeuge, welche für die Auswertung verwendet werden können. Im linken unteren Feld ist das *MyMasterBucket* ausgewählt. Der mittlere untere Filter zeigt, dass nur Messungen von der *Measurement* „Measurement1“ angezeigt werden.

Abbildung 32: Darstellung von Onlinewerten der InfluxDB, Quelle: Eigene Darstellung.

10.3.2 Microsoft Azure Cosmos DB

Für die Realisierung der Cosmos DB muss ein Account bei Microsoft Azure angelegt werden (https://portal.azure.com). Bei der Anmeldung muss eine E-Mail-Adresse angegeben und danach bestätigt werden. Nach dem Kreieren des Accounts kann aufgrund der Studententätigkeit ein gewisser Freibetrag für die Erforschung der Online-Services beantragt werden.

Konfiguration der Online-Datenbank:

Für das Speichern der Daten wird dafür eine Cosmos DB-Ressource angelegt. Bei der Konfiguration der Online-Datenbank können wesentlich mehr Einstellungen vorgenommen werden als bei der InfluxDB. Dies liegt auch daran, dass die Cosmos DB eine Multimodel-Datenbank ist. Bei der Erstellung der Datenbank kann angegeben werden, welche API verwendet wird, wodurch die Art der Datenbank bestimmt wird. Folgende APIs stehen bei der Datenbank zur Verfügung:

- SQL-API (Dokumenten-Datenbank)
- MongoDB API (Dokumenten-Datenbank)
- Cassandra API (Schlüssel-Wert-Datenbank)
- Gremlin API (Graph-Datenbank)
- Table API (Wide-Column-Datenbank)

Wie bereits definiert, soll eine Wide-Column-Datenbank realisiert werden, weshalb die *Table API* gewählt wird. Die wichtigsten Einstellungen, welche zusätzlich vorgenommen werden, sind:

Kontoname	my-master-db	
Standort	Europa, Western	Ort, wo die Daten gespeichert werden
Kontotyp	Nicht-Produktion	Kosten sind dadurch günstiger, da die Datenbankinstanz nicht hochverfügbar sein muss
Sicherungsintervall	60 Minuten	Zeitintervall, wie oft die Daten gesichert werden
Aufbewahrungsdauer für Sicherungen	8 Stunden	Dauer, wie lange die Sicherungen gespeichert werden
Datenverschlüsselung	Dienstseitig verwalteter Schlüssel	

Nach der Konfiguration kann damit begonnen werden, Daten in die Datenbank zu speichern. Für die Verbindung zur Datenbank wird nur ein Connection-String benötigt, der nach erfolgreicher Ressourcen-Generierung dem Account entnommen werden kann.

Konfigurationsdatei Einträge (Conf_DBConnector):

Für die Verbindung zur Cosmos DB wird nur der Connection-String benötigt. Dafür wird im Feld General der Konfigurationsdatei *Conf_DBConnector* ein Eintrag *ODB_ConString* angelegt und der Wert aus dem Account eingefügt.

Wie bei der InfluxDB werden auch bei der Cosmos DB Einträge pro Messung hinzugefügt, um diese in der Online-Datenbank abspeichern zu können. In der verwendeten Art der Cosmos DB (Table API) werden Tabellen und einzelne Einträge für die Speicherung verwendet. Deshalb erhält zuerst jede Variable den Eintrag *Table*, um zu definieren, in welcher Tabelle die Messung gespeichert wird.

Ein Eintrag in einer *Table* muss immer aus einem *Partition Key* und einem *Row Key* bestehen. Darüber hinaus können noch beliebig viele selbst definierte Felder hinzugefügt werden. In dieser Realisierung wird in der Konfigurationsdatei ein Eintrag *Tag* hinzugefügt, welcher dem *Partition Key* zugewiesen wird. Dem *Row Key* wird immer der Zeitpunkt der Messung zugewiesen. In der Cosmos DB müssen *Partition Key* und *Row Key* eindeutig sein, da bereits bestehende Einträge überschrieben werden würden. Da eine Variable immer nur eine Messung pro Zeitpunkt generieren kann, ist auch sichergestellt, dass nie ein gleiches Paar auftreten kann.

Des Weiteren darf pro Konfigurationsdatei die Kombination aus *Table* und *Tag* nur einmal vergeben werden.

DLL Implementation:

Die Implementierung der Kommunikation beginnt mit der Installation des NuGet-Package „Microsoft.Azure.Cosmos.Table". Dieses Package enthält alle Funktionalitäten, welche für die Realisierung der Online-Datenbankanbindung benötigt werden. Wie bei der InfluxDB werden all diese Funktionalitäten in der Klasse *OnlineDB* realisiert.

Bei der Initialisierung der Klasse *OnlineDB* werden anfangs wieder die Einträge der Konfigurationsdatei übergeben. Mit dem Eintrag *ODB_ConString* wird ein Online-Datenbank-Client initialisiert.

Die Liste *VarList,* welche auch bei der InfluxDB angelegt wird, um die Zuweisung der Messpunkte zur Online-Datenbank zu erhalten, wird auch bei der Cosmos DB gleich implementiert. Es werden jedoch nur die Einträge verwendet, welche spezifisch für die Cosmos DB angelegt werden, implementiert.

Nach dem Erstellen der Liste wird in der Online-Datenbank überprüft, ob alle referenzierten *Tables* verfügbar sind. Ist ein *Table* nicht vorhanden, wird dieser in der Online-Datenbank erstellt.

Für das Starten des Upload-Prozesses wird eine Methode einem eigenen Thread zugewiesen, welcher mit der Schnittstellen-Methode *StartPostVarsToOnlineDB()* gestartet wird.

Die Methode ist in derselben Art und Weise implementiert wie bei der InfluxDB. Es wird nacheinander die gesamte Liste *VarList* durchgegangen und aus der *BufferDB* die Messwerte geholt, wobei wieder 100 Messungen pro Durchlauf entnommen werden. Pro Upload kann nur ein Messwert übergeben werden, jedoch ist die Upload-Methode *als async* ausgeführt. Das heißt, dass die 100 Methoden für das Uploaden gestartet werden und danach auf die Fertigstellung sämtlicher Methoden gewartet wird. Der Upload-Methode werden die Parameter (Messzeitpunkt, Messwert, *Table* und *Tag*) nicht direkt übergeben, sondern ein Objekt von einer selbst definierten Klasse. Diese Klasse erbt von einer Basisklasse des Packages „Microsoft.Azure.Cosmos.Table", worin auch die Zuweisung für den *Partition Key* und des *Row Key* realisiert ist. Sind alle Methoden abgearbeitet, wird mit der nächsten Variablen weitergearbeitet.

Online Daten:

In Abbildung 33 ist ein Ausschnitt von der Cosmos DB zu sehen. Im rechten unteren Feld sind einige Einträge von Messpunkten zu sehen. Wie beschrieben ist der *Partition Key* derselbe Wert, wie der Eintrag „*Tag"* in der Konfigurationsdatei. Als *Row Key* ist der Zeitpunkt der Messung definiert und *Value* repräsentiert den Messwert. Zusätzlich wird noch das Feld *Timestamp* angezeigt, welches den Zeitpunkt angibt, wann der Eintrag hinzugefügt wurde. Im linken oberen Feld werden die verschiedenen *Tables* angezeigt, wobei hier nur der *Table Measurement2* erstellt ist. Wie aus dem Ausschnitt zu sehen ist, bietet Cosmos-DB einige Werkzeuge für Abfragen der Daten. Standardmäßig ist keine Visualisierung implementiert, jedoch kann mit *Notebooks* (links unten) eine eigene Darstellung generiert werden.

