

Berufliche Schule Direktorat 2  
Rudolf-Diesel-Fachschule Nürnberg  
Informatiktechnik



## PROJEKTARBEIT

# DATENBANK- UND WEBANBINDUNG EINER VIRTUELLEN FABRIK MITTELS OPC UA

Verfasser:

**Michael Mölle, Ondrej Hruby**

Die Arbeit wurde betreut von:

**Dr. Markus Hofmann**

Abgabetermin:

**09.03.2020**

---

---

## **Erklärung**

Hiermit versichern wir, die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet zu haben.

Nürnberg, den 17. Februar 2020

---

Michael Mölle, Ondrej Hruby

## Aufgabenstellung

Die Aufgabenstellung für die vorliegende Arbeit war die selbstständige Auswahl und Durchführung eines Projektes im Rahmen der Ausbildung zum staatlich geprüften Techniker für Informationstechnik. Neben dessen Realisierung soll vor allem der komplette Projektmanagement- und Planungsprozess umgesetzt werden. Hierfür sind im Vorfeld eine Machbarkeitsanalyse, ein Projektstrukturplan sowie eine Zeitplanung zu erstellen. Der gesamte Projektverlauf soll in Form einer Dokumentation festgehalten werden.

Aufgrund dieser Aufgabenstellung und der immer größer werdenden Bedeutung der Industrie 4.0, kam es zu der Idee sich mit dem Thematik OPC-UA und der Datenauswertung virtueller Fabriken auseinanderzusetzen. Vor allem die Möglichkeiten sich Wissen und Praxiserfahrung rund um die Punkte Netzwerktechnik, 3D-Simulationssoftware und node-RED anzueignen waren letztendlich ausschlaggebend für die Wahl des Projektthemas „Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC-UA“.

## **Danksagungen**

An dieser Stelle möchten wir uns bei allen bedanken, die uns während der Projekt-durchführung unterstützt haben.

Vor allem gebührt unser Dank unserem Projektbetreuer Herrn Dr. Hofmann, der während der Projektarbeit als technischer Ansprechpartner zur Verfügung stand. Für die hilfreichen Ratschläge und die Motivierung bezüglich der Wahl des Projektes, stetige Erweiterungsvorschläge zur Durchführung und nicht zuletzt für die unkomplizierte Kommunikation möchten wir uns hiermit herzlich bedanken.

Weiterhin möchten wir der Firma Visual Components für die einjährige Bereitstellung ihrer 3D-Simulationssoftware danken, ohne die eine Realisierung des Projektes in dieser Form nicht möglich gewesen wäre.

Zuletzt möchten wir uns noch bei allen Freunden und unseren Familien bedanken, die uns während der Erstellung dieser Arbeit und der Projektumsetzung unterstützt haben.

Michael Mölle, Ondrej Hruby

Nürnberg, 17. Februar 2020

## Vorwort

Angekommen im Jahre 2019 ist das Thema „Industrie 4.0“ von mehr Bedeutung denn je. „Industrie 4.0“ bezeichnet die intelligente Vernetzung von Maschinen und Abläufen in der Industrie mit Hilfe von Informations- und Kommunikationstechnologie.<sup>1</sup> Um diese intelligente Vernetzung zu realisieren, ist ein gemeinsamer Kommunikationsstandart nicht wegzudenken. OPC Unified Architecture (kurz OPC-UA) ist ein Standard für den Datenaustausch als plattformunabhängige, service-orientierte Architektur. Dieser ermöglicht es, unabhängig von der Art der Maschine oder dem Rechensystem, dass diese uneingeschränkt bidirektional miteinander kommunizieren können, was vor allem für den Austausch und die Weiterverarbeitung von maschinellen Nutzdaten ein gravierender Vorteil ist.

Mit Hilfe der 3D-Simulationssoftware „Visual Components“ lassen sich einzelne Produktionsabschnitte, aber auch gesamte Fabriken abbilden. Somit kann im Vorfeld mit virtuellen lizenzierten Maschinen und echten physikalischen Werten die gesamte Produktionslinie einer Fabrik geplant, entworfen und die Kommunikation unter den Maschinen getestet werden.

Eine vorher bereits entworfene Limonaden-Abfüll-Fabrik dient als Vorlage, um vorhandene Akteure anzusteuern und die Daten der ebenfalls vorhandenen Sensoren mittels dem OPC-UA Standard zu erfassen.

Grundgedanke dieses Projektes war es über den Standard OPC UA die virtuelle Limonaden-Abfüllfabrik von Visual Components im vollen Funktionsumfang bidirektional steuern und die Daten ihrer Sensoren in einer relationalen Datenbank lagern und später über ein Dashboard einer Website auswerten zu können.

Mit der Software „OPC Router“, welche eine zentrale Kommunikationsplattform für OPC-Projekte in der Industrie darstellt, wird eine Vernetzung des Servers, der Simulationssoftware und einer relationalen Datenbank zur Pflege der erfassten Sensordaten realisiert.

---

## Abstract

A database and web connection of a virtual factory using OPC UA means the possibility to store and manage the captured data of the sensors, such as time stamps or produced quantities, which are modeled according to the OPC UA data model, in a relational database. In addition, the web connection and the fact that OPC UA works bidirectionally allows the user to modify the database or to record virtual work processes of the factory via various actuators.

Whereas in the past, problems in production could usually only be determined on site, nowadays 3D simulation software such as Visual Components can be used to test production machines and simulate complete production lines or factories, thus minimizing and eliminating potential sources of error in advance. Problems usually arose when a machine from one manufacturer tried to establish connectivity with a machine from another manufacturer. Thanks to the OPC UA standard, nothing now stands in the way of unrestricted communication between the various production machines, databases, servers and employee computers. This standard can also be implemented in Visual Components on machines in a factory and then connected to a corresponding server and databases.

## Inhalt

Erklärung (MM) .....	3
Aufgabenstellung (MM) .....	4
Danksagungen (MM).....	5
Vorwort (MM) .....	6
Abstract (MM).....	7
Inhalt .....	8
1    Einleitung (MM) .....	11
1.1  Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC-UA (MM) ...	11
1.2  Projektbeschreibung (MM) .....	11
1.3  Projektziel (MM) .....	12
2    Projektplanung (MM) .....	13
2.1  Machbarkeitsanalyse (MM) .....	13
2.2  Soll-Analyse (MM) .....	13
2.3  Zeitplanung (MM, OH).....	13
3    Verwendete Komponenten (MM).....	14
3.1  Vagrant und Oracle VM VirtualBox (MM).....	14
3.2  OPC Unified Architecture (MM) .....	14
3.2.1  OPC UA C++ Demo-Server (MM) .....	16
3.2.2  UaExpert (MM) .....	17
3.3  Visual Components (OH) .....	17
3.3.1  Die virtuelle Limonaden-Abfüllfabrik (OH).....	17
3.4  OPC Router (MM) .....	18
3.5  Relationale Datenbank (MM) .....	18
3.5.1  MySQL Workbench (MM) .....	18
3.5.2  phpMyAdmin (MM) .....	18

3.6	node-RED (OH).....	18
3.7	Interne Kommunikation (MM) .....	19
3.7.1	GitHub (MM).....	19
3.7.2	SharePoint (MM) .....	19
4	Projektumsetzung (MM) .....	20
4.1	Inbetriebnahme und Konfiguration einer virtuellen Arbeitsumgebung (MM).....	20
4.2	OPC-UA-Server (MM) .....	22
4.2.1	Erstellen eines einfachen OPC-UA-Server in JavaScript und node.js (MM) .....	22
4.2.2	Verbindungsauflaufbau und die daraus resultierende Problematik (MM).....	30
4.2.3	Konfiguration und Inbetriebnahme des OPC-UA-Demo-Servers (MM) .....	33
4.2.4	Serverhierarchie (MM).....	34
4.2.5	Verwendete Pub-/ Sub-Ziel-Variablen (MM) .....	34
4.3	Inbetriebnahme der Limonaden-Abfüllfabrik in Visual Components (MM) .....	36
4.3.1	Konzeption eines Addons zum Auswerten von Benachrichtigungen (MM) .....	36
4.3.2	Konzeption eines Volumensensors zum Erfassen der Flaschenstückzahl(MM) .	38
4.3.3	Sensoren und ihre zu erfassenden Daten (OH).....	41
4.3.4	Aktoren und ihre Funktionalitäten (OH) .....	44
4.3.5	Konnektivität zum OPC UA C++ Demo-Server (OH) .....	46
4.4	Erstellung einer relationalen Datenbank (MM).....	49
4.4.1	Konfiguration der Datenbank und Sicherung der Datenintegrität (MM) .....	49
4.5	Installation und Konfiguration des OPC Router (MM) .....	53
4.5.1	Konfiguration des OPC-UA-Plug-ins (MM) .....	53
4.5.2	Konfiguration des MySQL-6-Plug-ins (MM) .....	54
4.5.3	Konfiguration der Verbindung „StorageRequest_Produziert“ mit Trigger (MM) ...	54
4.5.4	Konfiguration der Verbindung „StorageRequest_Defekt“ mit Trigger (MM).....	57
4.5.5	Produktivschaltung und Test der Verbindung (MM).....	59
4.6	Installation und Konfiguration von node-RED (OH) .....	60
4.6.1	Konfiguration der nodes (OH).....	61

4.6.2 Konfiguration der Verbindung zu Visual Components (OH) .....	62
4.6.3 Konfiguration der Verbindung zur relationalen Datenbank (OH) .....	64
4.6.4 Erstellung eines Dashboards für Bestellung und Bedienung (OH) .....	66
5 Bewertung des Projektprozesses (OH) .....	72
5.1 Reflexion der Machbarkeitsanalyse (OH) .....	72
5.2 Reflexion des Projektstrukturplans (OH) .....	73
5.3 Reflexion der Zeitplanung (OH) .....	73
5.4 Reflexion der Projektziele (OH) .....	73
6 Erkenntnisse (OH) .....	74
6.1 Ausblick (OH) .....	75
Glossar .....	76
Literaturverzeichnis .....	77
Abbildungsverzeichnis .....	79
Quellcodeverzeichnis .....	80
Tabellenverzeichnis .....	81
Anlagenverzeichnis .....	82
Anhang A (Projektmanagement) .....	82
A.1 Machbarkeitsanalyse .....	82
A.2 Projektvereinbarung .....	83
A.3 Projektstrukturplan .....	87
A.4 Zeitplanung .....	88
A.5 Meilensteinprotokoll Nr. 1 vom 25.10.2019 .....	89
A.6 Meilensteinprotokoll Nr. 2 vom 19.12.2019 .....	92
A.7 Meilensteinprotokoll Nr. 3 vom .....	95
A.8 Tätigkeitsnachweise .....	95
Anhang B (Projektmaterialien) .....	107

# 1 Einleitung

Die vorliegende Projektarbeit dokumentiert den Verlauf zum Projekt „Datenbank- und Webanbindung einer virtuellen Fabrik“.

Ziel dieser Dokumentation ist es den Projektverlauf und alle Schritte des genannten Projektes von der Planung bis hin zum Abschluss, über die Einrichtung einer internen Kommunikation mittels diversen Tools und Austauschplattformen festzuhalten und durch geeignete Visualisierungen zu unterstützen.

## 1.1 Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC-UA

Unter einer Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC-UA versteht man die Möglichkeit, die erfassten Daten der Sensoren, wie beispielsweise Zeitstempel oder produzierte Stückzahlen, welche nach dem OPC-UA-Datenmodell modelliert sind, in einer relationalen Datenbank speichern und verwalten zu können. Außerdem hat man mit der Webanbindung und der Tatsache, dass OPC-UA bidirektional arbeitet, die Möglichkeit Bestände der Datenbank zu modifizieren oder über diverse Akten virtuelle Arbeitsprozesse der Fabrik aufzunehmen.

Während in der Vergangenheit Probleme bei der Produktion meist erst vor Ort festgestellt werden konnten, kann man heutzutage mit 3D-Simulationssoftware, wie Visual Components, Produktionsmaschinen testen und komplett Produktionslinien oder Fabriken simulieren und somit im Vorfeld potenzielle Fehlerquellen minimieren und beseitigen. Problematiken entstanden meist dann, wenn eine Maschine eines Herstellers versuchte Konnektivität mit einer Maschine eines anderen Produzenten herzustellen. Dank dem Standard OPC-UA steht nun einer uneingeschränkten Kommunikation zwischen den verschiedenen Produktionsmaschinen, Datenbanken, Servern und Mitarbeiterrechnern nichts mehr im Wege. Der besagte Standard lässt sich auf Maschinen einer Fabrik in Visual Components implementieren und diese dann mit einem entsprechenden Server und Datenbanken verbinden.

## 1.2 Projektbeschreibung

Ein in der Industrie oft auftretendes Problem ist die nicht garantierter reibungslose Interoperabilität zwischen Produktionsmaschinen und Anlagen im informationstechni-

schen Bereich. So können trotz sorgfältiger Planung im Vorfeld dennoch im Nachhinein Probleme hinsichtlich der maschinellen Kommunikation auftreten. So war es in der Vergangenheit nicht selbstverständlich, dass jeder Maschinenhersteller sich an ein einheitliches Programmier- und Treiberschema hielt. Viele verschiedene Hardwaretreiber auf dem Markt verlangen den Einsatz von Adapters und Wrappern. Dies führt unter Umständen zu zeitlichem Mehraufwand und im schlimmsten Fall zu finanziellen Einbußen.

OPC-UA, ein Standard für den Datenaustausch als plattformunabhängige, serviceorientierte Architektur, kann für die beschriebene Problematik Abhilfe schaffen. Der Standard ermöglicht es, dass nahezu alle informationstechnischen Geräte, Produktionsmaschinen und auch Abwicklungssysteme bidirektional miteinander kommunizieren können. Dies wird realisiert, indem Daten maschinenlesbar semantisch beschrieben werden. Das bedeutet, dass eine Maschine, unabhängig von ihrem Hersteller, mit den Daten anderer Maschinen arbeiten kann und umgekehrt.

3D-Simulationssoftware bietet die Möglichkeit, Produktionsvorgänge und sogar ganze Fabriken im Vorfeld darzustellen und mit Hilfe von echten physikalischen Werten zu testen. Mit Hilfe des oben beschriebenen Standards lassen sich Datenbanken und andere Unternehmensnetzwerke einbinden. Die zentrale Kommunikationseinheit stellt dabei der OPC-UA-Server dar. Mit Hilfe des OPC Routers lassen sich alle Komponenten miteinander verbinden.

So werden Daten, die mit Sensoren einer virtuellen Limonaden-Abfüllfabrik erfasst werden, über den OPC-UA-Server in eine relationale Datenbank eingepflegt. Somit können Simulationsdaten in der echten Welt verwertbar gemacht werden. Parallel dazu wird die Möglichkeit bestehen, Akteure über ein mit node-RED implementiertes Dashboard, ansteuern zu können. Somit kann über einen Webshop eine Bestellung an die virtuelle Fabrik übergeben werden.

### 1.3 Projektziel

Das Ziel des Projektes ist es für die virtuelle Limonaden-Abfüllfabrik in der Software Visual Components eine Web- und Datenbankanbindung mittels OPC-UA zu erschaffen. Akteure sollen angesteuert und deren Funktionen ausführen können und Produktionsdaten sollen von Sensoren erfasst und in einer relationalen Datenbank eingepflegt werden.

Innerhalb einer virtuellen Arbeitsumgebung soll der OPC-UA-Demo-Server betrieben und über den OPC Router auf dem Hostsystem mit allen Clients verbunden werden. Die MySQL-Datenbank soll die Anzahl der produzierten Flaschen, deren Abfülldatum und eventuell auftretende Defekte aufnehmen. Die Ansteuerung der Akteure soll über ein Dashboard, welches mit node-RED konzipiert wird, realisiert werden. Ebenfalls sollen

potenzielle Kunden die Möglichkeit haben über die Webanbindung Bestellungen aufzugeben zu können.

## 2 Projektplanung

In diesem Kapitel findet eine Erläuterung der Planung des Projektes, bestehend aus Soll-Zustand, Machbarkeitsanalyse und Zeitplanung, statt.

### 2.1 Machbarkeitsanalyse

Bevor mit der eigentlichen Projektumsetzung begonnen werden kann, muss vorab die Kompatibilität der verwendeten Komponenten untereinander geprüft werden. Insbesondere ist hierbei der Zugang zu lizenpflichtiger Software und deren Operabilität untereinander zu beachten. Eine ausführliche Machbarkeitsanalyse findet sich im Anhang A.1.

### 2.2 Soll-Analyse

Um das Projekt entsprechend der Projektvereinbarung erfolgreich abzuschließen, müssen Lizenzen für Software organisiert und diese auf den Hostmaschinen identisch konfiguriert werden. In der 3D-Simulationsanwendung Visual Components befindet sich ein Layout, welches eine Limonaden-Abfüllfabrik abbildet. In dieser fiktiven Fabrik sind diverse Aktoren und Sensoren vorhanden. Ziel ist es, die Aktoren anzusteuern und Sensordaten mittels OPC-UA zu gewinnen. Die erfassten Daten werden durch eine Anbindung in eine Datenbank eingepflegt und die bidirektionale Steuerung der Füllanlage über eine Webanbindung mittels node-RED realisiert. Eine genauere Erklärung der einzelnen Werkzeuge und der zugehörigen Programmiersprachen befindet sich in Kapitel 3.

### 2.3 Zeitplanung

Bevor mit der Durchführung des Projektes begonnen werden kann, muss anhand der im Projektstrukturplan definierten Arbeitspakete eine Zeitplanung erstellt werden. Generell gliedert sich das Projekt in die drei Phasen Projektplanung, Projektdurchführung und Projektabschluss. Die Arbeitspakete wurden entsprechend der Terminierung der Meilensteinsitzungen aufgeteilt. Beide Projektleiter haben ein Arbeitspensum von 240 Stunden. Eine detaillierte Aufschlüsselung der Zeitplanung befindet sich im Anhang unter dem Punkt A.4.

### 3 Verwendete Komponenten

In diesem Kapitel findet eine Erläuterung der eingesetzten Software und Werkzeuge, die zur Umsetzung des Projektes genutzt werden, statt.

#### 3.1 Vagrant und Oracle VM VirtualBox

Bei Vagrant handelt es sich um eine freie Ruby-Anwendung, die zum Erstellen und Verwalten virtueller Maschinen verwendet wird. Entwickelt wurde sie von der Firma Hashi-Corp. Der Fokus des Tools liegt auf der Inbetriebnahme und Verwaltung von virtuellen Maschinen mithilfe eines einzigen Workflows.<sup>[2]</sup> Sie ermöglicht einfaches Deployment insbesondere in der Softwareentwicklung und dient als Schnittstelle zwischen Virtualisierungssoftware und Konfigurationsmanagement. Die Funktionsweise von Vagrant gestaltet sich denkbar einfach und wird mittels eigenen Kommandos über eine Shell gesteuert.

Oracle VM VirtualBox ist eine Virtualisierungssoftware des US-amerikanischen Unternehmens Oracle. Mittels dieser Software ist es möglich auf einem Hostsystem mehrere voneinander unabhängige virtuelle Maschinen mit eigenen Betriebssystemen zu erzeugen und parallel zu betreiben. Diese können dabei, je nach Konfiguration, komplett isoliert oder in Konnektivität mit dem Hostsystem betrieben werden. Die aktuelle Version „VirtualBox 6.1.2 platform packages“ und die Erweiterung mit dem größten Nutzungs-umfang „VirtualBox 1.6.2 Oracle VM VirtualBox Extension Pack“ sind kostenlos auf der Herstellerwebsite erhältlich.

#### 3.2 OPC Unified Architecture

Der Standard Open Platform Communications (kurz OPC) wurde bereits erstmals im Jahre 1996 von der OPC Foundation ins Leben gerufen. „Heute ist er der weltweit am weitesten verbreitete Interoperabilitätsstandard für den sicheren, zuverlässigen und plattformunabhängigen Informationsaustausch, mit über 730 Mitgliedern der OPC Foundation und tausenden von OPC-konformen Produkten.“<sup>[3]</sup> OPC wird da eingesetzt, wo Regler, Steuerungen oder Sensoren verschiedener Hersteller ein gemeinsames Netzwerk bilden. Ohne besagten Standard benötigen zwei Geräte zum Datenaustausch genaue Kenntnis über die Kommunikationsmöglichkeiten des Gegenübers. OPC Unified

Architecture stellt die neueste Generation aller Spezifikationen der Open Platform Communications dar und unterscheidet sich von seinem Vorgänger insbesondere durch die Fähigkeit, Maschinendaten nicht nur zu transportieren, sondern auch maschinenlesbar semantisch beschreiben zu können. Während die vorherige OPC-Version an die COM/DCOM-Schnittstelle gebunden ist und dadurch entscheidende Nachteile entstehen, wurde für OPC-UA ein eigener Kommunikationsstack entwickelt, der besagte Schnittstelle ersetzt. Der Aufbau einer UA-Anwendung, egal ob Server- oder Clientseitig, gliedert sich in folgende Schichten.

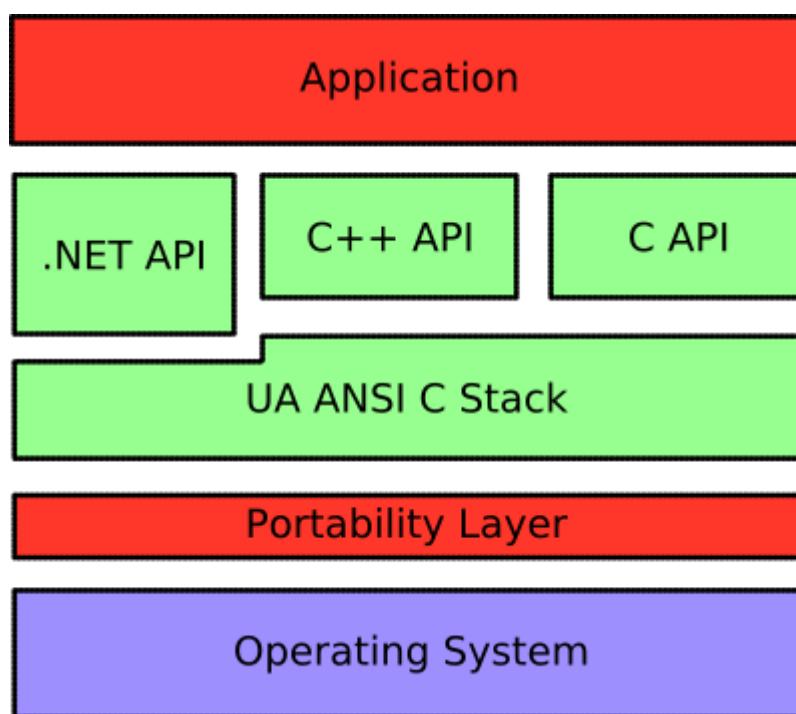


Abbildung 1: OPC-UA Kommunikationsstack

Um Daten mit Hilfe des Kommunikationsstacks auszutauschen, gibt es zwei Mechanismen:

- Ein Client-Server-Modell, bei dem UA-Clients die dedizierten Dienste des UA-Servers nutzen
- Ein Publisher-Subscriber-Modell, bei dem ein UA-Server konfigurierbare Untermengen von Informationen an eine beliebige Anzahl Zuhörer verfügbar macht

Letzterer Mechanismus wird bei der Realisierung des Projektes verwendet. Im Allgemeinen ist die Kommunikation über OPC-UA nach dem Client-Server-Prinzip aufgebaut.

Hierbei stellt ein Server Informationen und Dienste in einem Netzwerk mehreren Clients bereit. Ein Client kann diese von mehreren Servern nutzen. OPC-UA ist eine serviceorientierte, internetfreundliche Architektur, die auf den Grundprinzipien der Bereitstellung einfacher Schnittstellen, eines einheitlichen Nachrichtenformats, flexibler Erweiterungsmöglichkeiten und der Implementierung hoher Sicherheitsstandards und verschiedener Sicherheitslevel, beruht. Dabei setzt sich Unified Architecture aus vielen Spezifikationen zusammen. All diese aufzuzählen würde den Rahmen dieser Projektdokumentation sprengen. Die wichtigsten Spezifikationen für den operativen Einsatz sind:

- Data Access
- Historical Access
- Alarms and Conditions

Um den uneingeschränkten Datenaustausch zwischen herstellerunabhängigen Maschinen realisieren zu können, setzt OPC-UA auf das Prinzip der einheitlichen semantischen Datenmodellierung. OPC-UA definiert ein generisches Objektmodell inklusive dem zugehörigen Typsystem. Zusätzlich zu diesem Datenmodell sind Regeln definiert, die beschreiben, wie man jedes physikalische System in ein UA-konformes Modell transformiert, um es im OPC-Server zu repräsentieren.

### **3.2.1 OPC UA C++ Demo-Server**

Von der Unified Automation GmbH ist ein in der Programmiersprache C++ geschriebener Server zu Demonstrationszwecken auf deren Website zum freien Download verfügbar. Die nodes im Namespace und die gesamte Serverhierarchie basieren auf den Tutorials der gleichnamigen Gesellschaft und repräsentieren letztendlich einen Server zur bidirektionalen Steuerung einer automatisierten Gebäudesteuerung.<sup>[4]</sup> Der Server kann mit Hilfe der Tutorials selbst implementiert werden. Es kann aber eine bereits ausführbare Datei des Servers auf der Website der Unified Automation GmbH heruntergeladen werden. Die bereits konfigurierte Serverhierarchie und die darin enthaltene Verzeichnisstruktur enthält elementare Variablen zum Serverbetrieb im Bereich der Gebäudeautomation. Diese können allerdings mit beliebigen Werten oder Variablen vom selben Datentypen initialisiert werden und somit für anderweitige Zwecke genutzt werden.

### 3.2.2 UaExpert

Bei der Software UaExpert handelt es sich um einen OPC-UA-Client, der ebenfalls wie der Demo-Server von „Unified Automation GmbH“ zur Verfügung gestellt wird. Entwickelt wurde er zu Testzwecken im universellen OPC-UA-Einsatzbereich und unterstützt OPC-UA-Features wie Data Access, Alarms & Conditions, Historical Access und den Aufruf von OPC-UA-Methoden.<sup>[5]</sup> Ebenfalls in der Programmiersprache C++ geschrieben, kann dieser plattformunabhängig betrieben werden. Mit UaExpert kann der Betrieb des OPC-UA-Servers überwacht, auf Alarne und Events reagiert, mittels Monitorings auf Serverdaten zugegriffen, und die Serverperformance überprüft werden.

## 3.3 Visual Components

Die Plattform Visual Components wurde zur Unterstützung fortschrittlicher 3D-Fertigungssimulationsanwendungen entwickelt. Die Software ermöglicht die Simulation und Visualisierung von Funktionen, die durch physikalische Kräfte wie Kollisionen, Schwerkraft und Materialeigenschaften beeinflusst werden. Dabei ist es möglich, Maschinen von echten Herstellern und sogar gesamte Fabriken mit realistischen physikalischen Werten darzustellen.

Darüber hinaus können die Layouts und 3D-Simulation in einer virtuellen Umgebung gesichtet werden. Die Virtual-Reality-Anwendung, bietet den Herstellern und Entwicklern eine neue Möglichkeit, das Produktionslayout in der fiktiven Welt zu inspizieren und die Produktionsplanung zu optimieren.

### 3.3.1 Die virtuelle Limonaden-Abfüllfabrik

Die Abfüllfabrik ist nur eine der vielen programminternen Layouts in Visual Components. Die Flaschen passieren hier mehrere automatisierte Arbeitsstationen. Im ersten Schritt gelangen leere Flaschen mittels Förderbandes in eine rotierende Waschanlage. Als nächstes werden sie in einer Füllmaschine mit Limonade befüllt. Nach der Befüllung werden die Flaschen im dritten Arbeitsschritt zugeschraubt und falls bei der Produktion Defekte aufgetreten sind, aussortiert. In der vorletzten Station werden die Limonadenflaschen mit einem Etikett versehen und schließlich von einem Roboterarm in Kisten verpackt. Die Fabrik bietet eine sehr hohe Konfigurationsvielfalt wie beispielsweise Ausschussquoten, Produktionsgeschwindigkeit, Füllmengen, Auslastungsparameter etc.

## 3.4 OPC Router

OPC Router, von der Firma inray Industriesoftware GmbH, ist eine zentrale Kommunikationsplattform, die eine Konnektivität zwischen den automatisierten Systemen herstellt und einen automatisierten Datenaustausch zulässt. Auf Standardschnittstellen basierend, ermöglicht die Software mit verschiedenen Verbindungsarten, einen Datenaustausch durch alle Automatisierungsebenen hinweg. Er stellt in dem Projekt die zentrale Verbindung zwischen dem OPC-UA-Server, der MySQL-Datenbank und Visual Components her. Der Downloadlink für die Software muss explizit auf der Website des Herstellers angefordert werden.

## 3.5 Relationale Datenbank

Eine relationale Datenbank wird zur Speicherung von Informationen in verschiedenen Tabellen genutzt. Diese stehen in Beziehungen zueinander, womit Redundanzen, Inkonsistenzen und Probleme bei der Aktualisierung mehrfach gespeicherter Datensätze, verhindert werden.

### 3.5.1 MySQL Workbench

MySQL Workbench ist ein grafisches Entwicklungswerkzeug zum Erstellen von Datenbanken, deren Bearbeitung und Verwaltung.<sup>[6]</sup> Die Software eignet sich zum Erzeugen von ER-Modellen und bietet Möglichkeiten für Dokumentationen. Eine kostenlose Community-Version ist auf der Website des Herstellers erhältlich.

### 3.5.2 phpMyAdmin

phpMyAdmin ist ein freies Softwaretool zur Administration von MySQL-Datenbanken über das Internet.<sup>[7]</sup> Implementiert wurde es in PHP, woher sich auch dessen Name ableitet. phpMyAdmin unterstützt eine breite Palette von Operationen auf MySQL und MariaDB. Häufig verwendete Operationen können über die Benutzeroberfläche ausgeführt werden, während weiterhin die Möglichkeit besteht, jede beliebige SQL-Anweisung direkt auszuführen.

## 3.6 node-RED

Node-RED ist ein Programmierwerkzeug, mit dem Hardware-Geräte, APIs und Online-Dienste auf neue und interessante Weise miteinander verbunden werden können.

Es bietet einen browserbasierten Editor, mit dem es einfach ist, Flüsse mit Hilfe der breiten Palette von Knoten zu verkabeln, die mit einem einzigen Mausklick zur Laufzeit bereitgestellt werden können. Node-RED ist ein von IBM entwickeltes grafisches Entwicklungswerkzeug. Die Software ermöglicht es, Anwendungsfälle im Bereich des Internets der Dinge mit einem simplen Baukastenprinzip umzusetzen. Die einzelnen Funktionsbausteine werden durch Ziehen von Linien miteinander verbunden. Eine große Auswahl an mitgelieferten Bausteinen deckt die meisten der gängigsten Dienste und Technologien ab.

## 3.7 Interne Kommunikation

Dieses Kapitel beschreibt die verwendeten Komponenten und Onlineplattformen, welche zur internen Kommunikation genutzt werden. Zu Beginn des Projektes beschränkte sich die Zusammenarbeit lediglich auf persönliche Treffen und elektronischem Schriftverkehr. Da der Projektumfang die Grenzen des E-Mail-Verkehrs sprengt, ist es nötig die Austauschplattform Microsoft SharePoint und die Entwicklerplattform GitHub zu nutzen.

### 3.7.1 GitHub

Ein Repository auf der Entwicklerplattform [www.github.com](https://www.github.com) dient als Ablage- und Austauschverzeichnis für sämtliche Dokumente, Dateien und Quellcodes. Um die gemeinsame Zusammenarbeit zu erleichtern und Interessenten am Projekt Zugriff auf die zugehörigen Dateien zu gewährleisten, wurde das öffentliche Repository am 11.10.2019 erstellt und seitdem bei jeglichen Änderungen und Erweiterungen der Projektdateien aktualisiert.

### 3.7.2 SharePoint

Die Bearbeitung eines Dokumentes durch zwei Personen ist mit viel organisatorischem Aufwand verbunden. Termine müssen vereinbart und wahrgenommen werden, die wiederum mit Anfahrtskosten und zeitlichem Mehraufwand verbunden sind. Die Webanwendung Microsoft SharePoint bietet die Lösungen für die Zusammenarbeit an Projekten. Durch die Erstellung von Team-Websites besteht die Möglichkeit zur Verwaltung von Projekten oder der Koordination von Aufgaben. Content-Management über Dokumentenmanagement-Funktionen erleichtern die gemeinsame Bearbeitung einer Datei.<sup>[8]</sup>

## 4 Projektumsetzung

In diesem Kapitel ist die Umsetzung des Projektes beschrieben, welche sich in vier Teilbereiche abgrenzen lässt. Zum einen die Konfiguration und Inbetriebnahme des OPC-UA-Servers, der Bedienung der Aktoren und Sensoren der virtuellen Limonaden-Abfüllfabrik und zum anderen der Erstellung einer relationalen Datenbank samt ihrer Anbindung an die Infrastruktur. Der letzte Teil besteht daraus, mittels node-RED ein Dashboard für Webservices zu implementieren, welches eine bidirektionale Steuerung der Anlage ermöglicht.