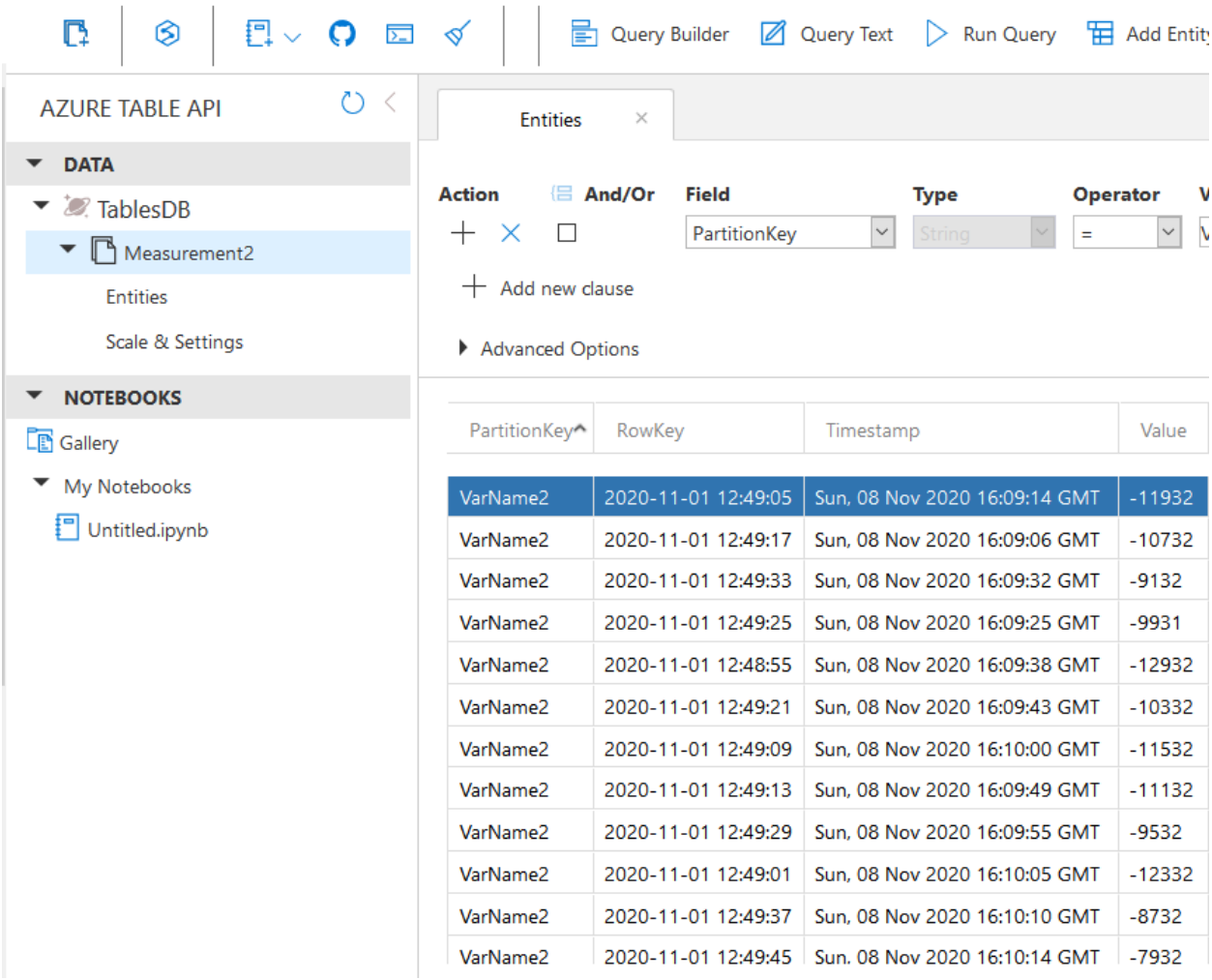

Abbildung 33: Comos DB Online-Darstellung, Quelle: Eigene Darstellung.

11 MANAGER

Als letzte Komponente wird der Manager realisiert. Der Manager wird im System, im Gegensatz zu den anderen Komponenten, nur ein einziges Mal ausgeführt. Der Manager dient als Bindeglied zwischen den einzelnen Data-Collectors und den DB-Connectors.

11.1 Definitionen

Konfigurationsdateien:

Wie des Öfteren bereits angeführt ist, erfolgt die komplette Konfiguration mittels Konfigurationsdateien. Die Konfigurationsdateien des Data-Collectors (Conf_DataCollector) und DB-Connectors (Conf_DBConnector) sind bereits definiert. Der Manager benötigt sämtliche Konfigurationsdateien, um Informationen daraus zu erhalten und um mit den anderen Komponenten kommunizieren zu können. Damit der Manager weiß, welche Komponenten zu verwenden sind, befindet sich in der Konfigurationsdatei *Conf_Manager* eine Auflistung mit den Pfaden der Konfigurationsdateien (*Conf_DataCollector* und *Conf_DBConnector).*

Mit den Konfigurationsdateien kann der Manager die richtige Zuordnung der gesammelten Messungen von den Data-Collectors zu den Online-Datenbanken herstellen. Der Manager soll die Datenpunkte nur an jene DLL schicken, welche die Daten letztendlich benötigen. In Abbildung 34 ist eine Zuordnung der einzelnen Messpunkte grafisch dargestellt. Es ist ersichtlich, dass ein Data-Collector mehrere Variablen hat, welche aber in verschiedenen Online-Datenbanken abgespeichert werden können. Es muss nur die Konfiguration richtig erstellt werden.

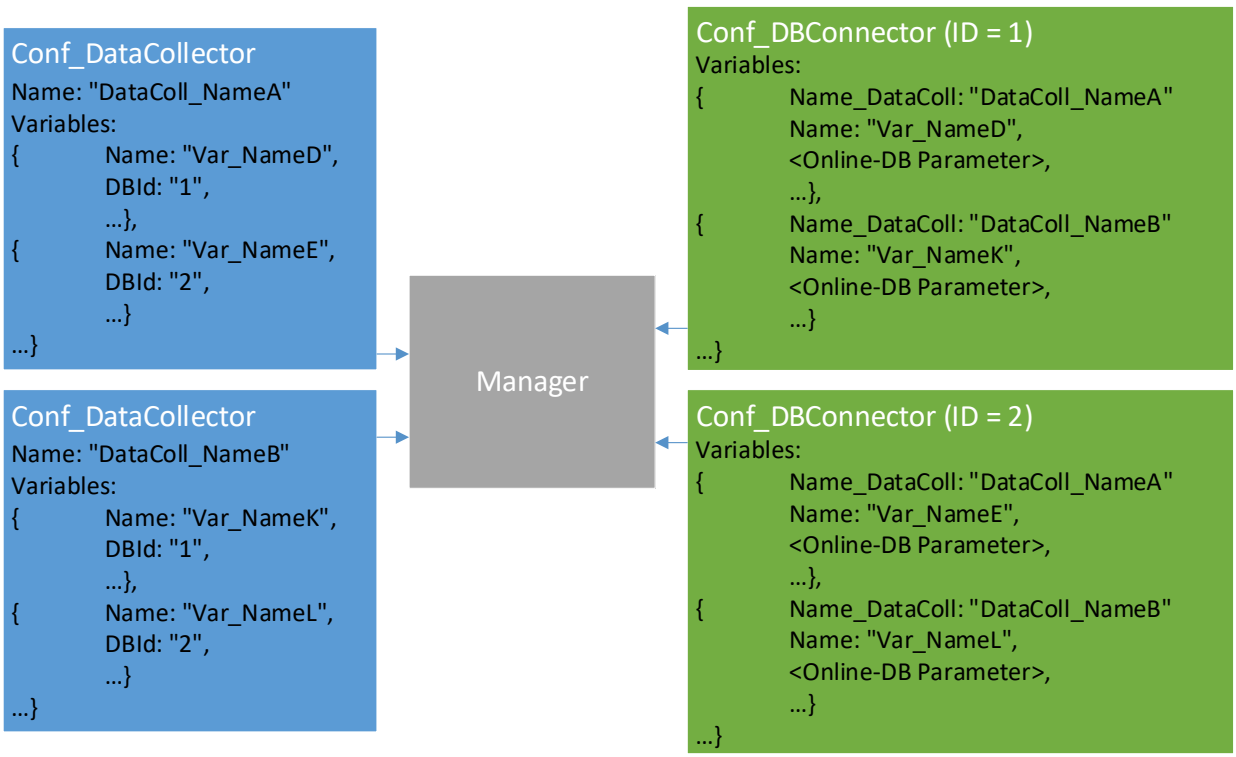

Abbildung 34: Zuordnung von Variablen über das System, Quelle: Eigene Darstellung.

Des Weiteren befindet sich in der Konfigurationsdatei *Conf_Manager* der Pfad für das Zertifikat, womit die Authentizität der Data-Connectors überprüft wird. Das Auslesen der einzelnen Parameter aus den Konfigurationsdateien erfolgt gleich wie bei den Data-Collector und DB-Connector und wird deshalb nicht nochmals angeführt.

Holen der Daten von den Data-Connectors:

Im Manager wird für das Holen von Daten jeweils ein eigenes Objekt pro Data-Connector instanziiert. In diesem Objekt werden alle Funktionalitäten implementiert, welche dafür erforderlich sind. Um eine Unabhängigkeit von den anderen Programmteilen zu erhalten, wird dafür ein eigener Thread generiert. Jedes Objekt für das Holen der Daten enthält einen eigenen HTTP-Client für die Kommunikation mit dem jeweiligen Data-Collector. Nach der Initialisierung des Objektes wird zuerst der Hash-Wert der Konfigurationsdatei des Data-Collectors gelesen und mit der, für diesen Data-Collector, dem Manager zugewiesenen Konfigurationsdatei verglichen. Passen die Hash-Werte nicht genau überein, soll eine Warnung ausgegeben und die Kommunikation nicht fortgeführt werden. Danach werden ständig alle Variablen des Data-Collectors abgefragt und an die jeweiligen DLL (DB-Connector) weitergeleitet.

Zur sicheren Authentifizierung soll bei jeder Kommunikation das Zertifikat des Data-Collectors überprüft werden.