### 4.1 Inbetriebnahme und Konfiguration einer virtuellen Arbeitsumgebung

In diesem Teil der Projektumsetzung ist beschrieben, welche Schritte für die Inbetriebnahme und Konfiguration einer virtuellen Arbeitsumgebung für den Betrieb des OPC-UA-Servers notwendig sind. Zu diesem Zweck sind die Werkzeuge Vagrant und Oracle VM Virtual Box zu verwenden.

Als Betriebssystem für die virtuelle Maschine dient eine Microsoft Windows 10 64-bit-Version, die in Form einer Image-Datei mit Oracle VM VirtualBox installiert wird. Der OPC-UA-Server verbraucht nur geringfügig Speicher, weshalb eine Bereitstellung von 4096 MB RAM völlig ausreichend ist. Um eine reibungslose Bedienung der Benutzeroberfläche zu gewährleisten, ist es von Vorteil 128 MB für Grafikspeicher freizugeben. Damit die virtuelle Arbeitsumgebung die gleichen Netzwerkdienste, wie das Hostsystem nutzen kann, ist eine Netzwerkbrücke in der Konfiguration einzustellen. Sämtliche Konfigurationen am System selbst sind direkt in der grafischen Benutzeroberfläche von Oracle VM VirtualBox Manager über das Zahnradssymbol „Ändern“ vorzunehmen. Dieses befindet sich in der oberen Toolbar links an zweiter Stelle.

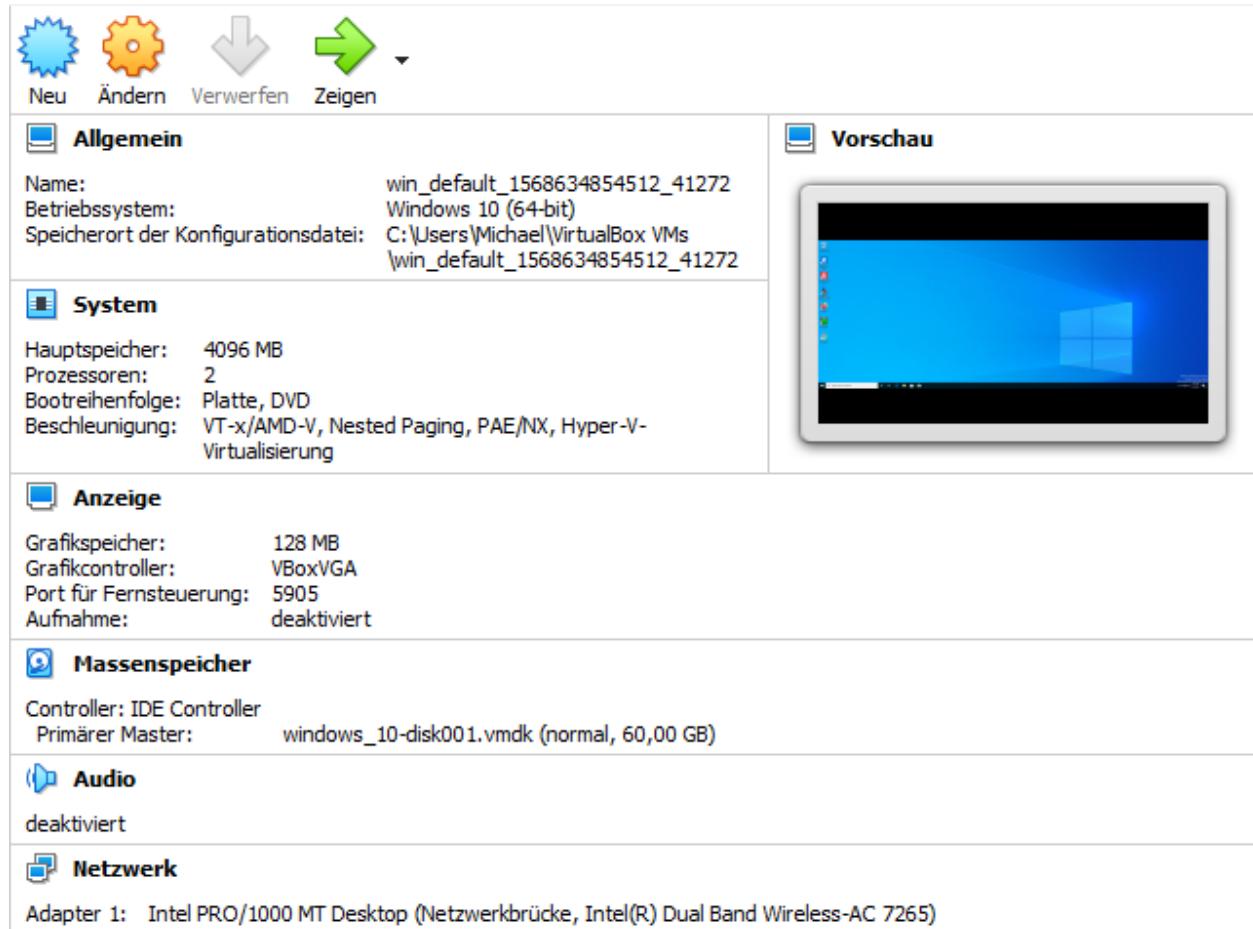


Abbildung 2: Systemkonfiguration der virtuellen Arbeitsumgebung

Für eine uneingeschränkte Kommunikation zwischen dem OPC-UA-Server, der Datenbank und den OPC-UA-Clients, ist in der Konfiguration der Windows Defender Firewall in den erweiterten Einstellungen jeweils eine eingehende und eine ausgehende Regel erstellt. Damit der Server Konnektivität zu Clients auf dem Hostsystem erhält, ist in der Regeleinstellung die beidseitige Verbindung für den Port 48010 zugelassen.

Name	Group	Ena...	Action	Local subnet	Protocol	Local Port	Remote Port	Authorized Users
Atmel Studio		Yes	Allow	I	Any	Any	Any	Any
Atmel Studio - Debug Agent		Yes	Allow	I	Local subnet	Any	Any	Any
Firefox (C:\Program Files (x86)\Mozilla Fi...)		Yes	Allow	I	Any	UDP	Any	Any
Firefox (C:\Program Files (x86)\Mozilla Fi...)		Yes	Allow	I	Any	TCP	Any	Any
Node.js: Server-side JavaScript		Yes	Allow	I	Any	TCP	Any	Any
Node.js: Server-side JavaScript		Yes	Allow	I	Any	UDP	Any	Any
opc ua		Yes	Allow	I	Any	TCP	48010	Any

Abbildung 3: Festlegung der eingehenden Regel

## 4.2 OPC-UA-Server

Die Basis der OPC-Kommunikation stellt der OPC-UA-Server dar. Mit seinen Spezifikationen Data Access, Historical Access und Alarms and Conditions dient er als Knotenpunkt für alle verbundenen OPC-Clients. Dieser Teil der Projektdurchführung beschreibt die Implementierung und Konfiguration des OPC-UA-Servers.

### 4.2.1 Erstellen eines einfachen OPC-UA-Server in JavaScript und node.js

Damit die erfassten Sensordaten und Funktionalitäten der Akteure in der virtuellen Limonaden-Abfüllfabrik nicht mit den vorkonfigurierten Variablen und Methoden des OPC-UA-Demo-Servers initialisiert werden müssen, ist ein eigener OPC-UA-Server in JavaScript zu implementieren. Somit ist auch eine zweckentfremdete Nutzung des OPC-UA-Servers, der für eine Gebäudeautomation konzipiert wurde, ausgeschlossen.

Wegen auftretenden Komplikationen bei der Integration des C++ basierten SDK in Microsoft Visual Studio 2019, fällt die Wahl der Programmiersprache für den Server kurzerhand auf JavaScript. Die einzige Notwendigkeit bei der Implementierung besteht dabei, die serverseitige Plattform Node.js auf der virtuellen Arbeitsumgebung zu installieren. Node.js 12.14.1 LTS ist kostenfrei auf der Website des Herstellers erhältlich. Es wird in der JavaScript-Laufzeitumgebung V8 ausgeführt und arbeitet somit asynchron und ereignisgesteuert. Node.js ist für die Erstellung skalierbarer Netzwerkanwendungen konzipiert.<sup>[10]</sup> Die Laufzeitumgebung, Node-Package-Manager und Module müssen bei der Installation von Node.js mitinstalliert werden. Unter Windows geschieht dies in Form einer ausführbaren Exe-Datei.

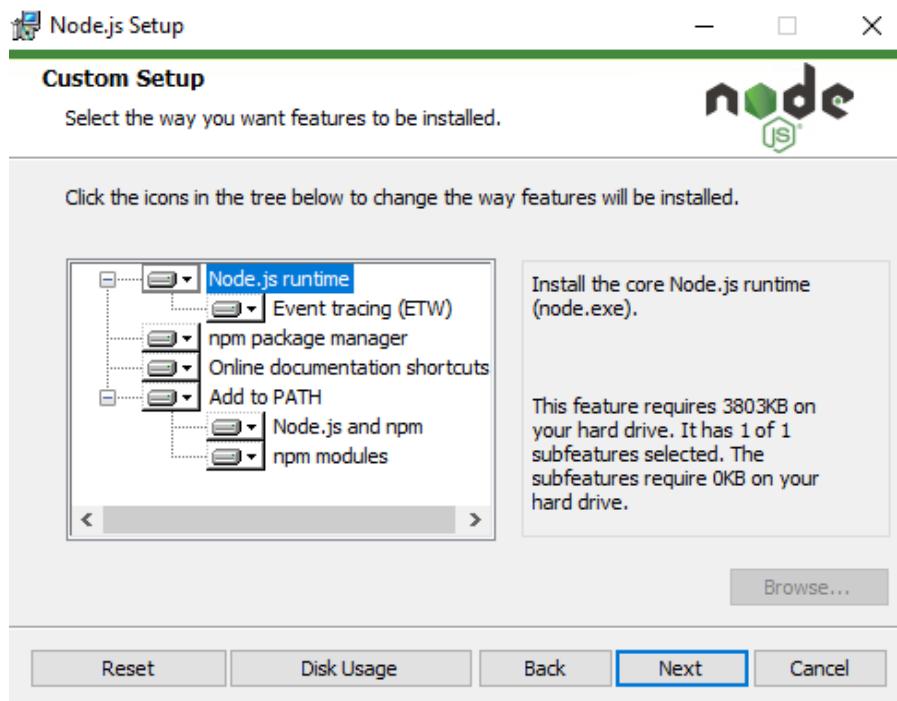


Abbildung 4: Installation von Node.js mit sämtlichen Paketen und Modulen

Nach erfolgreicher Installation beginnt die Konzeptionierung des OPC-UA-Servers. Der Server ist mit drei Objekten für die Sensoren in der Limonaden-Abfüllfabrik deklariert. Diesen Objekten unterliegen jeweils vier Lese-/ Schreibvariablen, die der Server für die Clients veröffentlicht.

Die ersten Schritte für die Serverimplementierung setzen voraus, dass eine Git Bash unter Windows ausgeführt ist. Git für Windows bietet eine BASH-Emulation, mit der Git von der Befehlszeile aus ausgeführt werden kann.<sup>[11]</sup> Die Version 2.25.0 ist kostenfrei auf der Website des Herstellers erhältlich.

Ist die Git Bash ausgeführt, so ist zunächst ein Verzeichnis für die Serverapplikation anzulegen, ein sogenanntes „node-project“. Dies geschieht mit dem Befehlszeilenkommando `mkdir opcserver`. Mit dem Kommando `cd opcserver` gelangen Sie in den neu angelegten Ordner. Das node-project benötigt eine package-Datei, geschrieben in JSON. In ihr befinden sich die zum Server gehörigen Metadaten. Die Datei wird mit dem Befehlszeilenkommando `npm init` generiert. Nach Eingabe des Kommandos öffnet sich ein interaktives Menü, in dem die Servereigenschaften einzutragen sind.

```
vagrant@vagrant-10 MINGW64 ~
$ mkdir opcserver

vagrant@vagrant-10 MINGW64 ~
$ cd opcserver

vagrant@vagrant-10 MINGW64 ~/opcserver
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (opcserver) opcserver
version: (1.0.0) 1.0.0
description: OPC UA Server
entry point: (index.js) index.js
test command: testopc
git repository: -
keywords: -
author: Michael Moelle
license: (ISC) ISC
```

Abbildung 5: Verzeichnis anlegen und npm init ausführen

Sind alle Eigenschaften eingetragen und die JSON-Datei generiert, ist der node mit `npm install node-opcua --save` zu installieren. Nach abgeschlossener Installation befinden sich die JSON-Dateien und die zugehörigen node-module im zuvor angelegten Verzeichnis.

 node_modules	2/5/2020 12:55 AM	File folder
 package.json	2/5/2020 12:55 AM	JSON File
 package-lock.json	2/5/2020 12:55 AM	JSON File

Abbildung 6: Package-Dateien und node-module

Nun ist mit einem beliebigen Texteditor eine Skriptdatei `opcserver.js` zu erstellen. Das Skript wird um die folgenden drei Schritte herum organisiert:

- declaration
- server instantiation
- server initialisation

Bei der Deklaration muss das node-opcua-SDK für die Serverapplikation verfügbar sein. Dies geschieht mit folgendem Statement:

```
const opcua = require("node-opcua");
```

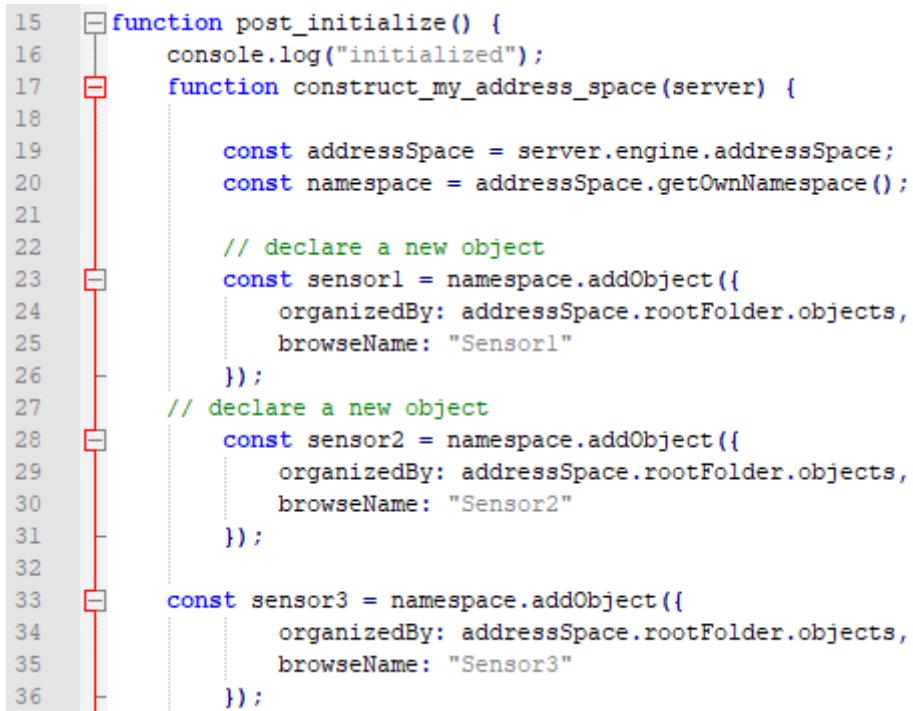
Es ist eine OPC-UA-Server-Instanz zu erstellen. Um das Serververhalten anzupassen, sind an diesen Optionen zu übergeben. Für einen simplen Server ist nur der TCP-Port anzugeben. Der Ressourcenpfad wird verwendet, um die einheitliche Endpunkt-Ressourcenkennung (uri) des Servers zu konstruieren. Da der Hostname der virtuellen Arbeitsumgebung `vagrant-10` ist, ergibt sich die uri namens `opc.tcp://vagrant-10:4334/UA/MyLittleServer`. Clients müssen diese Adresse nutzen, um Konnektivität zum Server zu erhalten.

```
1  /*global require, setInterval, console */
2  const opcua = require("node-opcua");
3
4  // create an instance of OPCUAServer
5  const server = new opcua.OPCUAServer({
6    port: 4334, // the port of the listening socket of the server
7    resourcePath: "/UA/MyLittleServer", // this path will be added to the endpoint resource name
8    buildInfo : {
9      productName: "MySampleServer1",
10     buildNumber: "7658",
11     buildDate: new Date(2014,5,2)
12   }
13 }) ;
```

Abbildung 4: Deklaration des Servers und Erstellung einer Instanz

Innerhalb der Ebene der Serverinstanziierung können noch zusätzliche Informationen, wie zum Beispiel `buildInfo` des Servers, ergänzt werden. Diese Informationen sind allerdings nicht zwingend für eine Konfiguration erforderlich.