Weiterleiten der Daten an die DB-Connectors:

Das Weiterleiten der Daten an die DB-Connectors soll in einer eigenen Klasse implementiert werden. In der Konfigurationsdatei *Conf_DBConnector* ist der Pfad der DLL angegeben, welche für den jeweiligen DB-Connector geladen werden soll. Die DLL wird zur Laufzeit geladen und kann auch mehrmals geladen werden. In der Klasse werden sämtliche Schnittstellen-Methoden implementiert, welche bereits beim DB-Connector definiert sind:

- InitCheck(…)
- GetIdxOfVars(…)
- StartPostVarsToOnlineDB()
- ReceiveValues(…)

Die Funktionsweise der einzelnen Schnittstellen-Methoden sind bereits beim DB-Connector erklärt. Wie die Umsetzung im Manager erfolgt, wird in Kapitel 11.3 ausgeführt.

11.2 Architektur

In Abbildung 35 ist der Ablauf des Managers grafisch dargestellt. Zu Beginn wird die Konfigurationsdatei *Conf_Manager* ausgelesen, worin definiert ist, welche anderen Konfigurationsdateien (*Conf_DataCollector* und *Conf_DBConnector*) geladen werden müssen. Aus all den Konfigurationsdateien (*Conf_DataCollector* und *Conf_DBConnector*) wird je ein einzelnes Objekt instanziiert. Anschließend finden diverse Überprüfungen der Konfigurationen statt, um einen reibungslosen Ablauf sicherzustellen. Wenn alle Initialisierungsschritte abgeschlossen und alle Einstellungen vorgenommen sind, wird mit dem Holen der Daten von den Data-Collectors und dem Transfer der Daten zu den Online-Datenbanken begonnen.

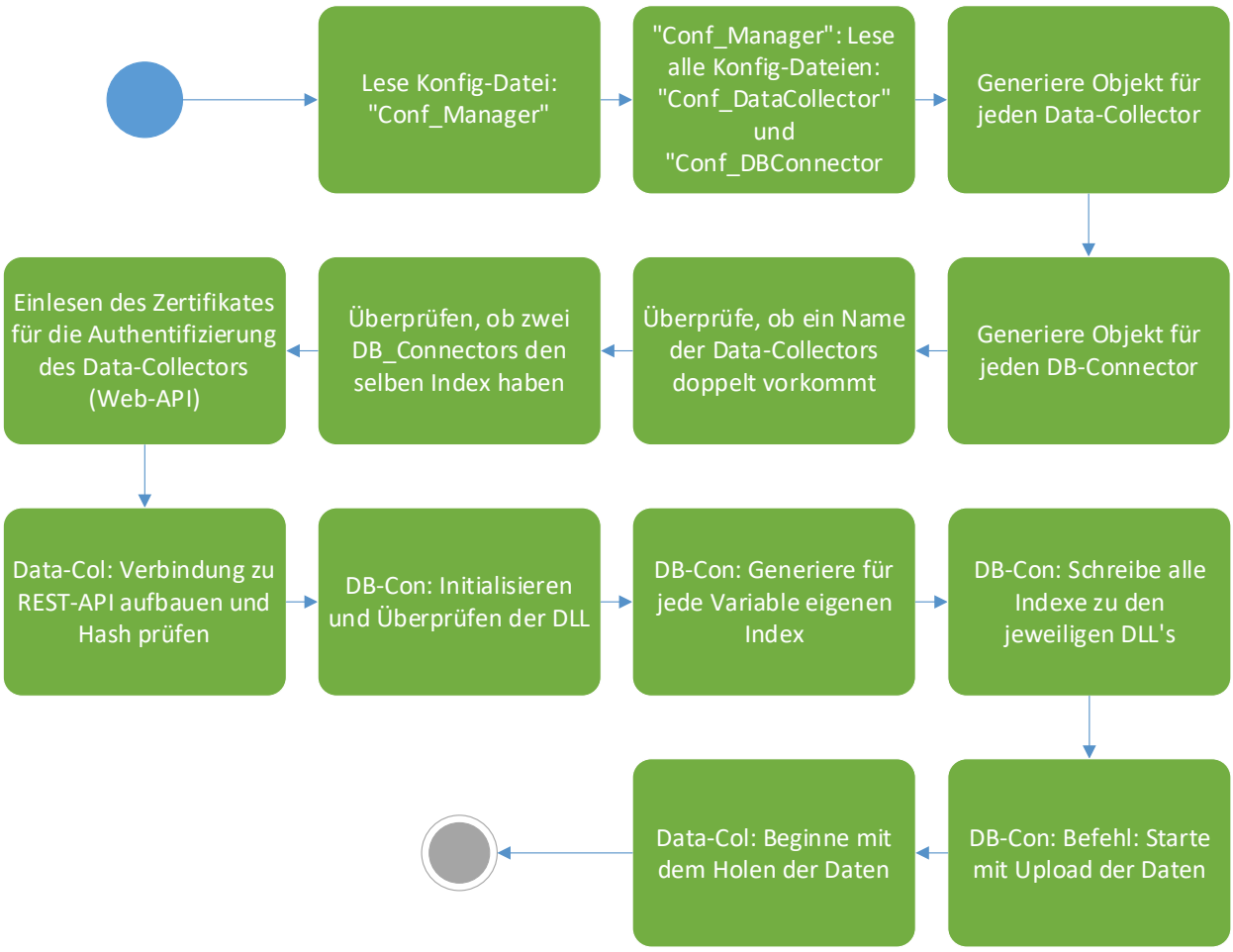

Abbildung 35: Genereller Ablauf Manager: Quelle: Eigene Darstellung.

11.3 Umsetzung

Dieses Unterkapitel behandelt wichtige Umsetzungen der Definitionen im Manager.

Authentifizierung des Data-Collectors:

Der Manager holt die Daten der einzelnen Data-Collectors über HTTP. Eine Anforderung des Systems ist es, verschlüsselt zu kommunizieren. Deshalb wird ein Zertifikat erstellt, mit dem sich der Data-Collector authentifizieren und einen öffentlichen Schlüssel zur Verfügung stellen kann. Die Verschlüsselung wird nicht mit dem Zertifikat erreicht, sondern mit dem Schlüsselpaar (Zertifikat des Data-Collectors) kann ein Sitzungsschlüssel generiert werden. Um sicherzustellen, dass der Webserver des Data-Collectors wirklich das Zertifikat besitzt, überprüft der Manager den Hash-Wert des übermittelten Zertifikats. Der Manager hat dasselbe Zertifikat gespeichert, jedoch ohne dem privaten Schlüssel. Denn nur wenn der Data-Collector den privaten Schlüssel zum öffentlichen Schlüssel kennt, kann das gemeinsame Geheimnis generiert werden.

Generieren der Indizes:

Die Zuordnung der Variablen wird im Manager mit Indizes realisiert. Bevor Daten geholt und weitergeleitet werden, erhält jeder Datenpunkt einen eigenen, im gesamten System eindeutigen, Index. Nachdem alle Konfigurationsdateien (*Conf_DataCollector* und *Conf_DBConnector*) geladen sind, stehen alle Variablen

fest, welche zu sammeln sind. Es werden in jeder Konfigurationsdatei *Conf_DataCollector* die Variablen nacheinander gelesen und ein Index vergeben. Gleichzeitig wird mit dem Eintrag *DBId* im *Conf_DataCollector* geprüft, ob sich dieser Eintrag auch in der jeweiligen Konfigurationsdatei *Conf_DBConnector* befindet. Kann eine Referenz hergestellt werden, wird dieser Index im Objekt des jeweiligen Data-Collectors und zugleich im Objekt des DB-Connectors abgespeichert. Wenn die Zuordnung nicht passt, wird eine Warnung ausgegeben und die Variable wird im weiteren Verlauf nicht bearbeitet.

Die Übertragung der Indizes zu den DLLs der DB-Connectors wird separat mit der Schnittstellen-Methode *GetIdxOfVars(…)* gestartet.

Laden der DLL:

In jeder Konfigurationsdatei der DB-Connectors ist der Pfad der DLL definiert. Die DLL wird zur Laufzeit mit der Systembibliothek „*System.Reflection*“ geladen. Ist das Laden erfolgreich, wird mit einer Methode der Inhalt der DLL durchsucht. Durch die Festlegung, dass sich die Schnittstellen-Methoden immer in der Klasse „*Interface“* befinden müssen, wird ein Objekt mit diesem Namen instanziiert. Ist keine Klasse mit diesem Namen enthalten, wird ein Fehler ausgegeben und die DLL kann nicht geladen werden. Das instanziierte Objekt wird als *dynamic* definiert und somit können die Schnittstellen-Methoden einfach aufgerufen werden, ohne dass sie bei der Kompilierung feststehen. Wäre eine Methode bei der Ausführung nicht vorhanden, würde der Manager einen Fehler ausgeben.