Nach der Instanziierung folgt die Initialisierung des Servers. Während der Initialisierung lädt der Server sein Standard-node-set und bereitet die Bindung aller OPC-UA-Variablen vor. Die Initialisierungsmethode ist eine asynchrone Operation, die eine Callback-Funktion erfordert, die ausgeführt wird, wenn der Initialisierungsprozess abgeschlossen ist. Der Callback ist eine Funktion, die die Initialisierungsschritte enthält, die postwendend ausgeführt werden. Der AddressSpace wird verwendet, um das Objektmodell anzupassen, das der Server der externen Umgebung zugänglich macht. Bei den Objekten innerhalb dieses Adressspeichers handelt es sich um drei Sensoren mit den Bezeichnungen `sensor1`, `sensor2` und `sensor3`.



```
15  function post_initialize() {
16    console.log("initialized");
17    function construct_my_address_space(server) {
18
19      const addressSpace = server.engine.addressSpace;
20      const namespace = addressSpace.getOwnNamespace();
21
22      // declare a new object
23      const sensor1 = namespace.addObject({
24        organizedBy: addressSpace.rootFolder.objects,
25        browseName: "Sensor1"
26      });
27      // declare a new object
28      const sensor2 = namespace.addObject({
29        organizedBy: addressSpace.rootFolder.objects,
30        browseName: "Sensor2"
31      });
32
33      const sensor3 = namespace.addObject({
34        organizedBy: addressSpace.rootFolder.objects,
35        browseName: "Sensor3"
36      });
37
```

Abbildung 5: Erstellung des AddressSpace samt Objekten

Um den angelegten Sensorobjekten entsprechende Werte aus der Simulation der Limonaden-Abfüllfabrik übergeben zu können, sind für jeden Sensor Lese-/ Schreibvariablen zu definieren. Jeder Sensor besitzt vier Variablen vom Datentyp Int32, die mit dem Schlüsselwort `let` deklariert werden. Die Vergabe von nodeIDs für die Variablen ist nicht zwingend erforderlich, da der Server eine automatische Zuweisung durchführt. `sensor1` erhält die Variablen `variable1_s1`, `variable2_s1`, `variable3_s1` und `variable4_s1`. Die Namen der Variablen der anderen Sensoren unterscheiden sich lediglich in deren Endungen. Für `sensor2` die Endung `_s2` und `sensor3` die Endung `_s3`. Die Eigenschaft `componentOf` ist jeweils mit dem zugehörigen Sensorobjekt initialisiert. Für den Wert selbst, also `value`, sind entsprechende get- und set-Methoden implementiert. Die Funktion `parseFloat`, mit der die Variablen in der set-Methode initialisiert sind, ist für den Fall, dass aus gegebenen Umständen der Datentyp der Variable von Int32 auf Double geändert werden kann. In der get-Methode ist der zu lesende Wert und dessen Datentyp angegeben.

```
38 //Variablen für das Objekt sensor1 definieren und dem Namespace hinzufügen
39 let variable1_s1 = 0;
40 namespace.addVariable({
41
42     componentOf: sensor1,
43     browseName: "Variable_1",
44     dataType: "Int32",
45     value: {
46         get: function () {
47             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable1_s1 });
48         },
49         set: function (variant) {
50             variable1_s1 = parseFloat(variant.value);
51             return opcua.StatusCodes.Good;
52         }
53     }
54 });
55 // zweite Variable fuer Sensor1 - read and write
56 let variable2_s1 = 0;
57 namespace.addVariable({
58
59     componentOf: sensor1,
60     browseName: "Variable_2",
61     dataType: "Int32",
62     value: {
63         get: function () {
64             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable2_s1 });
65         },
66         set: function (variant) {
67             variable2_s1 = parseFloat(variant.value);
68             return opcua.StatusCodes.Good;
69         }
70     }
71 });
72 // dritte Variable fuer Sensor1 - read and write
73 let variable3_s1 = 0;
74 namespace.addVariable({
75
76     componentOf: sensor1,
77     browseName: "Variable_3",
78     dataType: "Int32",
79     value: {
80         get: function () {
81             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable3_s1 });
82         },
83         set: function (variant) {
84             variable3_s1 = parseFloat(variant.value);
85             return opcua.StatusCodes.Good;
86         }
87     }
88 });
89 // vierte Variable fuer Sensor1
90 let variable4_s1 = 0;
91 namespace.addVariable({
92
93     componentOf: sensor1,
94     browseName: "Variable_4",
95     dataType: "Int32",
96     value: {
97         get: function () {
98             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable4_s1 });
99         },
100         set: function (variant) {
101             variable4_s1 = parseFloat(variant.value);
102             return opcua.StatusCodes.Good;
103         }
104     }
105 });


```

Abbildung 6: Definieren der Variablen für das Objekt sensor1

```
108 //Variablen für das Objekt sensor2 definieren und dem Namespace hinzufügen
109 let variable1_s2 = 0;
110 namespace.addVariable({
111
112     componentOf: sensor2,
113     browseName: "Variable_1",
114     dataType: "Int32",
115     value: {
116         get: function () {
117             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable1_s2 });
118         },
119         set: function (variant) {
120             variable1_s2 = parseFloat(variant.value);
121             return opcua.StatusCodes.Good;
122         }
123     }
124 });
125 //Zweite Variable für sensor2 - read and write
126 let variable2_s2 = 0;
127 namespace.addVariable({
128
129     componentOf: sensor2,
130     browseName: "Variable_2",
131     dataType: "Int32",
132     value: {
133         get: function () {
134             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable2_s2 });
135         },
136         set: function (variant) {
137             variable2_s2 = parseFloat(variant.value);
138             return opcua.StatusCodes.Good;
139         }
140     }
141 });
142 //Dritte Variable für sensor2 - read and write
143 let variable3_s2 = 0;
144 namespace.addVariable({
145
146     componentOf: sensor2,
147     browseName: "Variable_3",
148     dataType: "Int32",
149     value: {
150         get: function () {
151             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable3_s2 });
152         },
153         set: function (variant) {
154             variable3_s2 = parseFloat(variant.value);
155             return opcua.StatusCodes.Good;
156         }
157     }
158 });
159 //Vierte Variable für sensor2 - read and write
160 let variable4_s2 = 0;
161 namespace.addVariable({
162
163     componentOf: sensor2,
164     browseName: "Variable_4",
165     dataType: "Int32",
166     value: {
167         get: function () {
168             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable4_s2 });
169         },
170         set: function (variant) {
171             variable4_s2 = parseFloat(variant.value);
172             return opcua.StatusCodes.Good;
173         }
174     }
175 });
```

Abbildung 7: Definieren der Variablen für das Objekt sensor2

```

177 //Variablen für das Objekt sensor3 definieren und dem Namespace hinzufügen
178 let variable1_s3 = 0;
179 namespace.addVariable({
180
181     componentOf: sensor3,
182     browseName: "Variable_1",
183     dataType: "Int32",
184     value: {
185         get: function () {
186             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable1_s3 });
187         },
188         set: function (variant) {
189             variable1_s3 = parseFloat(variant.value);
190             return opcua.StatusCodes.Good;
191         }
192     }
193 });
194
195 //Zweite Variable für sensor2 - read and write
196 let variable2_s3 = 0;
197 namespace.addVariable({
198
199     componentOf: sensor3,
200     browseName: "Variable_2",
201     dataType: "Int32",
202     value: {
203         get: function () {
204             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable2_s3 });
205         },
206         set: function (variant) {
207             variable2_s3 = parseFloat(variant.value);
208             return opcua.StatusCodes.Good;
209         }
210     }
211 });
212 //Dritte Variable für sensor2 - read and write
213 let variable3_s3 = 0;
214 namespace.addVariable({
215
216     componentOf: sensor3,
217     browseName: "Variable_3",
218     dataType: "Int32",
219     value: {
220         get: function () {
221             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable3_s3 });
222         },
223         set: function (variant) {
224             variable3_s3 = parseFloat(variant.value);
225             return opcua.StatusCodes.Good;
226         }
227     }
228 });
229 //Vierte Variable für sensor2 - read and write
230 let variable4_s3 = 0;
231 namespace.addVariable({
232
233     componentOf: sensor3,
234     browseName: "Variable_4",
235     dataType: "Int32",
236     value: {
237         get: function () {
238             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable4_s3 });
239         },
240         set: function (variant) {
241             variable4_s3 = parseFloat(variant.value);
242             return opcua.StatusCodes.Good;

```

Abbildung 8: Definieren der Variablen für das Objekt sensor3

Der Aufruf der Funktion `construct_my_address_space(server)` baut den Adressspeicher auf. Nachdem der Server erstellt und initialisiert wurde, ist die asynchrone Startmethode zu verwenden, damit der OPC-UA-Server all seine Endpunkte initiieren und mit dem Abhören der Clients beginnen kann. Das Abhören soll mittels der Tastenkombination **CTRL+C** deaktiviert werden können.

```

248     construct_my_address_space(server);
249     server.start(function() {
250         console.log("Server is now listening ... ( press CTRL+C to stop )");
251         console.log("port ", server.endpoints[0].port);
252         const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
253         console.log(" the primary server endpoint url is ", endpointUrl );
254     });
255 }
256 server.initialize(post_initialize);

```

Abbildung 9: Asynchrone Startmethode des Servers

Am Ende des Skriptes befindet sich der Aufruf der postwenden Initialisierungsmethode `server.initialize(post_initialize);`, die den Initialisierungsprozess in Gang setzt. Der Quellcode ist unter dem Dateinamen `sample_server.js` gespeichert.

#### 4.2.2 Verbindungsaubau und die daraus resultierende Problematik

Nachdem der Quellcode, geschrieben in JavaScript, zusammen mit den nötigen node-modules und den Package-Dateien, geschrieben in JSON, zusammen im angelegten Verzeichnis auf der virtuellen Arbeitsumgebung gespeichert sind, kann mittels dem Konsolenkommando `cd opcserver` in jenes Verzeichnis gewechselt werden. Mit dem Befehl `node sample_server.js` wird Serverkonsolenanwendung gestartet.

```
vagrant@vagrant-10 MINGW64 ~/opcserver
$ node sample_server.js
initialized
Server is now listening ... ( press CTRL+C to stop)
port 4334
the primary server endpoint url is  opc.tcp://vagrant-10.fritz.box:4334/UA/MyLittleServer
```

Abbildung 7: Ausführen der Serverkonsolenanwendung

Nachdem die Serverapplikation erfolgreich in der Konsole gestartet wurde, kann die Konnektivität zum Hostsystem beispielsweise mit dem Testclient UaExpert verifiziert werden. Hierzu muss dieser, installiert auf dem Hostsystem, ausgeführt und mit der entsprechenden Endpoint-URL des OPC-UA-Servers konfiguriert werden. Mit einem Mausklick auf das *Plus-Symbol* in der oberen Leiste des Clients kann ein Server der Konfiguration hinzugefügt werden. Nach genanntem Mausklick öffnet sich ein Fenster, in welchem durch Doppelklick auf den Listenpunkt *Custom Discovery*, die Server-URL einzutragen ist.

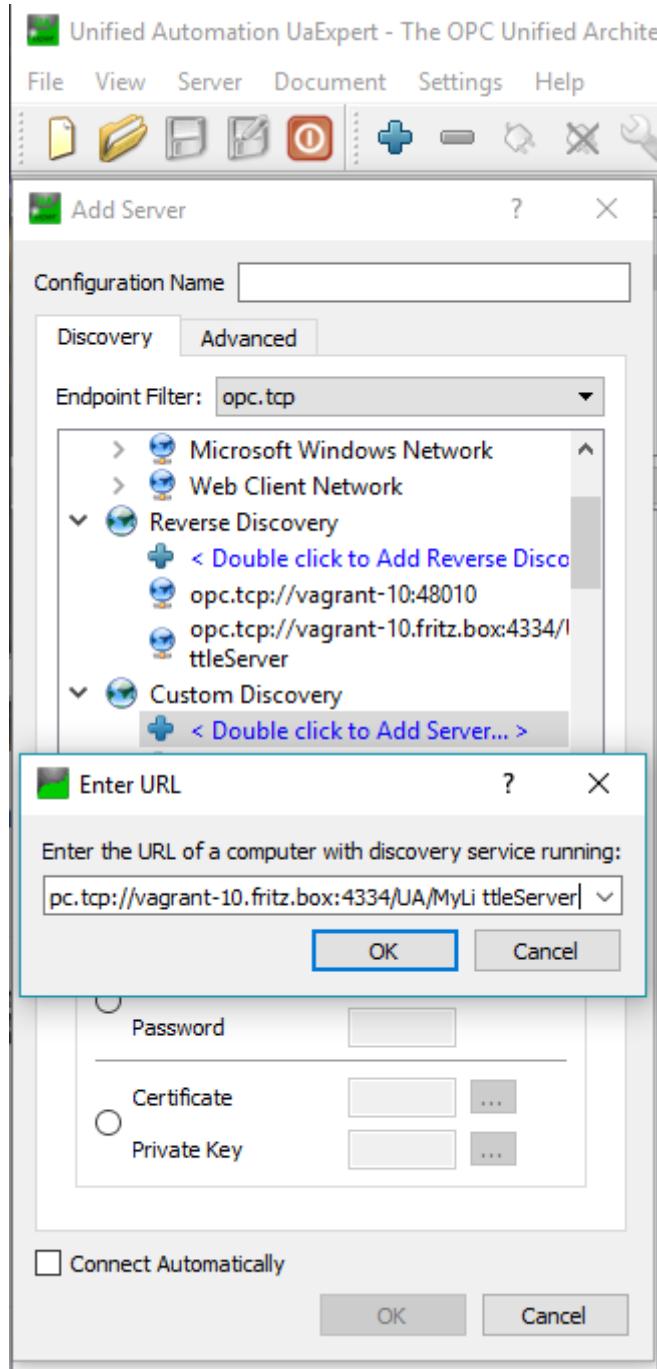


Abbildung 8: Server in der Konfiguration von UAExpert aufnehmen

Aufgenommen in der Serverliste, öffnet sich durch einen Mausklick auf die hinzugefügte Server-URL ein Drop-Down-Menü, in dem durch Doppelklick auf *None* eine unverschlüsselte Verbindung zum OPC-UA-Server aufgebaut wird.

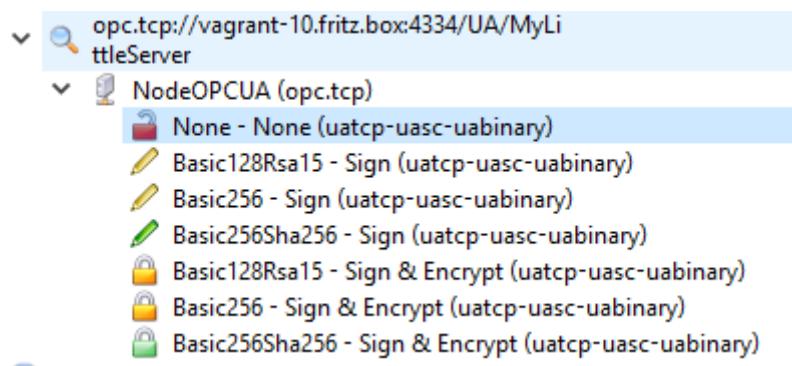


Abbildung 9: Auswählen des Security-Standards None

Durch die anschließende Betätigung des *Elektrostecker-Symbols* wird eine Connection nach vorher konfigurierten Einstellungen aufgebaut. Durch die Verwendung einer unverschlüsselten Verbindung, öffnet sich ein Fenster, in welchem vor einem nicht vertrauenswürdigen Zertifikat gewarnt wird. Die Zertifikate sind temporär zuzulassen und mit Betätigung der Schaltfläche *Continue* die Verbindung zu aktivieren.

Im linken Bereich der Client-Software befindet sich der AddressSpace in Form eines hierarchischen Dateisystems. Sind bei der Implementierung keine Syntaxfehler unterlaufen und konnte die Serverkonsole anwendung erfolgreich gestartet werden, so befinden sich im Adressenspeicher drei Objekte namens *Sensor1*, *Sensor2* und *Sensor3* mit jeweils vier Lese-/ Schreibvariablen.

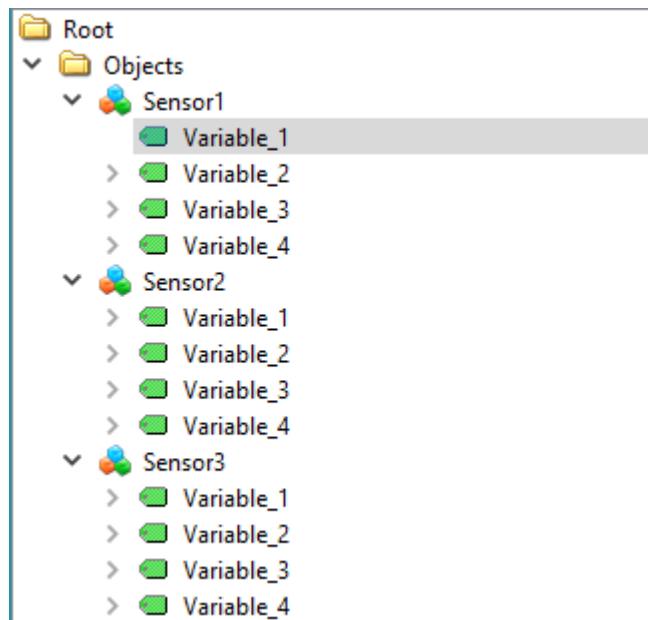


Abbildung 10: AdressSpace des implementierten OPC-UA-Servers

Um die Verbindung zur Software *Visual Components*, welche ebenfalls auf dem Hostsystem installiert ist, zu überprüfen, werden in besagter Anwendung Konfigurationen

unter dem Reiter *Konnektivität* unternommen. Auf diese Einstellungen sind in der weiteren Projektumsetzung dokumentiert.

Beim Verbindungsauflauf mit der Anwendung OPC Router führt dies zu einer Problematik, die im späteren Projektverlauf von elementarer Relevanz ist. Aufgrund nicht implementierter Sicherheitsstandards und Zertifikaten ist es zwar möglich den OPC-UA-Server mit dem OPC Router, allerdings nicht mit der relationalen Datenbank zu verbinden. Demzufolge wäre es nötig in der Datenbank einen Benutzer neben Root anzulegen, dementsprechende Zertifikate zu generieren und Authentifizierungsmöglichkeiten zu implementieren. Fehlende Kenntnisse in diesen Bereichen aber auch nicht mehr vorhandene Zeitressourcen veranlassen dazu, wie damals auch vorgesehen, den OPC-UA-Demo-Server, geschrieben in C++, zu verwenden.

#### 4.2.3 Konfiguration und Inbetriebnahme des OPC-UA-Demo-Servers

Nach erfolgreichem Downloaden und Installieren des OPC-UA-Demo-Servers (<https://www.unified-automation.com/de/downloads/opc-ua-servers.html>) auf der virtuellen Arbeitsumgebung, wird dieser in Form einer Konsolenanwendung ausgeführt. Die Datei namens *uaservercpp* befindet sich im jeweiligen Installationsverzeichnis im Ordner *bin*.

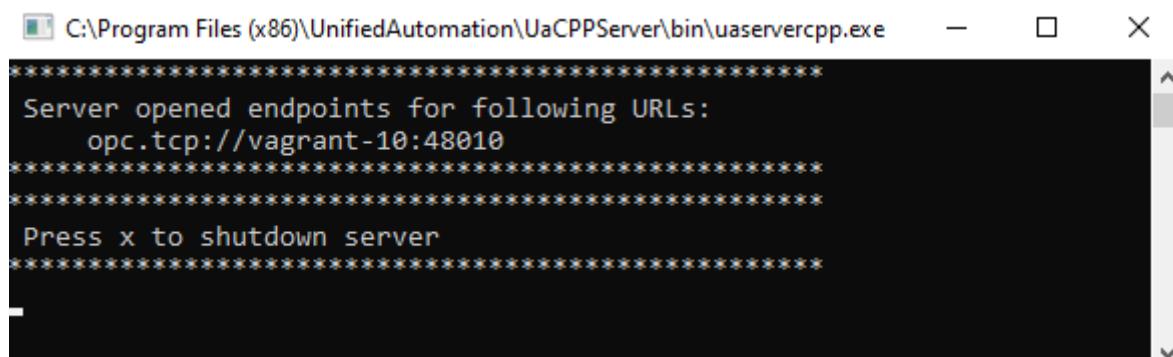
A screenshot of a terminal window titled 'C:\Program Files (x86)\UnifiedAutomation\UaCPPServer\bin\uaservercpp.exe'. The window displays the output of the server's startup process. It shows the server has opened endpoints for URLs: 'opc.tcp://vagrant-10:48010'. A message at the bottom of the window says 'Press x to shutdown server'.

Abbildung 11: OPC-UA-Demo-Server C++ in Betrieb

Die URL `opc.tcp://vagrant-10:48010` ist die Adresse, über welche sich Visual Components, UaExpert, der OPC Router, die relationale Datenbank und node-RED mit dem Server verbinden. Diese muss in der Konfiguration einer jeden Komponente eingetragen sein. Mit dem Testclient UaExpert ist die Hierarchie des Servers samt seinen Ordnern und Variablen einzusehen und für die Verbindung „Server zur Simulation“ Werte einzutragen.

#### 4.2.4 Serverhierarchie

Wie anfangs beschrieben, ist der von der Unified Automation zur Verfügung gestellte OPC-UA-Demo-Server für das Konzept einer Gebäudeautomatisierung implementiert worden. Daher befindet sich in der Verzeichnisstruktur, unter dem Ordner Objekte, das Verzeichnis *BuildingAutomation*. Variablen und Methoden in diesem Bereich können allerdings auch für andere Zwecke verwendet werden. Der Sauberkeit halber befinden sich die konfigurierten Variablen für die Limonaden-Abfüllfabrik-Simulation unter dem Ordner *PubSubTargetVariables*. Dieser ist untergliedert in drei *Publisher*-Sets mit jeweils drei *DataSets*. In diesen befinden sich frei konfigurierbare Variablen unterschiedlicher Datentypen.

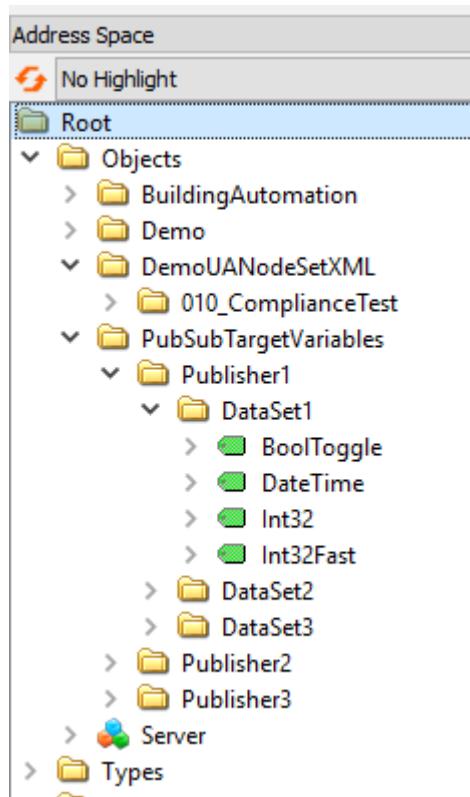


Abbildung 12: Serverhierarchie

#### 4.2.5 Verwendete Pub-/ Sub-Ziel-Variablen

Um eine klare Verzeichnisstruktur beizubehalten und Variablen nicht zu zweckentfremden, sind Variablen aus dem Ordner *PubSubTargetVariables* zu nutzen. Jede Variable des Servers kann über eine eindeutige Adresse aus dem sogenannten *AddressSpace* identifiziert und angesprochen werden. Diese Adresse bezeichnet man als *Node-ID*. Für die zu erfassenden Sensordaten kommen die Datentypen Int16, Int32, Int64 und String

in Frage. OPC-UA arbeitet bidirektional. Aufgrund dieser Tatsache sind auch Variablen vom Typ Boolean für gewisse Funktionalitäten von Aktoren zu verwenden. Eine Tabelle zeigt eine Auflistung aller zu benutzenden Variablen samt Datentypen, Verwendungszweck und Node-ID.

<b>Node-ID</b>	<b>Datentyp</b>	<b>Verwendungszweck</b>
ns=2;s=PSTV.P1.DS1.Int32	Int32	FlaschenID_Palette: Zählt von 0 bis 800 und setzt den Wert wieder auf 0
ns=2;s=PSTV.P1.DS1.Int32Fast	Int32Fast	PalettenID: Zählt von 0 aufsteigend. Eine Palette enthält 800 Flaschen
ns=2;s=PSTV.P1.DS2.String	String	Abfuell-Datum: Tag und Uhrzeit
ns=2;s=PSTV.P1.DS2.Int32	Int32	Flaschen-ID: Zählt von 0 aufsteigend. Jede Flasche erhält eine eindeutige ID
ns=2;s=PSTV.P1.DS2.Int64	Int64	FlaschenID: Zählt von 0 aufsteigend. IDs für defekte Flaschen
ns=2;s=PSTV.P1.DS2.Int16	Int16	BottleBatchCapper: Regelt, wie viele Flaschen gleichzeitig verschraubt werden
ns=2;s=PSTV.P1.DS1.BoolToggle	BoolToggle	ReferenceFeeder: Eingehendes Signal zur Erzeugung einer Flasche

Tabelle 1: Verwendete Variablen

Diese Variablen sind im späteren Verlauf in Visual Components mit Variablen der Software zu verbinden.

## 4.3 Inbetriebnahme der Limonaden-Abfüllfabrik in Visual Components

Dieses Kapitel der Projektdurchführung beschreibt die Inbetriebnahme der virtuellen Limonaden-Abfüllfabrik in der Software Visual Components. Die anfangs beschriebene Anwendung ist auf dem Hostsystem installiert und enthält einige bereits implementierte Layouts. Die Abfüllfabrik, genannt „BottleFillingLayout“ ist eines davon. Im Bereich des linken Bildschirmrandes befindet sich ein Menü namens *Sammlungen*, über welches durch Eingabe des genannten Layouts, dieses aufgerufen werden kann. Ausgewählt, erscheint das Layout im 3D-Editierbereich. Die Limonaden-Abfüllfabrik ist voll funktionsfähig und sofort einsatzbereit. In ihr befinden sich etliche Sensoren und Aktoren, die es erlauben die Anlage zu bedienen und auszuwerten. Im oberen mittleren Bildschirmbereich befindet sich ein Abspielmenü. Durch Betätigung des Play-Buttons wird die Simulation in Betrieb gesetzt.

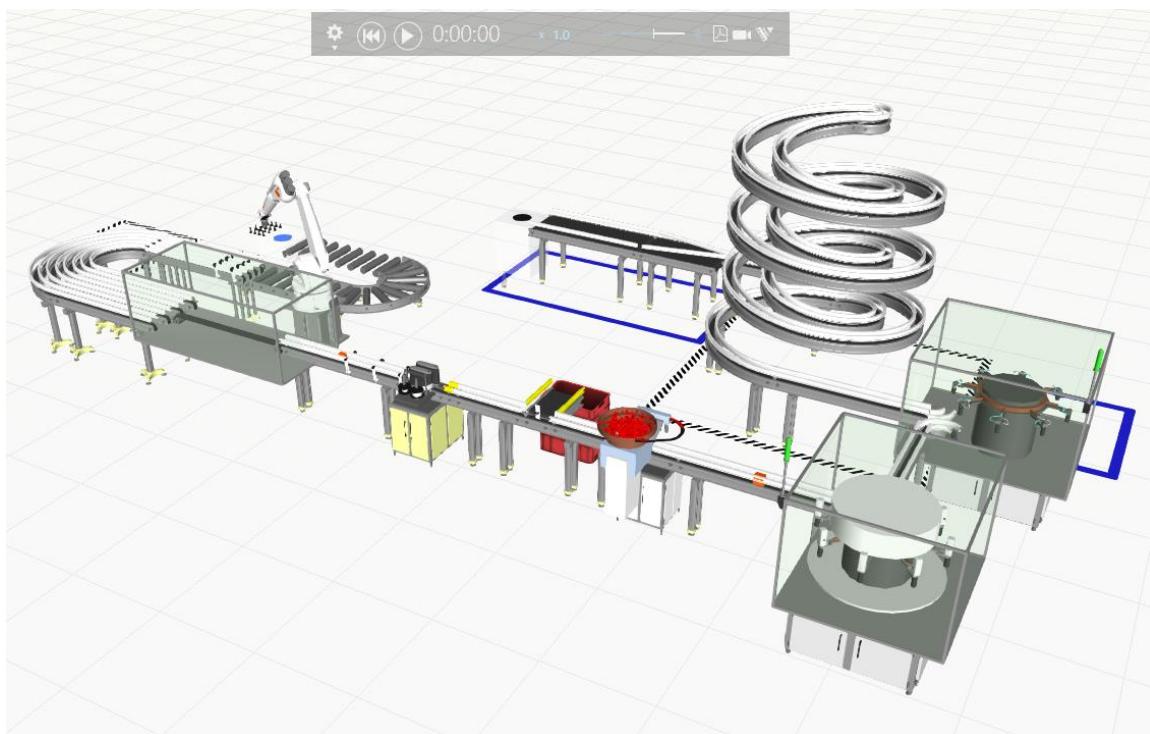


Abbildung 13: Gesamtansicht der Limonaden-Abfüllfabrik

### 4.3.1 Konzeption eines Addons zum Auswerten von Benachrichtigungen

Um ein Verständnis für die Funktionsweise der Sensoren und von Produktionsvorgängen zu erhalten, ist es von elementarer Bedeutung ein Addon zu konzipieren, welches zur Auswertung von Sensornachrichten benutzt wird. Nachrichten erscheinen am unte-

ren Ausgabefenster der Software. Das Addon ermöglicht die Speicherung dieser Nachrichten in externe Dateien. Zudem ist es möglich den Inhalt des Ausgabafensters zu löschen. Die Python-API von Visual Components erlaubt es, Programmerweiterungen selbst zu implementieren. Unter dem Dateipfad \VisualComponents\4.1\MyCommands sind die Dateien messageHelper.py und \_\_init\_\_.py anzulegen. Die erste enthält den Quellcode des Addons und die letztere dient zu dessen Initialisierung.

```
1  from vcCommand import *
2
3  cmd = getCommand()
4  app = getApplication()
5
6  def clearMessages(prop):
7      app.clearMessages()
8
9  def saveMessages(prop):
10     msg = app.getMessages()
11     uri = r"C:\Users\Michael\Documents\Visual Components\4.1\output.txt"
12     with open(uri, "a") as f:
13         f.write(msg)
14
15 def first_state():
16     executeInActionPanel()
17
18 addState(first_state)
19
20 btnClear = cmd.createProperty(VC_BUTTON, "Clear Messages")
21 btnClear.OnChanged = clearMessages
22
23 btnSave = cmd.createProperty(VC_BUTTON, "Save Messages")
24 btnSave.OnChanged = saveMessages
```

Abbildung 10: Quellcode des Addons messageHelper

In Zeile eins werden sämtliche Python-Methoden (\*) aus der Bibliothek vcCommand in das Skript importiert. Darunter ist die Variable cmd deklariert und mit der Methode getCommand() initialisiert. Sie gibt einen Befehl mit einem angegebenen Namen zurück. In der darunterliegenden Zeile wird die Variable app mit der Methode getApplication() initialisiert. In der Definition des Events clearMessages ist festgelegt, dass nach Drücken des Buttons *Clear* die Outputnachrichten im Ausgabefeld gelöscht werden. In der Definition des darunterliegenden Events saveMessages ist die Variable msg deklariert. Diese wird mit der Methode getMessage() initialisiert, welche mit der Variable app aufgerufen wird. Die Variable uri darunter ist mit dem absoluten Pfad, in dem später die Ausgabenachrichten abgespeichert werden und mit der Aufforderung den Inhalt der Variable msg dort zu speichern, initialisiert. Damit nach Drücken auf einen Button erkannt werden kann, dass es sich dabei um eine Statusänderung handelt, ist es nötig ein Event first\_state zu definieren (Zeile 15).