Datenmanagement:

Sind alle Voreinstellungen abgeschlossen, wird mit dem Holen der Daten begonnen. Jeder Data-Collector ist im Manager durch ein eigenes Objekt repräsentiert. In diesem Objekt werden in einem eigenen Thread die Daten vom Data-Collector geholt. Die Zieladresse und Portnummer entnimmt der Manager aus der Konfigurationsdatei *Conf_DataCollector*, mit der auch der Data-Collector selbst konfiguriert wird. Durch die in Kapitel 8.4.1 durchgeführte Schnittstellendefinition der REST-API können die Daten mit den Namen der Variablen abgefragt werden. Bei jeder Abfrage gibt der Data-Collector die Messungen einer Variablen, maximal jedoch 1000 Wert, zurück. Die Antwort des Data-Collectors wird dann vom JSON-Format in eine C#-Struktur umgewandelt, wobei der Messwert als *dynamic* definiert ist, um keine Probleme mit den verschiedenen Datentypen, beim Sammeln der Daten, zu erhalten. Nach der Konvertierung wird aufgrund des Ziels der Variable, die Methode der jeweiligen DLL aufgerufen, die die Messung mit dem Index an den DB-Connector sendet.

12 KOMPONENTENTESTS

Nach der Fertigstellung sämtlicher Komponenten, wird in diesem Kapitel ein finaler Test der Komponenten durchgeführt. Dieser Test soll zeigen, wie ausgereift das Gesamtsystem ist und Rückschlüsse auf die Verwendbarkeit in der Automatisierungstechnik geben.

Bei dem Test sollen alle entwickelten Komponenten getestet werden:

- Data-Collector OPC UA
- Data-Collector TCP/IP
- Manager
- DB-Connector InfluxDB
- DB-Connector Cosmos DB

12.1 Aufbau

Zum Testen der Komponenten und Anforderungen ist ein Testsystem aufgebaut. Bei dem Test werden drei verschiedene Hardwaregeräte verwendet:

- **SPS**
  Die SPS dient als Datenquelle für OPC UA sowie für TCP/IP. Auf der SPS ist ein Test-Programm installiert, welches verschiedene Simulationsdaten generiert. Die SPS stellt die Daten per OPC UA Server zur Verfügung und sendet sie mit TCP/IP an den Data-Collector TCP/IP.
- **Laptop1**
  Ein Laptop ist direkt mit der Ethernet-Schnittstelle der SPS verbunden. Auf dem Laptop1 sind ein Data-Collector OPC UA und ein Data-Collector TCP/IP installiert. Diese sind so konfiguriert, dass Daten von der SPS abgefragt werden. Beide Data-Collectors nutzen hierfür die Ethernet-Verbindung. Die Web-APIs der Data-Collectors werden beim Initialisieren so konfiguriert, dass sie sich in einem WLAN befinden, in dem sich auch der Laptop2 befindet. Die beiden REST-APIs werden über unterschiedliche Ports angesprochen.
- **Laptop2**
  Am Laptop2 ist der Manager, DB-Connector InluxDB und DB-Connector Cosmos DB installiert. Laptop2 befindet sich mit der WLAN-Schnittstelle im selben Netzwerk wie Laptop1 und holt über die REST-API die Daten der Data-Collectors. Wie in den Konfigurationsdateien festgelegt ist, werden diese Daten an den DB-Connector InfluxDB oder DB-Connector Cosmos DB weitergeleitet und anschließend in die jeweilige Online-Datenbank transferiert. Dabei wird wieder die WLAN-Schnittstelle des Laptop2 verwendet.

In Abbildung 36 ist der Aufbau grafisch dargestellt. Aufgrund der besseren Darstellung sind WLAN-Access-Point und Router getrennt dargestellt, obwohl sie eigentlich ein Gerät sind.

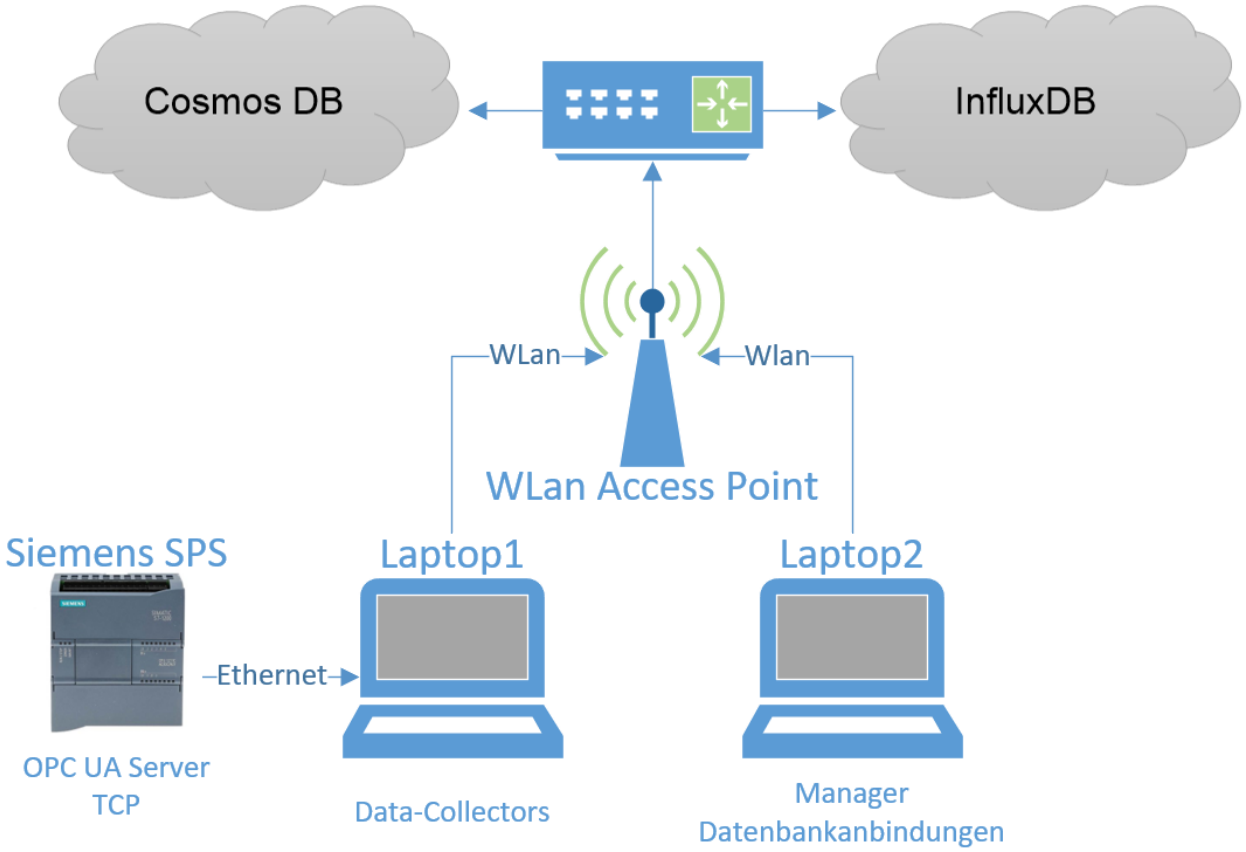

Abbildung 36: Komponententest Aufbau, Quelle: Eigene Darstellung.

Komponenten:

- **SPS**
  Modell: CPU 1214C AC/DC/Rly
  Version: 4.4
- **Laptop1**
  Modell: HP ProBook 650 G2
  OS: Windows 10.0.19041
- **Laptop2**
  Modell: Microsoft Surface Book
  OS: Windows 10.0.19042

12.2 Umsetzung

Konfiguration SPS:

Die Konfiguration und Programmierung der SPS wird mit TIA Portal V16 durchgeführt. Um eine skalierbare Testumgebung zu schaffen, wird eine Struktur generiert, welche zwei Boolean, zwei Integer und zwei Real beinhaltet. Diese Struktur wird in einem Datenbaustein, hinter einem Zehn-Byte-Array (Header TCP/IP DB-Collector), in einem Array 200 mal definiert. Für die Berechnung von Simulationswerten wird ein Funktionsbaustein generiert, der als Übergabeparameter die Test-Struktur erhält und diese mit Simulationswerten versorgt. In einer For-Schleife werden alle 200 Array-Einträge mit Simulationswerten

beschrieben. Die Berechnungen werden in einem *Cycle Interrupt OB* durchgeführt. Die Berechnungen werden alle 50 Millisekunden durchgeführt, denn bei einer Berechnungszeit von zehn Millisekunden tritt bei der SPS eine Zykluszeitüberschreitung auf.