In Zeile 20 wird noch eine Variable `btnClear` zum Löschen des Ausgabetextes angelegt. Nachdem der Button geklickt wurde, also eine Zustandsänderung erfolgt ist (`OnChanged`), wird das ausgeführt, was unter `clearMessages` definiert ist. In Zeile 23 wird dieses noch einmal mit der Variable `btnSave` zum Speichern der Ausgabenachrichten festlegt. Wird hier eine Zustandsänderung festgestellt wird das ausgeführt, was in `saveMessages` definiert ist.

`__init__.py` ist eine Initialisierungsdatei. Wenn sich eine init-Datei in einem Ordner befindet, wird dieser von Python als Package angesehen und kann importiert werden. Folgender Code befindet sich in der `__init__.py`-Datei:

```
1  from vcApplication import *
2
3  def OnStart():
4      cmduri = getApplicationPath() + "messageHelper.py"
5      cmd = loadCommand("messageHelper", cmduri)
6      addMenuItem("VcTabHome/Test", "messageHelper", -1, "messageHelper")
7
```

Abbildung 11: Pythonskript der Init-Datei

In Zeile eins werden sämtliche Methoden (\*) der `vcApplication` in das Skript importiert. Im definierten `OnStart`-Event wird die Variable `cmduri` mit der Methode `getApplicationPath` (gibt den Installationspfad der Software zurück) und der Datei `messageHelper.py` initialisiert. In Zeile sechs wird noch mit der Methode `addMenuItem()` der Reiter in der grafischen Benutzeroberfläche implementiert. Mit Mausklick auf diesen wird die Programmerweiterung in Visual Components gestartet.

#### 4.3.2 Konzeption eines Volumensensors zum Erfassen der Flaschenstückzahl

Dieses Kapitel beschreibt, wie zu Testzwecken ein Sensor an ein Fließband fixiert und programmiert wird. Aufgabe dieses Sensors ist es die Anzahl der überquerenden Flaschen zu erfassen. Der Volumensensor befindet sich, in Betracht auf die Reihenfolge der Produktionsschritte, hinter der Flaschenlabelstation und hinter einem weiteren Sensor, der fehlerhaft verschlossene Flaschen erkennt und aussortieren lässt. Somit ist gewährleistet, dass der Sensor nur korrekt abgefüllt und verschlossene Flaschen erfasst. Das Fließband trägt den Komponentennamen `Conveyor #9`. Nachdem die Komponente mit dem Cursor markiert ist, wird ihr über den Reiter `Modellierung` und dessen Unterpunkt `Verhalten` ein Volumensensor hinzugefügt. Eine Flasche ist immer dann vom Sensor zu erfassen, wenn das Objekt einen bestimmten Erfassungsbereich, einen sogenannten „Frame“, überquert. Zur Erschaffung eines solchen Frames ist es nötig den unteren Bereich (Lower Frame) und den oberen Bereich (Upper Frame) festzulegen. Dies geschieht, indem dem zuvor installiertem Volumensensor über den Unter-

punkt Features zwei Frames hinzugefügt werden. Nachdem die oberen und unteren Eckpunkte des Erfassungsbereichs gekennzeichnet sind, werden diese über das Eigenschaftenpanel des Volumensensors auf der rechten Seite den entsprechenden Punkten *LowerFrame* und *UpperFrame* zugewiesen. Nun wird eine ausgefüllte Fläche des Frames in der Simulation angezeigt.

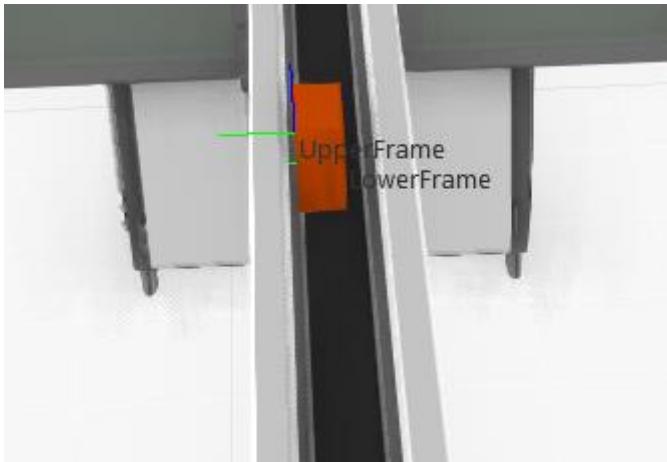


Abbildung 14: Erfassungsbereich des Sensors

Damit der Sensor die gewünschte Funktionalität erhält, ist es nötig ihm über den Reiter *Verhalten* ein Skript in der Sprach *Python* und ein *Boolean Signal* hinzuzufügen. Das Signal ist mit dem Skript gekoppelt und ermöglicht eine Aufnahme der Anzahl überquerender Komponenten. Um feststellen zu können, um welche genaue Art Komponente es sich bei den Objekten handelt, muss dem Sensor ein *ComponentSignal* über denselben Reiter hinzugefügt und ebenfalls mit dem Skript verbunden werden. Nachdem Python-skript, ComponentSignal und BooleanSignal hinzugefügt sind, ist über den in Visual Components integrierten Texteditor das Skript zu implementieren.

```
1  from vcScript import *
2
3  comp = getComponent()
4  sensor = comp.findBehaviour("VolumeSensor")
5  #Deklaration und Initialisierung einer Zählvariable
6  bottle_counter = 0
7
8  def OnSignal( signal ):
9      #print signal.Value
10     global bottle_counter
11     bottle_counter += 1
12     print "Filled bottle no. ", bottle_counter
13
14 def OnRun():
15     pass
16
17
18
```

Abbildung 12: Quellcode des Volumensensors

In der ersten Zeile des Skripts werden sämtliche Python-Methoden (\*) aus der Bibliothek `vcScript`, welche von Visual Components zur Verfügung gestellt werden, importiert. In Zeile drei ist eine Variable namens `comp` deklariert und zugleich mit der Methode `getComponent()` initialisiert. Die `getComponent()`-Methode ermöglicht es, eine Objektreferenz zu dem Objekt zu erhalten, welches die Verhaltensweise, nämlich das Python Skript, enthält. Direkt unter dieser Zeile ist die Variable `sensor` mit der Methode `findBehaviour()` initialisiert. Das Komponentenobjekt `comp` wird hier verwendet, um die Methode aufzurufen, um ein Verhalten nach Namen `VolumeSensor` zu erhalten. In Zeile sechs findet die Deklaration der Variable `bottle_counter` statt, welche die Anzahl der passierenden Flaschen aufnimmt. Diese Variable ist zunächst mit dem Wert `null` initialisiert.

Das `BooleanSignal` kommt im `OnSigal`-Event zum Einsatz. Jedes Mal, wenn ein Objekt, in diesem Falle eine Flasche, den Erfassungsbereich durchquert, wechselt das Signal von `null` auf `eins`. In Zeile acht ist das Event definiert. Dass sich der Wert der Variable `bottle_counter` bei jedem wechselnden Signal um eins erhöhen soll, ist in Zeile elf festgelegt. Darunter wird mit der Funktion `print` der Text „`Filled bottle no.`“ und der entsprechende Wert von `bottle_counter` auf der Ausgabe der Software Visual Components ausgegeben.

### 4.3.3 Sensoren und ihre zu erfassenden Daten

Einer der ersten Schritte in der Anfangsphase, war das Erstellen und die Konfiguration eines Volumensensors für das Zählen der produzierten Flaschen. Trotz Tutorials gestalte sich die Einrichtung schwer. Zusätzlich musste noch ein Python Skript, der das Verhalten des Sensors definiert, neu geschrieben werden. Dies führte dazu, dass man nach der ersten Kennenlernphase des Programms, die programminternen Fließbandsensoren entdeckt und eingesetzt hat. Jene haben sich später bei einem Kontrollzählendurchgang als genau zählend erwiesen, als der Volumensensor.

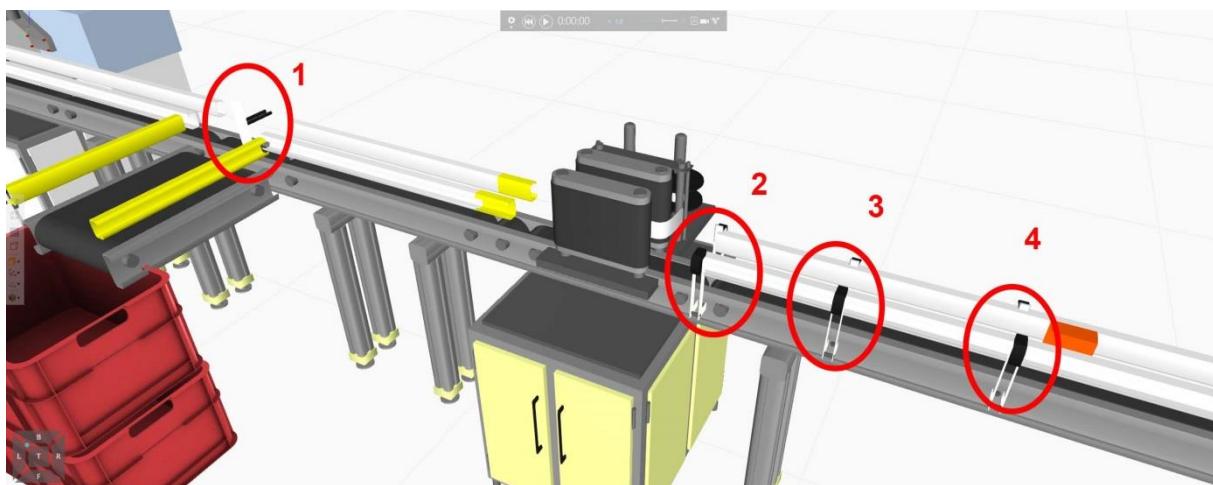


Abbildung 13: Hinzugefügte und konfigurierte Sensoren

#### Sensor 1

In Visual Components gibt es die Möglichkeit einen prozentualen Ausschuss einzustellen. In unserem Fall werden Flaschen vom Programm selbst sporadisch als „defekt“ markiert. Wie die Abbildung 16 zeigt, ist eine falsche Füllmenge, oder eine nicht verschlossene Flasche kein Kriterium für eine „defektes“ Produkt. Die defekten Flaschen werden im Regelfall mit dem sog. Pusher heraussortiert. Dieser, bereits von Visual Components in der Fabrik installierter Sensor war aber ohne Funktion bzw. zeigte keine Reaktion auf die defekten Flaschen. Die Intension, einen völlig neuen Pusher mit Hilfe von Tutorials zusammenzusetzen und zu konfigurieren schlug fehl. Auch das Durchsuchen weiterer Fabriktemplates in Visual Components nach einem Ersatzpusher blieb erfolglos. Daher wurde ein Pusher aus einer Visual-Components-Online-Bibliothek als einzelne Komponente heruntergeladen und anstelle des ursprünglichen Pushers in die Flaschenfabrik eingefügt. Danach wurden die Eigenschaften wie die Ausfahrdistanz, für das Erreichen der Flasche des Förderbands zum Aussortieren und die Ausfahrgeschwindigkeit des Pushers angepasst. Da die Ausfahrgeschwindigkeit des Pushers jedoch bei der Standardproduktionsgeschwindigkeit der Flaschen schwer mit dem

menschlichen Auge wahrnehmbar ist, wurde die ganze Produktionsgeschwindigkeit heruntergedrosselt. Die Produktionsgeschwindigkeit ist in dem „ReferenceFeeder“ einzustellen.

Ein Python-Script, in dem das Verhalten des Pushers von Visual Components vordefiniert ist, ist mit dem Pusher verknüpft. In dieses Script wurde zusätzlich folgende Zeile eingefügt:

```
aussortiertProp.Value += 1
```

Bei einer, als „defekt“ markierten Flasche, bekommt der Sensor ein Signal, der Pusher drückt die Flasche auf ein anderes Förderband, der zu einer Kiste für den Ausschuss führt und die Anzahl der aussortierten Flaschen wird um Faktor 1 erhöht.

Dieser kann mit einem manuellen Reset-Knopf wieder auf 0 gesetzt werden. Für den Reset-Knopf wurde ein vorhandener Python-Code anderer Knöpfe kopiert und angepasst.

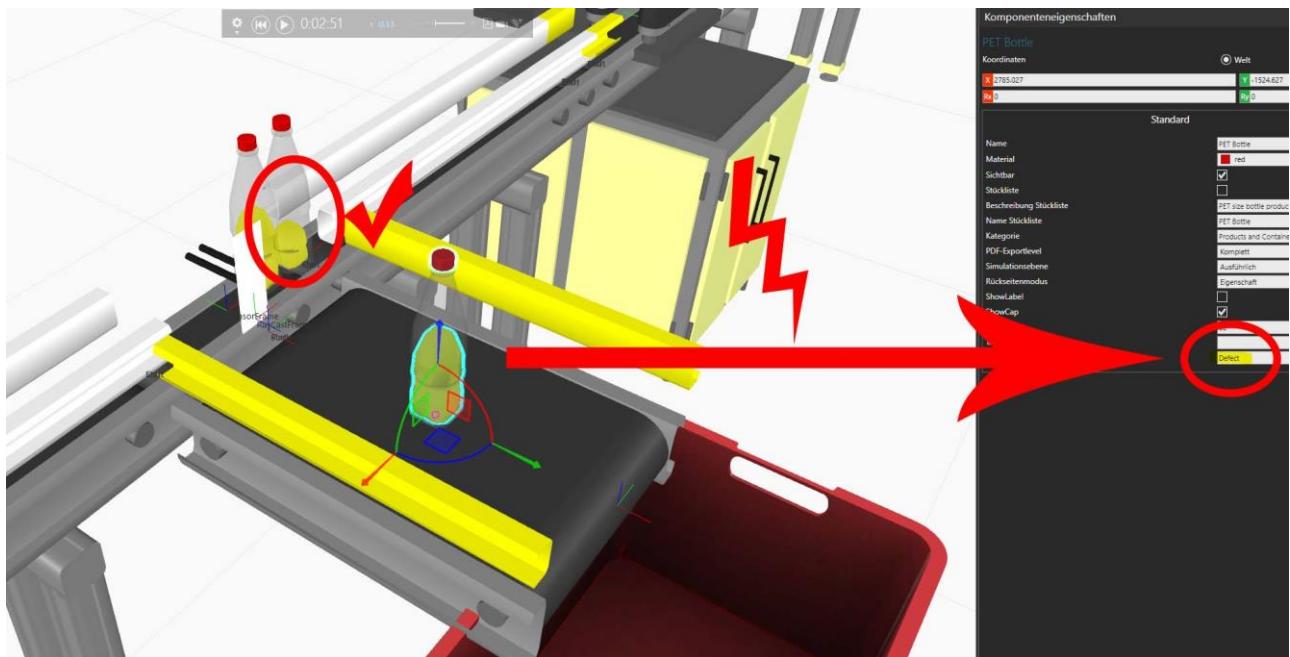


Abbildung 14: Inkorrekt er Aussortievorgang. Defekte Flaschen werden nicht aussortiert

### Sensor 2:

Der zweite Sensor schreibt bei einem Signaleingang das aktuelle Datum mit der Uhrzeit in eine Char-Variable, so dass beide Werte in den Eigenschaften der Flasche zu sehen ist.

```
propDat = part.createProperty(VC_STRING, 'Target')
prop.Value = stampingProdID.Value
propDat.Value = Datum.Value
format_str = "%A, %d %b %Y %H:%M:%S %p"
datumProp.Value = datetime.now().strftime(format_str)
```