OPC UA: Um die Simulationsdaten per OPC UA zur Verfügung zu stellen, muss in der Hardwarekonfiguration der SPS der OPC UA Server aktiviert werden. Des Weiteren wird das *Sampling Interval* und *Publishing Interval* erhöht, um eine bessere Performance zu erhalten. Dieses Modell der SPS hat eine Begrenzung von 500 Variablen, die gleichzeitig beobachtet werden können und es dürfen maximal 1000 Variablen für OPC UA definiert werden. Für den Test werden bei OPC UA keine Sicherheitsmechanismen aktiviert.

TCP/IP: Für die Datenerfassung über TCP/IP muss die SPS definierte Datenpakete an den Data-Collector senden. Dafür werden Systembibliotheken von Siemens verwendet, welche unter Angabe von Daten und Ziel, TCP-Pakete versenden können. Als Ziel wird die IP-Adresse der Ethernet-Schnittstelle des Laptop1 verwendet. Als Ziel-Port wird der Port 8500 eingestellt, welcher in der Konfigurationsdatei *Conf_DataCollector* definiert ist. Als Daten wird der Datenbaustein gewählt, in dem auch die Simulationsdaten gespeichert sind. Auch der Header von zehn Bytes ist in diesem nicht speicheroptimierten Datenbaustein bereits enthalten. Beim Senden wird nicht der gesamte Datenbaustein übermittelt, sondern nur die Anzahl der Bytes, die bei der Systemfunktion definiert ist.

Konfigurationsdateien Conf_DataCollector:

Die Konfigurationsdateien der Data-Collector (*Conf_DataCollector)* sind in der Tabelle 4 und Tabelle 5 ausgeführt. Daraus ist ersichtlich, dass beide Data-Collectors Daten von demselben Gerät holen, da die IP-Adressen identisch sind. Da die Data-Collectors am selben Gerät betrieben werden, sind auch die Ports der API unterschiedlich. Repräsentativ für alle Variablen ist die erste Variable dargestellt. Die Data-Collectors sammeln dieselben Daten, jedoch mit unterschiedlichen Protokollen. Am Namen der Variable ist ersichtlich, dass es sich um das erste Element des Arrays handelt. Für Elemente der anderen Variablen im Array wird die Nummer nach „Struct“ dementsprechend erhöht.

OPC UA	
General:	
Name:	DataCollector1_OPC,
UrlOPCServer:	opc.tcp: //192.168.1.10: 4840,
IPAddress_DataCollector_API:	192.168.0.103,
Port_DataCollector_API:	8010,
UseCertificateOPCUA:	false,
Path_APIexe:	C:\\Proj\\Master\\DataCollector_WebAPI_OPC \\DataCollector_WebAPI.exe
Variables:	
{	
Name:	Struct0.Bool1,
Datatype:	Bool,
UpdInterv:	15000,
Mode:	1,

Address:	ns=4;i=17,
DBId:	1
},	
…	

Tabelle 4: Einträge Konfigurationsdatei Data-Collector OPC UA, Quelle: Eigene Darstellung.

TCP/IP	
General:	
Name:	DataCollector_TCP1,
IPAddress_ReceiveData:	192.168.1.5,
Port_ForCollection:	8500,
IPAddress_DataCollector_API:	192.168.0.103,
Port_DataCollector_API:	8000,
Path_APIexe:	C:\\Proj\\Master\\DataCollector_WebAPI_TCP \\DataCollector_WebAPI.exe,
ByteSwapping:	true
Variables:	
{	
Name:	Struct0.Bool1,
Datatype:	Bool,
UpdInterv:	5000,
Mode:	1,
Position:	1,
DBId:	2
},	
…	

Tabelle 5: Einträge Konfigurationsdatei Data-Collector TCP/IP, Quelle: Eigene Darstellung.

Konfigurationsdateien Conf_DBConnector:

Die Konfigurationsdateien für die DB-Connectors sind in Tabelle 6 und Tabelle 7 ausgeführt. Bei diesen Dateien ist ein erheblicher Unterschied bei den Einträgen im Reiter *General ersichtlich*, da sie für unterschiedliche Online-Datenbanken ausgelegt sind. In der Darstellung sind Passwörter unkenntlich gemacht und lange Pfade gekürzt. Im Reiter *Variables* ist auch die Zuweisung der Variablen aus den vorherigen Konfigurationsdateien ersichtlich. Bei der InfluxDB werden die Messungen dahingehend separiert, dass *Measurement* mit der Nummer des Arrayelements beschrieben wird und der Tag mit dem Namen des Elements. Die Separierung wird bei der Comos DB gleich angewandt, nur dass die Bezeichnung für das Arrayelement *Table* heißt.

InfluxDB	
General:	
Name:	DB_Connector1,
ODB_Token:	***
ODB_Bucket:	MyMasterBucket,
ODB_Org:	bernhard.rossmann@***,

ODB_URL:	https: //eu-central-1-1.aws.cloud2.influxdata.com,
Path_DLL:	C:***\\DBConnector_InfluxDB.dll,
PathLocalDB:	C:***\\DBConnector1\\LocDB_Connector1.db,
Path_Logger:	C:***\\DBConnector1\\MyLogger1.log
Variables:	
{	
NameDataCollector:	DataCollector1_OPC,
Name:	Struct0.Bool1,
Datatype:	Bool,
Measurement:	TestMeas0,
Tag:	Bool1
},	
…	

Tabelle 6: Einträge Konfigurationsdatei DB-Connector InfluxDB, Quelle: Eigene Darstellung.

Cosmos DB	
General:	
Name:	DB_Connector2,
ODB_ConString:	DefaultEndpointsProtocol=https; AccountName=my-master-db;AccountKey=<***>; TableEndpoint=https: //my-master-db.table.cosmos.azure.com:443/;,
Path_DLL:	C:***\\DBConnector_Cosmos.dll,
PathLocalDB:	C:***\\DBConnector2\\LocDB_Connector2.db,
Path_Logger:	C:***\\DBConnector2\\MyLogger2.log
Variables:	
{	
NameDataCollector:	DataCollector_TCP1,
Name:	Struct0.Bool1,
Datatype:	Bool,
Table:	TestMeas0,
Tag:	Bool1
},	
…	

Tabelle 7: Einträge Konfigurationsdatei DB-Connector Comos DB, Quelle: Eigene Darstellung.

Konfiguration der Online-Datenbanken:

Bei den Online-Datenbanken werden keine Änderungen zu den bereits gemachten Einstellungen vorgenommen. Die neuen *Tables* und *Measurements* werden von den Data-Collectors selbständig angelegt.

Allgemeine Einstellungen:

Für die Kommunikation zwischen *Laptop1* und *Laptop2* müssen bei der Firewall des *Laptop1* die verwendeten Ports geöffnet werden.

12.3 Testablauf

Nachdem alle Einstellungen vorgenommen sind, werden verschiedene Durchläufe mit dem System ausgeführt. Es gilt dabei zu erarbeiten, wie das System auf unterschiedliche Lasten reagiert. Dabei wird immer eine unterschiedliche Anzahl der Strukturelemente des Datenbausteins ausgelesen.

Wenn pro Element in der Struktur vier Byte angenommen werden und für den Zeitstempel acht Byte, ergibt das pro Struktur insgesamt 96 Byte. Im Mittel wird ein jeder Wert zehn Mal pro Minute abgefragt, wodurch pro Struktur eine Datenmenge von 16 Byte in der Sekunde generiert wird.

Bei den Online-Datenbanken ist die Datenmenge höher, denn es werden verschiedene Strings mit in die Datenbanken geschrieben. Dies und der Overhead muss somit für die Berechnung der benötigten Datenübertragungsgeschwindigkeit berücksichtigt werden.

Es werden insgesamt vier verschiedene Durchläufe gemacht:

- 4 Strukturen
- 10 Strukturen
- 20 Strukturen
- 50 Strukturen

In Abbildung 37 ist ein Double-Wert, in dem jedem Zyklus ein Cosinus-Wert berechnet wird, aus jedem Strukturelement dargestellt. Aus dieser Darstellung ist ersichtlich, dass jedes Element separat berechnet wird. Durch die durchschnittliche Abtastzeit von sechs Sekunden kann kein nachvollziehbarer Cosinus-Verlauf dargestellt werden.

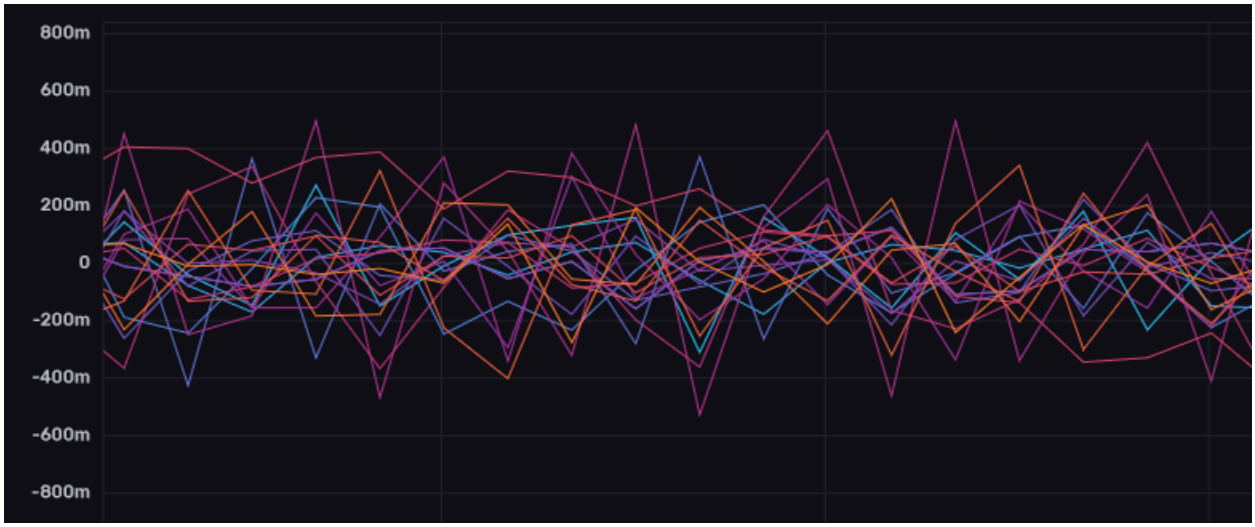

Abbildung 37: Wertdarstellung von „Cosinus“ bei Komponententest, Quelle: Eigene Darstellung.

12.4 Erkenntnisse

Bei den verschiedenen Durchläufen gibt es keine wesentlichen Vorkommnisse, welche auf ein Fehlverhalten der Software hinweisen. Es wird geprüft, ob die Messwerte richtig in die Online-Datenbanken abgelegt werden und ob alle Messwerte in den Online-Datenbanken ankommen.

Bei den Versuchen von vier und zehn Strukturen funktionierte das System fehlerfrei. Bei den Versuchen mit 25 und 60 Strukturen kam es zu Verzögerungen in den Data-Collectors. Im Nachhinein betrachtet ist die Architektur, die alle Messwerte in einer Datenbank zwischenpuffert und die Eigenschaft der SQLite-Datenbank, die es nur einem Prozess erlaubt auf die Datenbank schreibend zuzugreifen eine suboptimale Lösung. Abhilfe schaffen würde eine Architektur, in der für jede Variable eine eigene Datei erstellt würde und somit der Zugriff aufgeteilt ist. Bei OPC UA kann festgestellt werden, dass beim Lesen der 60 Strukturen zwischenzeitlich einige Werte fehlen. Dies liegt aber an der Performance der SPS, da bei einem Test mit einem lokalen OPC UA Server diese Probleme, bei mehreren Variablen, nicht auftreten.

Bei der Datenerfassung über TCP/IP gibt es keine Probleme, wobei hier davon ausgegangen werden kann, dass noch wesentlich mehr Daten erfasst werden könnten.

Obwohl die Puffer bei größeren Datenmengen zu kleineren Verzögerungen führen, erweisen sie sich als äußert nützlich. Jegliche Unterbrechung zwischen Data-Collector und Manager, sowie Deaktivierung der Internetverbindung führen zu keinem Datenverlust.

Der Datenupload zu den Online-Datenbanken funktioniert einwandfrei. Trotz einer für europäische Verhältnisse langsamen Internetverbindung (5 Mbit/s Upload) können alle Daten ohne Verzögerung hochgeladen werden. Das wird auch erwartet, denn die errechnete Datenübertragungsrate für ein Strukturelement beträgt 16 bit/s und es werden 50 Strukturen übermittelt, wodurch sich ein theoretischer Wert von 800 bit/s ergibt. Dazu fallen zwar zusätzlich übertragene Daten an, dennoch werden 5 Mbit/s bei weitem nicht erreicht.

Nachdem das System in einem Langzeittest einige Tage ununterbrochen aktiv war, wurde die gesammelte Datenmenge recht groß. Es ist somit zu hinterfragen, ob sämtliche Daten tatsächlich benötigt werden und ob diese nicht komprimiert werden können.

13 FAZIT

In diesem Kapitel werden die erarbeiteten Ergebnisse zusammengefasst und mit der Zielsetzung verglichen. Abschließend wird ein Ausblick erstellt, wie mit den Ergebnissen und dem System für die Datenerfassung weiter verfahren wird.

13.1 Conclusio

In der Einleitung werden die Anforderungen an das zu entwickelnde System definiert, welche bei der Realisierung des Systems beachtet werden.

Das System ist **verteilt** aufgebaut, die Data-Collectors und Manager mit den DB-Connectors können sich auf unterschiedlichen Geräten befinden. Durch die Kommunikation mit HTTP-Requests kann zwischen den Data-Collectors in einer einfachen Art kommuniziert werden. Ein weiterer Vorteil dieser Kommunikationsart ist die Verwendung des Ports 443, da TLS genutzt wird. Das Freischalten dieses Ports ist auf Bezug der Firewall-Security am unbedenklichsten. Durch den verteilten Aufbau können die Data-Collectors näher an den Geräten sitzen, von denen die Daten gesammelt werden. Das System kann so konfiguriert werden, dass die Daten nicht permanent von den Data-Collectors geholt werden müssen, sondern immer eine gewisse Zeit verstreicht. Damit wird die Netzwerkauslastung minimiert, denn das Übertragen von größeren Datenmengen ist effizienter. Mit diesem System können Probleme umgangen werden, falls Daten mit einem spezielleren Protokoll, z.B. OPC classic, gesammelt werden sollen. Befindet sich der OPC classic Server auf einem Computer, kann der Data-Collector am selben Computer installiert werden. Dadurch fallen beispielsweise die Benutzer-Konfigurationen auf dem Computer weg, auf dem der Data-Collector sonst installiert wäre.

Durch den **modularen** Aufbau können die einzelnen Teile des Systems leichter gewartet und adaptiert werden. Des Weiteren sind auch die einzelnen Programme kleiner, denn es muss nicht der gesamte Source-Code von allen Protokollen oder Datenbanken abgespeichert werden. Dies kann vor allem dann ein großer Vorteil sein, wenn kleine billige Geräte mit wenig Speicherplatz zur Anwendung kommen. In Hinblick auf den Verkauf bietet dieses System gute Voraussetzungen. Es können einzelne Module zu variablen Preisen angeboten werden.

Die fixe Definition von Schnittstellen ermöglicht eine einfache **Erweiterung** verschiedener Module. Wenn alle Schnittstellenfunktionen richtig implementiert werden, kann das neue Modul verwendet werden. In den Modulen sind die Funktionalitäten, welche nichts mit den Protokollen oder Datenbanken zu tun haben, generisch aufgebaut, sodass neue Module diese übernehmen können und nicht neu entwickelt werden müssen. Damit können Neuentwicklungen effizient und kostengünstig realisiert werden.

Durch die Verwendung der Frameworks *.Net Core* und *ASP.Net Core* kann das gesamte System auf den Betriebssystemen Windows, macOS und ausgewählten Linux-Distributionen (Debian, Ubuntu, Raspbian uvm.) betrieben werden und ist somit **plattformunabhängig**. Es muss lediglich vor dem Starten des Programms die *.Net Core Runtime* installiert werden. Vor allem durch die Verwendung von günstigen Linux-Computern können sich erhebliche Kosteneinsparungen durch den Wegfall der im Vergleich teuren Anschaffung von Windows-Computern realisieren lassen.

Auf die **Verschlüsselung** der Daten zwischen den Applikationen wird besonderes Augenmerk gelegt. Dieses Thema ist sehr umfangreich und benötigt für eine sinnvolle Realisierung fundierte Kenntnisse in diesem Bereich. Es ist zwar einfacher, das System ohne derartige Sicherheitsmechanismen zu entwickeln, doch übersteigt der Schaden eines Angriffs deutlich die Kosten der Implementierung. Bei vielen Protokollen sind keine Sicherheitsmechanismen inkludiert, da diese noch aus einer Zeit stammen, in der noch keine so starke Vernetzung gegeben war und diese Protokolle auch nur in abgeschotteten Netzwerken zum Einsatz kamen. Bei neueren Protokollen, wie OPC UA sind Sicherheitsmechanismen standardmäßig verfügbar und es wird empfohlen nur verschlüsselt zu kommunizieren. Daher bietet die Realisierung des Protokolls OPC UA in dieser Arbeit auch die Möglichkeit mit Zertifikaten verschlüsselt zu kommunizieren.

Die realisierten Protokolle decken ein großes Spektrum für die Kommunikation in der Automatisierungstechnik ab. Fast jedes neu ausgelieferte Gerät besitzt bereits standardmäßig einen OPC UA Server. Besitzt ein Gerät keinen OPC UA Server, aber eine Ethernet-Schnittstelle, kann mit dem Protokoll TCP/IP kommuniziert werden. Dieses Protokoll zielt vor allem auf ältere SPSen ab, da die meisten schon seit geraumer Zeit mit solch einer Schnittstelle ausgeliefert werden. Diese Art der Kommunikation ist sehr schlank und performant, wie die durchgeführten Tests gezeigt haben.