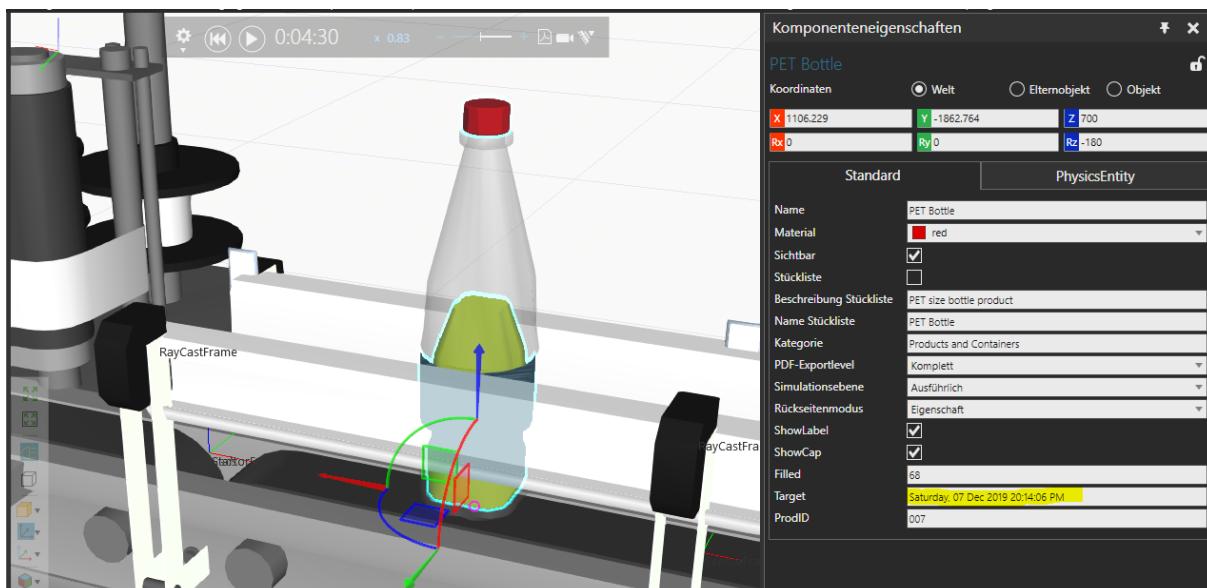


Abbildung 15: Eigenschaften einer Flasche beim Labelvorgang

### Sensor 3

Der nächste Sensor zählt die Palettenanzahl. Die vordefinierte Zählfunktion des Fließbandsensors zählt solange die Flaschen, bis der Wert von 800 (800 Flaschen = Palette) erreicht wird. Wenn sich die Anzahl der passierten Flaschen dem Wert 800 gleicht, wird batchReadySignal.signal auf True gesetzt, dadurch die Variable PaletteProp.Value vom Typ Integer um den Wert 1 erhöht und der Sensor zählt durch eine automatische Reset-Funktion wieder von 0.

```
if counterProp.Value == batchSizeProp.Value:
    batchReadySignal.signal(True)
    PaletteProp.Value += 1
```

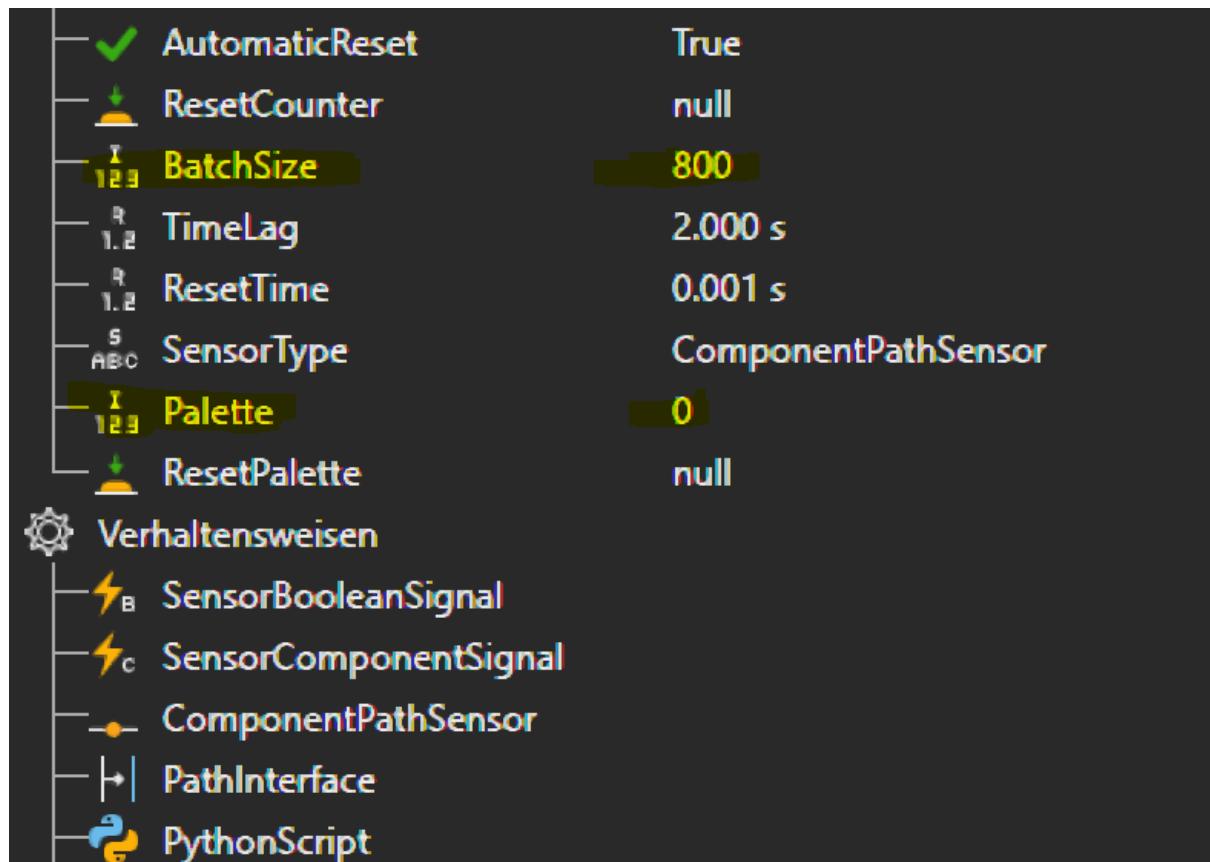


Abbildung 16: Eigenschaften des Palettenzählers

#### Sensor 4

Der letzte Fließbandsensor zählt die gesamte Flaschenanzahl nach dem Aussortieren, die verpackt werden. Dazu wird die interne Default-Zählfunktion des Sensors ohne automatisches Reset verwendet

#### **4.3.4 Aktoren und ihre Funktionalitäten**

Da unsere Projektzielsetzung einen bidirektionalen Datenaustausch erfordert, werden zusätzlich zu den Sensoren in der Fabrik auch Aktoren (auch Aktuator genannt) benötigt. Aktoren wandeln eingehenden Signale in mechanische Bewegungen um und greifen somit aktiv in den gesteuerten Prozess ein. Sie sind die Umkehrung der Sensoren.

Während Sensoren ihre Daten an den OPC-UA-Server schicken, empfangen die Aktoren diese von dem Server. man spricht von einer Bidirektionaler Kommunikation.

Gerät	Variable	Datentyp	Erklärung
<b>BottleBatchCapper</b>	Process-BatchSize	Int16	Wie viele Flaschen sollen gleichzeitig zugeschraubt werden? (min 1 – max 5)
<b>ReferenceFeeder</b>	Refere- nceFeeder	Boolesch	Eingehendes Signal erzeugt leere Flasche

Tabelle 2: Konfiguration der Aktoren

Aktor für das Zuschrauben der Flaschen:

Die Flaschen-Verschließer-Maschine kann bis zu 5 Flaschen gleichzeitig nach dem Befüllvorgang zuschrauben. Diese Zahl ist in eine Integer-Variable definiert und kann durch einen Eintrag auf dem UPC UA überschrieben werden (Wert zwischen 1-5 erforderlich).

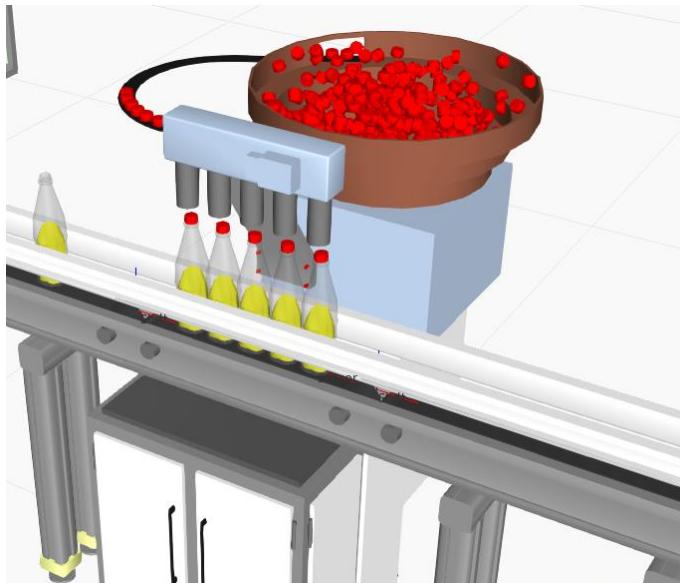


Abbildung 17: Flaschenverschlussmaschine

Aktor für das Kreieren der Flaschen:

Die leeren Flaschen werden durch sogenannten „ReferenceFeeder“ kreiert. Ein Boolesch-Signal (True) sorgt dafür, dass der Feeder eine neue leere Flasche auf das Förderband legt. Wenn die Variable „CreateOnlyOnSignal“ auf True gesetzt ist, muss zuerst ein Signal empfangen werden, bevor eine neue Flasche erstellt werden kann.

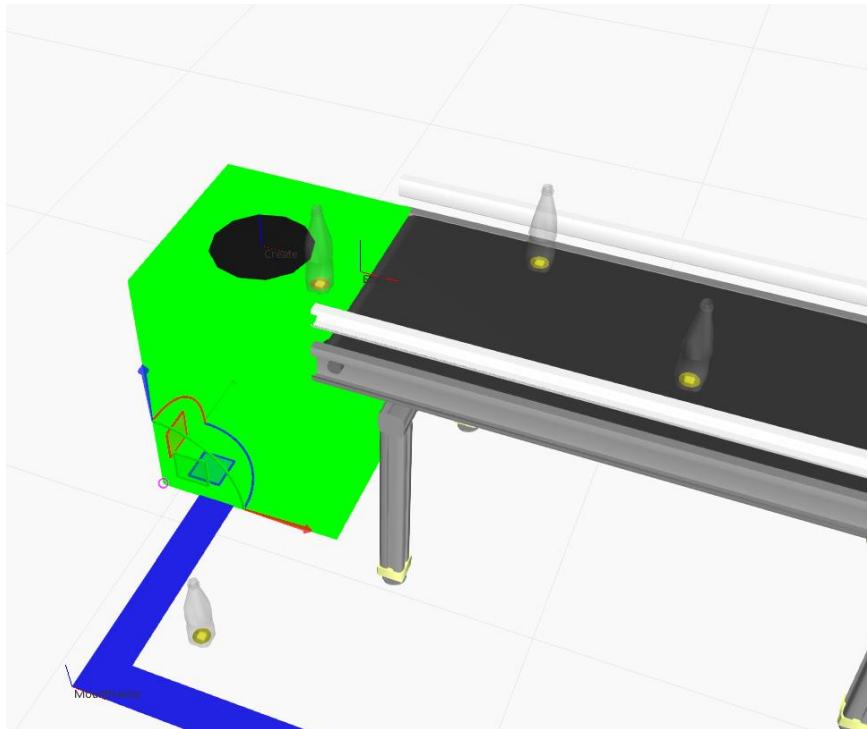


Abbildung 18: ReferenceFeeder beim Erstellen leerer Flaschen

#### 4.3.5 Konnektivität zum OPC UA C++ Demo-Server

Der Datenaustausch zwischen dem OPC-UA-Client und den Sensoren und Aktoren in Visual Components geschieht mittels Variablen.

Für die Konfiguration der Variablen muss im ersten Schritt der Reiter „Konnektivität“ in Visual Components sichtbar gemacht werden, um den OPC-UA-Server hinzufügen zu können. Dazu aktiviert man das Add-On „Konnektivität“ in den Optionen.

In dem Konfigurationspanel, unter dem Konnektivitätsreiter kann nun ein OPC-UA-Server eingetragen werden. Die Verbindung wird hergestellt, wenn der weiße Kreis grün ausgefüllt ist, ansonsten besteht keine Verbindung zum OPC-UA-Server.

Es ist eine Verbindung zum OPC-UA-Server namens „MeinServer“ eingerichtet.

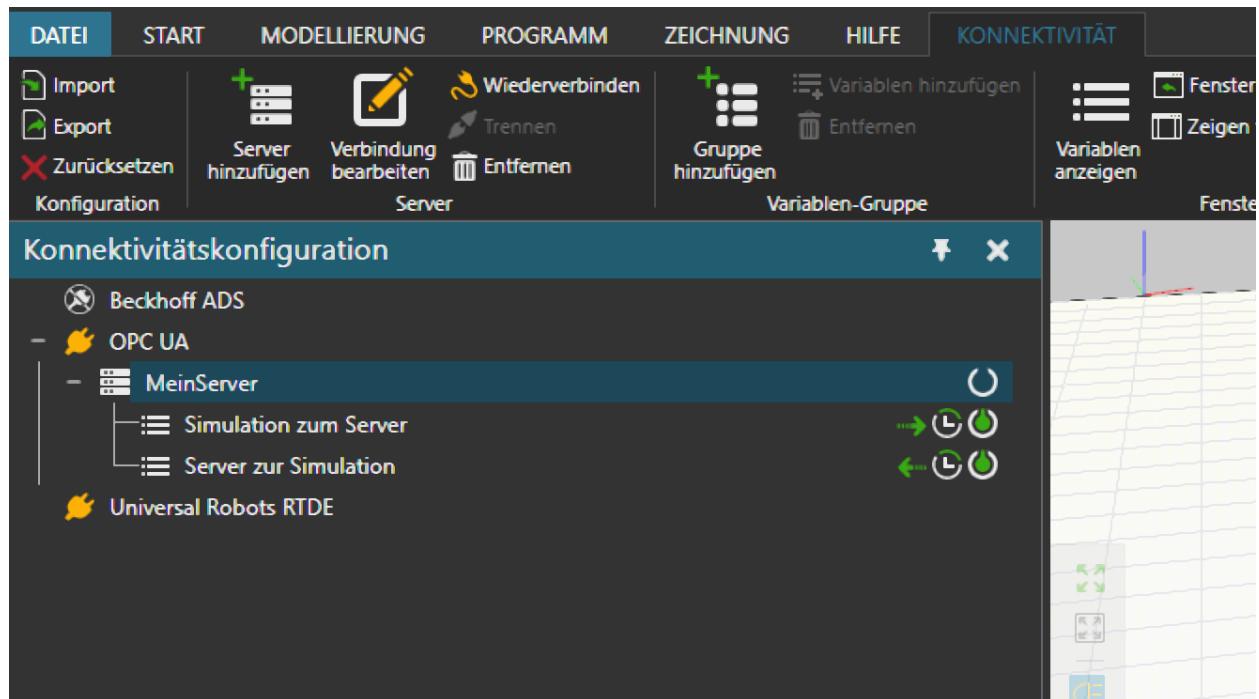


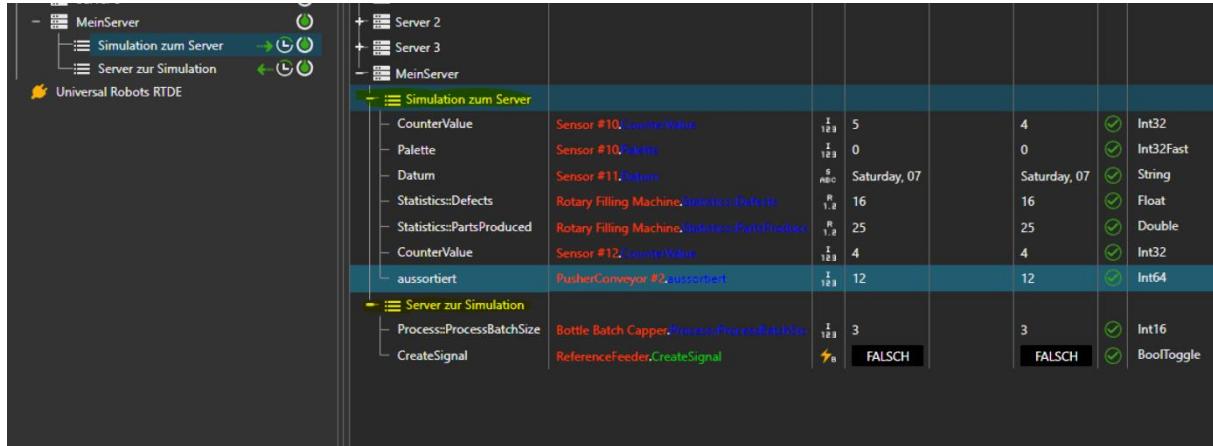
Abbildung 19: Server im Reiter "Konnektivität"

Durch die Bildung der Variablen-Paare werden die Werte der Sensoren und Aktoren zwischen dem OPC-UA-Server und Visual Components ausgetauscht. Wie man aus der Abbildung 22 entnehmen kann, sind auf der linken Seite die Variablen der Sensoren/Aktoren aufgeführt. Auf der Gegenseite finden sich die Variablentypen des OPC-UA-Servers. Ein Variablen-Paar muss vom gleichen Datentyp sein.