Hinsichtlich der Datenbanken sind zwei unterschiedliche Datenbankmodelle umgesetzt. Das Time-Series-Datenbankmodell mit InfluxDB ist performant und bietet in der Onlineumgebung eine gute Visualisierung und eine aufschlussreiche Dokumentation. Die InfluxDB ist unter den Time-Series-Datenbanken die am weitest verbreitete. Dies gilt nicht, wenn alle Datenbankmodelle betrachtet werden (Tabelle 1). Mit der zweiten Datenbank, der Cosmos DB, wurde absichtlich ein Datenbankmodell gewählt, welches dem relationalen Datenbankmodell sehr ähnlich ist. Wie aus Abbildung 7 hervorgeht, sind fast drei Viertel aller realisierten Datenbanken relationale Datenbanken und für solch einen Anwendungsfall, kann die Cosmos DB als Referenz herangezogen werden. Durch den Tabellenaufbau ohne Relationen ist der Performance-Unterschied beim Upload zwar marginal, jedoch ist das Datenhandling mit einer Time-Series-Datenbank besser, da die Daten strukturiert und indexiert abgelegt werden.

Bei den Konfigurationen der Online-Datenbanken muss auf die Einstellungen geachtet werden. Es können sich schnell durch falsche oder zu hoch ausgelegte Einstellungen höhere Kosten entwickeln, als erwartet. Speziell bei der Cosmos DB sind die Einstellungsmöglichkeiten sehr umfangreich und für einen Laien kaum überschaubar. Bei dem Test mit der Cosmos DB wird auch ein langsamerer Upload-Zyklus verwendet, um die zu transferierenden Daten auf einen längeren Zeitraum zu verteilen. Werden die Daten schnell auf einmal hochgeladen (z.B., wenn Data-Collectors über Nacht gelaufen sind und der Manager die Daten nicht geholt hat) entstehen für kurze Zeit hohe Datendurchsätze.

Die Konfiguration mittels Konfigurationsdateien erweist sich als äußerst einfach. Wie festgestellt wird, erfolgt die Haupt-Konfiguration nur einmal und danach werden nur kleine Änderungen vorgenommen. Im laufenden Betrieb läuft der Prozess im Hintergrund und für die Datensammlung wird keine grafische Oberfläche benötigt. Um den Zustand der einzelnen Applikationen zu prüfen, können die Konsolenfenster oder Log-Dateien betrachtet werden. Des Weiteren können mittels SQLite-Clients die Pufferdatenbanken im Data-Collector und DB-Connector manipuliert werden.

Das System kann in vielen Bereichen der Industrie und somit nicht nur in der Automatisierungstechnik eingesetzt werden. Durch die Erweiterbarkeit können alle Protokolle realisiert werden, welche gewünscht werden. So kann das System auch adaptiert werden, um Daten eines Simulationsprograms in einer Cloud abzulegen. Jeder Data-Collector benötigt nur wenig Ressourcen und kann somit auch auf kleinen Rechnern installiert und betrieben werden. Für den Schaltschrankbau werden kostengünstige Hutschienencomputer angeboten, welche verwendet werden können, um die Daten von einem Gerät zu erfassen und an den Manager weiterzuleiten. Sind bereits Computer vorhanden, können die Applikationen auf diesen betrieben werden, wodurch keine zusätzliche Hardware angeschafft werden muss. Der Computer, auf dem der Manager und die DB-Connectors betrieben werden, benötigt eine Internetverbindung. Das System kann so entworfen werden, dass nur der Computer mit Manager und DB-Connectors eine Internetverbindung hat, beispielsweise mit einer separaten Schnittstelle. Werden bei diesem Computer alle notwendigen Sicherheitsrichtlinien (Firewall, Updates, usw.) richtig verwaltet, dann besteht für das Anlagennetzwerk eine sehr geringe Gefahr der Kompromittierung. Die Pufferung der Daten im Data-Collector und DB-Connector ist für die Automatisierungstechnik ein großer Vorteil. In Anlagen kann es vorkommen, dass Geräte zwischenzeitlich nicht verfügbar sind. Sind Data-Collectors und Manager auf unterschiedlichen Geräten, hat ein Ausfall eines Data-Collector-Gerätes nur die Folge, dass keine Daten dieses Gerätes gesammelt werden. Ein häufigeres Problem sind die Netzwerkausfälle. Auch diesen Problemen wird entgegengewirkt indem die Daten solange gespeichert werden, bis das Partnergerät wieder verfügbar ist. Damit entsteht eine hohe Datensicherheit und auch der Ausfall der Internetverbindung, etwa in entlegeneren Gegenden, kann das System nicht in die Knie zwingen.

In Kapitel 5.3 wird die generelle Verwendung von Cloud-Computing in der Industrie behandelt. Daraus erschließt sich, dass das Cloud-Computing in der Industrie zur Anwendung kommen kann, aber jeder Anwendungsfall separat geprüft werden muss.

Um die Daten aus den Online-Datenbanken zu verwenden gibt es mehrere Möglichkeiten. Programme wie Power BI oder Grafana können direkt eine Datenbankanbindung realisieren, wodurch keine Programmierkenntnisse benötigt werden. Sollen die Daten direkt in der Cloud weiterverwendet werden, bieten die meisten Cloud-Hoster Tools für die Verbindungsherstellung an. Es können jedoch auch eigene Applikationen entwickelt werden, welche auf die Datenbank zugreifen. Hier sind vor allem Webanwendungen und APIs zu erwähnen.

Durch die Eigenentwicklung des Systems kann alles selbst definiert werden. Es könnte auch der Source-Code für eine spezielle Anwendung geändert werden, wovon aber abzuraten ist. Durch die fixierten Schnittstellen können zukünftig einfach neue Protokolle oder Datenbankanbindungen realisiert werden. Die Realisierung lässt auch zu, dass beispielsweise für ein Protokoll oder Datenbank mehrere Module entwickelt werden. So könnte ein Modul für eine Datenbankanbindung mit wenigen und eines mit mehreren Parametern entwickelt werden. Je nach Anwendungsfall könnte zwischen den beiden Modulen gewählt werden.

13.2 Ausblick

Die nächsten Schritte für das System sind die Erstellung eines Handbuchs für die Verwendung und die Implementation des Systems in einer Anlage. Das Handbuch soll so gestaltet werden, dass auch Laien in der Lage sind das gesamte System zu konfigurieren. Bei der Implementierung sollen Daten von drei verschiedenen Geräten erfasst werden und in eine Cosmos DB gespeichert werden. Nach einer Testphase soll das System nochmals evaluiert werden und bei positivem Ausgang in weiteren Anlagen verbaut werden.

Während der Realisierung und des Testens sind zwei sinnvolle Features aufgetaucht, welche zeitnah realisiert und auch in der Anlage getestet werden sollen. Zum einen soll bei den Data-Collectors eine Funktionalität implementiert werden, wo ausgewählt werden kann, dass nur Daten weitergegeben werden, wenn sich der erfasste Messwert ändert. Dies würde bei Messwerten, welche die meiste Zeit denselben Wert haben, eine große Datenreduktion bedeuten. Des Weiteren sollen bei den DB-Connectors Upload-Zeiten implementiert werden. Damit kann definiert werden, dass nur zu gewissen Zeiten die Daten an die Online-Datenbanken transferiert werden dürfen. Besonders relevant ist diese Funktion bei begrenzten Upload-Ressourcen eines Unternehmens.

Bei dem Protokoll TCP/IP soll als nächster Schritt eine Datenkompression realisiert werden. Nach der ersten Entwicklung werden für jede Variable vier Byte reserviert. Sollen viele einzelne Bits (Boolean) übertragen werden, resultiert daraus eine große Datenmenge bei kleiner Nutzmenge. Die Konfigurationsdatei soll dahin angepasst werden, dass nicht mehr die Position angegeben wird, sondern die genaue Bit-Position. Damit können Bits nacheinander gepackt werden und es ist eine genaue Zuordnung der Länge mit den Datentypen möglich.

LITERATURVERZEICHNIS

Gedruckte Werke (15)

Baun, Christian (2017): *Betriebssysteme kompakt*, Springer Vieweg, Berlin

Beutelspacher, Albrecht; Schwenk, Jörg; Wolfenstetter, Klaus-Dieter (2015): *Moderne Verfahren der Kryptographie*, 8. Auflage, Springer Spektrum, Wiesbaden

Bühler, Peter; Schlaich, Patrick; Sinner, Dominik (2019): *Datenmanagement*, Springer Vieweg, Berlin

Fielding, Roy (2020): *Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, USA

Gadatsch, Andreas; Mangiapane, Markus (2017): *IT-Sicherheit*, Springer Vieweg, Wiesbaden

Grassmuck, Volker (2004): *Freie Software*, 2. Auflage, Bundeszentrale für politische Bildung, Bonn

Hillar, Gastón (2017): *MQTT essentials*, 1. Auflage, Packt Publishing, Birmingham, UK

IHS Markit Ltd.; OMDIA; Point Topic (2020): *Broadband Coverage in Europe 2019*, European Commission, Luxenburg

Marinescu, Dan (2018): *Cloud computing*, 2. Auflage, Morgan Kaufmann, Cambridge, US

Meier, Andreas (2017): *Werkzeuge der digitalen Wirtschaft*, 1 Auflage, Springer Vieweg, Wiesbaden

Sauter, Thilo (2007): *Fieldbus Systems: History and Evolution*, in: Zurawski, Richard (Hrsg.): *Integration technologies for industrial automated systems*, CRC Taylor & Francis, Boca Raton, FL

Schnell, Gerhard; Wimann, Bernhard (2019): *Bussysteme in der Automatisierungs- und Prozesstechnik*, 9. Auflage, Springer Vieweg, Wiesbaden, Germany

Sen, Sunit (2014): *Fieldbus and networking in process automation*, CRC Press/Taylor and Francis Group, Boca Raton

Türker, Can; Saake, Gunter (2005): *Objektrelationale Datenbanken: ein Lehrbuch*, 1. Auflage, dpunkt.verlag, Heidelberg

Volz, Michael (2010): *Vergleich der Feldbussysteme*, in: (Hrsg.): *Industrielle Kommunikation mit Feldbus und Ethernet*, VDE-Verlag, Berlin

Wendzel, Steffen (2018): *IT-Sicherheit für TCP*, 1. Auflage, Springer Vieweg, Wiesbaden

Wirdemann, Ralf (2019): *RESTful Go APIs*, 1. Auflage, Hanser, München

Wissenschaftliche Artikel (2)

Siepmann, David (2016): *Einführung und Umsetzung von Industrie 4.0*, in: Industrie 4.0 – Technologische Komponenten, Springer Gabler, Berlin

Wollschlager, Martin; Sauter, Thilo; Jasperneite, Jürgen (2017): *The Future of Industrial Communication*, in: IEEE Industrial Electronics Magazine, 11/2017, IEEE
https://ieeexplore.ieee.org/abstract/document/7883994

Konferenzbeiträge (2)

Dillon, Tharam; Wu, Chen; Chang, Elizabeth (2010): *Cloud Computing: Issues and Challenges*, in: *2010 IEEE 24th International Conference on Advanced Information Networking and Applications*, IEEE, Perth, WA, Australia

Kumar, K.; Srividya; Mohanavalli, S. (2017): *A performance comparison of document oriented NoSQL databases*, in: *IEEE International Conference on Computer, Communication, and Signal Processing (ICCCSP-2017)*, IEEE, Chennai, India

Online-Quellen (24)

solid IT (2020): *DB-Engines.com*
https://db-engines.com/de/ranking_categories [Stand: 07.09.2020]

ParkMyCloud (o. J.): *Cloud Pricing Comparison in 2020*
https://www.parkmycloud.com/cloud-pricing-comparison [Stand: 12.09.2020]

Open Source Initiative (2007): *The Open Source Definition*
https://opensource.org/osd-annotated [Stand: 21.09.2020]

Rockwell Automation (2018): *Rockwell Automation Adds OPC UA Support to Its Product Portfolio*
https://ir.rockwellautomation.com/press-releases/press-releases-details/2018/Rockwell-Automation-Adds-OPC-UA-Support-to-Its-Product-Portfolio/ [Stand: 04.10.2020]

Siemens AG (o. J.): *OPC UA – Open Platform Communications Unified Architecture*
https://new.siemens.com/global/de/produkte/automatisierung/industrielle-kommunikation/opc-ua.html [Stand: 04.10.2020]

B&R (o. J.): *B&R-Lösungen mit OPC UA*
https://www.br-automation.com/de-at/technologie/opc-ua/br-loesungen-mit-opc-ua/ [Stand: 04.10.2020]

OPC Foundation (o. J.): *OPC Foundation*
http://opcfoundation.github.io/UA-.NETStandard/ [Stand: 11.10.2020]

OPC Foundation (o. J.): *Unified Architecture*
https://opcfoundation.org/about/opc-technologies/opc-ua/ [Stand: 16.09.2020]

Securai GmbH (o. J.): *Sichere SSL-/TLS-Konfiguration*
https://www.securai.de/veroeffentlichungen/blog/sichere-ssl-tls-konfiguration/ [Stand: 20.08.2020]

InfluxDB (o. J.): *InfluxDB Documentation*
https://docs.influxdata.com/influxdb/v2.0/ [Stand: 07.09.2020]

Stackscale (2020): *Main cloud service models: SaaS, PaaS and IaaS*
https://www.stackscale.com/blog/cloud-service-models/ [Stand: 09.09.2020]

abas Software GmbH (2017): *Mehr Akzeptanz von Cloud-Angeboten im ERP-Bereich*
https://abas-erp.com/de/news/abas-anwenderstudie-2017-mehr-akzeptanz-von-cloud-angeboten-im-erp-bereich [Stand: 11.09.2020]

National Institute of Standards and Technology (2015): *NIST Policy on Hash Functions* https://csrc.nist.gov/projects/hash-functions/nist-policy-on-hash-functions [Stand: 16.10.2020]

Aunkofer, Benjamin (2010): *Der Wirtschaftsingenieur* https://www.der-wirtschaftsingenieur.de/index.php/open-system-interconnection-referenzmodell/ [Stand: 17.08.2020]

Burke, Thomas (2018): *OPC Foundation* https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-DE.pdf [Stand: 19.08.2020]

De Roni, Micro (2019): *RESTful API* https://www.mircoderoni.ch/2019/08/06/restful-api/ [Stand: 17.09.2020]

Drolshagen, René (2015): *Hochschule RheinMain* https://wwwvs.cs.hs-rm.de/downloads/extern/pubs/thesis/drolshagen15.pdf [Stand: 25.09.2020]

Gronau, Norbert (2020): *Enzyklopaedie der Wirtschaftsinformatik* https://www.enzyklopaedie-der-wirtschaftsinformatik.de/lexikon/informationssysteme/Sektorspezifische-Anwendungssysteme/cyber-physische-systeme/industrie-4.0/?searchterm=industrie%204.0 [Stand: 31.7.2020]

Lahoz, Vicente (2019): *SecurityArtwork* https://www.securityartwork.es/2019/02/01/el-protocolo-mqtt-impacto-en-espana/ [Stand: 17.09.2020]

Rinaldi, John (2019): *Why use OPC UA instead of a RESTful interface?* https://www.maintworld.com/Partner-Articles/Why-use-OPC-UA-instead-of-a-RESTful-interface [Stand: 25.09.2020]

Rothhöft, Michaela (2020): *Marktstudien.org* http://www.marktstudien.org/marktstudien/marktstudie-industrielle-kommunikation/ [Stand: 18.09.2020]

Stallmann, Richard (2020): *Warum „Open Source" das Ziel Freie Software verfehlt* https://www.gnu.org/philosophy/open-source-misses-the-point.html.en [Stand: 21.09.2020]

Stauffer, Herbert (2020): *Business Application Research Center* https://barc.de/Artikel/loesungen-zeitreihen [Stand: 03.11.2020]

Stribos, Tim (2019): *Understanding Open-Source and Free Software Licensing* https://moqod.com/2019/09/23/understanding-open-source-and-free-software-licensing/ [Stand: 20.09.2020]

Normen (1)

International Electrotechnical Commission (Hrsg.) (2003): *IEC 61158-2*

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

AES	Advanced Encryption Standard
API	Application programming interface
CORS	Cross-Origin Resource Sharing
DB	Datenbank
DBaaS	Database as a Service
DBMS	Datenbankmanagementsysteme
DES	Data Encryption Standard
DLL	Dynamic Link Library
EMV	Elektromagnetische Verträglichkeit
ERP	Enterprise Resource Planning
HMI	Human Machine Interface
IaaS	Infrastructure as a Service
IoT	Internet of Things
ISO	International Standards Organization
JSON	JavaScript Object Notation
MAC	Message Authentication Code
MES	Manufacturing Execution System
MQTT	Message Queuing Telemetry Transport
NGA	Next Generation Access
OSI	Open Systems Interconnection
PaaS	Platform as a Service
PRNG	Pseudo Random Number Generator
RAMI	Referenzarchitekturmodell Industrie
REST	Representational state transfer
RFC	Request for Comments
SaaS	Software as a Service
SCADA	Supervisory Control and Data Acquisition
SHA	Secure Hash Algorithm
SPS	speicherprogrammierbare Steuerung
SSL	Secure Socket Layer

SQL	Structured Query Language
TSL	Transport Layer Security
URI	Uniform Ressource Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
VM	Virtuelle Maschine
XML	Extensible Markup Language
YAML	YAML Ain’t Markup Language