Variablen-Paare erstellen					Ausgewählter Server: MeinServer		
Inklusive:					Hinzufügen zur Gruppe: Simulation zum Server		
<input type="checkbox"/> Nur ausgewählte Komponenten <input checked="" type="checkbox"/> Komponenteneigenschaften <input type="checkbox"/> Verhaltenseigenschaften <input checked="" type="checkbox"/> Signale <input type="checkbox"/> Signalkarten <input checked="" type="checkbox"/> Freiheitsgrade							
.	Simulationsstruktur	Übergeordnet...	Variablenname	VariablenTyp	Server-Struktur	Datentyp	ServerTyp
PusherConveyor	PusherConveyor				Views		
Komponenteneigenschaften	PusherConveyor				Objects		
PushPartsWithNoProdID	PusherConveyor	PushPartsWithNo	Boolean		Server		
ConveyorWidth	PusherConveyor	ConveyorWidth	Real		Demo		
Pusher:RayOffset	PusherConveyor	Pusher:RayOffset	Real		PubSubTargetVariables		
Pusher::ShowRays	PusherConveyor	Pusher::ShowRays	Boolean		Publisher1		
Pusher::Length	PusherConveyor	Pusher::Length	Real		DataSet1		
Pusher::Thickness	PusherConveyor	Pusher::Thickness	Real		BoolToggle	Boolean	Boolean
Pusher::Height	PusherConveyor	Pusher::Height	Real		Int32	Int32	Int32
Pusher:ExitConveyorAngle	PusherConveyor	Pusher:ExitConve	Real		Int32Fast	Int32	Int32
Pusher:PushTo	PusherConveyor	Pusher:PushTo	String		DateTime		DateTime
Pusher:Stroke	PusherConveyor	Pusher:Stroke	Real		DataSet2		
Pusher:PushSpeed	PusherConveyor	Pusher:PushSpee	Real		DataSet3		

Abbildung 20: Bilden der Variablenpaare

Da Sensoren Daten versenden, werden die Sensoren-Variablen unter „Simulation zum Server“ angezeigt. Bei Aktoren werden keine Daten auf den OPC UA Server geschrieben, sondern von diesem Empfangen. Daher sind Aktoren unter „Server zur Simulation“ aufgeführt. Letzterer Hinweis kam in einem Telefongespräch vom Herrn Meyer, einem Lieferanten für Visual Components.



The screenshot shows the Universal Robots RTDE interface. On the left, there's a tree view of servers: 'MeinServer' (selected), 'Simulation zum Server', 'Server 2', 'Server 3', and 'Server zur Simulation'. Under 'Simulation zum Server', there are several variables listed in a table:

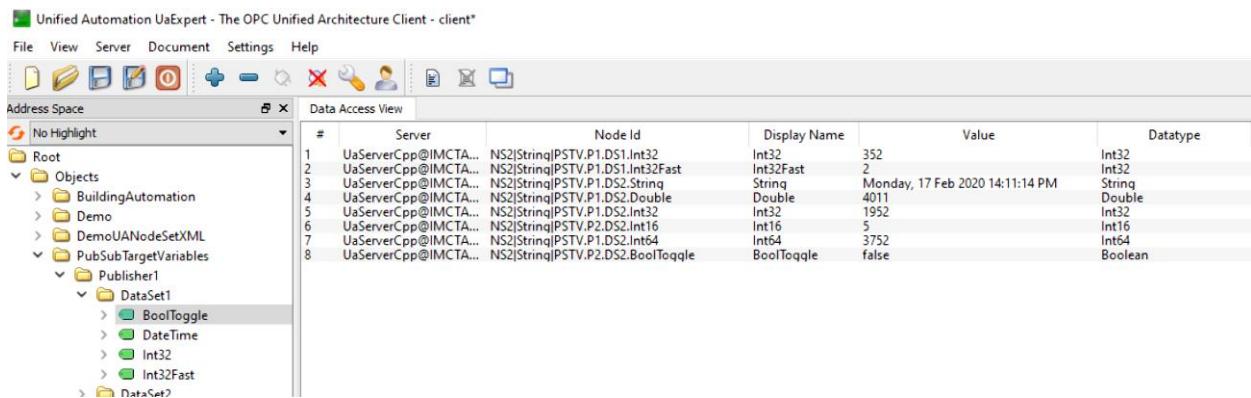
CounterValue	Sensor #10.CounterValue	<input type="text"/> 5	4	<input checked="" type="checkbox"/> Int32	
Palette	Sensor #10.palette	<input type="text"/> 0	0	<input checked="" type="checkbox"/> Int32Fast	
Datum	Sensor #11.Astum	<input type="text"/> Saturday, 07	Saturday, 07	<input checked="" type="checkbox"/> String	
Statistics::Defects	Rotary Filling Machine.Statistics::Defects	<input type="text"/> 16	16	<input checked="" type="checkbox"/> Float	
Statistics::PartsProduced	Rotary Filling Machine.Statistics::PartsProduced	<input type="text"/> 25	25	<input checked="" type="checkbox"/> Double	
CounterValue	Sensor #12.CounterValue	<input type="text"/> 4	4	<input checked="" type="checkbox"/> Int32	
aussortiert	PusherConveyor #2.aussortiert	<input type="text"/> 12	12	<input checked="" type="checkbox"/> Int64	

Under 'Server zur Simulation', there are two entries:

Process::ProcessBatchSize	Bottle_Batch_Capper.Demo::ProcessBatchSize	<input type="text"/> 3	3	<input checked="" type="checkbox"/> Int16
CreateSignal	ReferenceFeeder.CreateSignal	<input checked="" type="checkbox"/> FALSCH	FALSCH	<input checked="" type="checkbox"/> BoolToggle

Abbildung 21: Verwendete Variablen für Sensoren und Aktoren

Im UaExpert–Client selbst, ist die Verbindung mit den Visual Components Variablen anspruchslos. Nach dem Eintragen des Serverpfads müssen nur noch die Nodes aus der Ordnerhierarchie in das „Data Access View“ – Fenster per Drag & Drop gezogen werden.



The screenshot shows the Unified Automation UaExpert client. At the top, there's a menu bar with File, View, Server, Document, Settings, Help. Below it is a toolbar with various icons. On the left, there's a tree view of the address space under 'Address Space' tab, showing 'Root' and several objects like 'BuildingAutomation', 'Demo', 'PubSubTargetVariables', etc. On the right, there's a 'Data Access View' table:

#	Server	Node Id	Display Name	Value	Datatype
1	UaServerCpp@IMCTA...	NS2[String]PSTV.P1.DS1.Int32	Int32	352	Int32
2	UaServerCpp@IMCTA...	NS2[Int32]PSTV.P1.DS1.Int32Fast	Int32Fast	2	Int32
3	UaServerCpp@IMCTA...	NS2[String]PSTV.P1.DS2.String	String	Monday, 17 Feb 2020 14:11:14 PM	String
4	UaServerCpp@IMCTA...	NS2[Double]PSTV.P1.DS2.Double	Double	4011	Double
5	UaServerCpp@IMCTA...	NS2[Int32]PSTV.P1.DS2.Int32	Int32	1952	Int32
6	UaServerCpp@IMCTA...	NS2[Int16]PSTV.P2.DS2.Int16	Int16	5	Int16
7	UaServerCpp@IMCTA...	NS2[Bool]PSTV.P1.DS2.Bool	Int64	3752	Int64
8	UaServerCpp@IMCTA...	NS2[Bool]PSTV.P2.DS2.BoolToggle	BoolToggle	false	Boolean

Abbildung 22: Verwendete Variablen für Sensoren und Aktoren in UaExpert

## 4.4 Erstellung einer relationalen Datenbank

Eine relationale Datenbank soll dazu dienen erfasste Sensordaten, Vorratsdaten und Getränkebestellungen zu verwalten. Sie fungiert als Schnittstelle zwischen virtueller und realer, externer Welt. So sollen Bestellungsdaten, die über ein Dashboard, welches mit node-RED implementiert ist, eingegeben werden, Einfluss auf die Datenbank haben. Im gleichen Zuge ist ein Vorratslager mit fiktiven Stückzahlen von Produktionskomponenten implementiert. Ist die Simulation der Limonaden-Abfüllfabrik im aktiven Betrieb, so wird in Echtzeit der Lagerbestand je produzierte Flasche verringert.

Die relationale Datenbank ist in der Sprache SQL implementiert. Zur Verwaltung dienen die kostenfreien Anwendungen phpMyAdmin und MySQL Workbench. Erstere dient zum direkten Eingreifen und Letztere, um den eigentlichen Code zu implementieren. Die Datenbank wird mit der Befehlszeile `CREATE DATABASE IF NOT EXISTS `BottleFillingLayout` CHARACTER SET utf8 COLLATE utf8_unicode_ci;` angelegt. Das Kommando `USE `BottleFillingLayout`` wird angewendet, um im Code nicht vor jedem Statement explizit die Datenbank erwähnen zu müssen.

### 4.4.1 Konfiguration der Datenbank und Sicherung der Datenintegrität

Nachdem die Datenbank erfolgreich angelegt ist, sind entsprechende Tabellen zu erstellen. Es ist jeweils eine Tabelle für produzierte Flaschen, für defekt befüllte Flaschen, für Kundenbestellungen und für ein Vorratslager anzulegen. Diese tragen die Namen `Limonaden_Produziert`, `Limonaden_Defekt`, `Bestellungen` und `Vorratslager` und sind mit dem SQL-Statement `CREATE TABLE IF NOT EXISTS `Tabellen-Name`` zu erschaffen.

```

1   #Anlegen der Datenbank für Sensordaten-----#
2 • CREATE DATABASE IF NOT EXISTS `BottleFillingLayout`
3   CHARACTER SET utf8 COLLATE utf8_unicode_ci;
4
5 • USE `BottleFillingLayout`;
6
7   #Tabelle wurde mittels DROP TABLE wieder gelöscht-----#
8   #-----#
9   #Erstellung der endgültigen Tabellenaufteilung-----#
10
11 • CREATE TABLE IF NOT EXISTS `Limonaden_Produziert`
12   ( `FlaschenID` INT NOT NULL,
13     `FlaschenID_Palette` INT NOT NULL,
14     `PalettenID` INT NOT NULL,
15     `Abfuell-Datum` VARCHAR(30) NOT NULL);
16
17 • CREATE TABLE IF NOT EXISTS `Limonaden_Defekt`
18   ( `FlaschenID` INT NOT NULL,
19     `Abfuell-Datum` VARCHAR(30) NOT NULL);
20
21 • CREATE TABLE IF NOT EXISTS `Bestellungen`
22   ( `Name` VARCHAR(50) NOT NULL,
23     `Palettenanzahl` INT NOT NULL);
24
25 • CREATE TABLE IF NOT EXISTS `Vorratslager`(
26   `Leere Flaschen` INT NOT NULL,
27   `Flaschendeckel` INT NOT NULL);
28

```

Abbildung 36: Anlegen der Datenbank samt Tabellen

Hinter den Spaltennamen ist dessen Datentyp anzugeben. Außerdem unterbindet das Statement NOT NULL das Einfügen leerer Datensätze. Wie im obigen Kapitel beschrieben, sind der Tabelle Vorratslager fiktive Werte zu übergeben. Für den ersten Systemlauf ist ein Wert von 50000 für die Spalten Leere Flaschen und Flaschendeckel einzutragen. Dieser Wert kann nach Belieben gelöscht und durch einen anderen ersetzt werden.

```

29   #Befüllen des Lagers mit Komponenten
30 • INSERT INTO `Vorratslager`(`Leere Flaschen`, `Flaschendeckel`)VALUES
31   (50000,50000);

```

Abbildung 23: Auffüllen des Vorratslagers

Um die Datenintegrität zu sichern, sind entsprechende Unique-Schlüssel auf die Spalten der Tabellen Limonaden\_Produziert und Limonaden\_Defekt zu legen. Dies geschieht über die SQL-Statements ALTER TABLE und ADD UNIQUE. Die Vergabe dieses Schlüssels bezweckt, dass jede Komponente in der Datenbank eindeutig identifizierbar ist.

```

80  ALTER TABLE `bottlefillinglayout`.`limonaden_produziert`
81  ADD UNIQUE `Unique_Limonaden_Produziert`(`FlaschenID`, `FlaschenID_Palette`, `PalettenID`, `Abfuell-Datum`);
82
83  ALTER TABLE `bottlefillinglayout`.`limonaden_defekt`
84  ADD UNIQUE `FlaschenID`(`FlaschenID`, `Abfuell-Datum`);
```

Abbildung 24: Vergabe der Unique-Schlüssel

Damit bei Simulationsbetrieb aus jenem Vorratslager Elemente abgezogen werden, muss für die Datenbank entsprechende Trigger konfiguriert sein. Diese müssen, wenn ein neuer Datensatz in die Tabellen Limonaden\_Produziert oder Limonaden\_Defekt eingefügt wird, jedes Mal ein Element von jeweils beiden Spalten der Tabelle Vorratslager abziehen. Um diese Funktionalität zu realisieren sind zwei Trigger nötig.

```

33  #Erstellung der Trigger zur Wertänderung der Tabelle Vorratslager
34  DELIMITER |
35 • CREATE TRIGGER lager_flaschen_abzug
36  BEFORE INSERT ON `Limonaden_Produziert`
37  FOR EACH ROW
38  BEGIN
39    INSERT INTO `Vorratslager`(`Leere Flaschen`, `Flaschendeckel`)
40    SELECT MIN(`Leere Flaschen`)-1, MIN(`Flaschendeckel`)-1
41    FROM `Vorratslager`;
42  END |
43  DELIMITER ;
44
45  DELIMITER |
46 • CREATE TRIGGER lager_flaschen_abzug2
47  BEFORE INSERT ON `Limonaden_Defekt`
48  FOR EACH ROW
49  BEGIN
50    INSERT INTO `Vorratslager`(`Leere Flaschen`, `Flaschendeckel`)
51    SELECT MIN(`Leere Flaschen`)-1, MIN(`Flaschendeckel`)-1
52    FROM `Vorratslager`;
53  END |
54  DELIMITER ;
```

Abbildung 25: Trigger für die Tabelle Vorratslager

Die Verwendung des SELECT-Statements ist nötig, um Gruppenfunktionen, wie MIN() nutzen zu können. Andernfalls würde dieser Trigger eine Fehlermeldung hervorrufen. Die Trigger bewirken, dass vor jedem neu eingetragenen Datensatz in die Tabellen Limonaden\_Produziert und Limonaden\_Defekt eine Wertveränderung in den Spalten der Tabelle Vorratslager.

Die Konfiguration eines weiteren Triggers ist nötig, damit bei einer Kundenbestellung die entsprechende Stückzahl von der Tabelle Limonaden\_Produziert abgezogen wird. Kunden können palettenweise bestellen. Eine Palette enthält 800 Flaschen.

```

59      #Erstellen eines Triggers zur Wertveränderung der Tabelle Limonaden_Produziert
60  •  DELIMITER |
61   CREATE TRIGGER bestellungen_abzug
62     BEFORE INSERT ON `Bestellungen`
63       FOR EACH ROW
64     BEGIN
65       SET @FLASCHENANZAHL := 1;
66       SET @FLASCHENANZAHL = NEW.`Palettenanzahl` * 800;
67       #Löschen der Flaschen und Paletten
68     WHILE @FLASCHENANZAHL > 0
69       DO
70         DELETE FROM `Limonaden_produziert` WHERE `FlaschenID` IN (
71           SELECT MIN(`FlaschenID`));
72         SET @FLASCHENANZAHL = @FLASCHENANZAHL - 1;
73       END WHILE;
74     END |

```

Abbildung 26: Anlegen des Triggers für Bestellungen

Bevor ein neuer Datensatz in die Tabelle `Bestellungen` eingepflegt wird, wird die Anzahl der Paletten ermittelt und mit dem Wert 800 multipliziert. In der Schleife darunter wird jedes Datenelement von `Limonaden_Produziert` gelöscht und der Wert der Variable `FLASCHENANZAHL` dekrementiert. Dieser Vorgang wiederholt sich, bis der Wert der Variable nicht mehr größer null ist.

Nach Eingabe des kompletten Quellcodes ist dieser noch auszuführen und die Datenbank mit samt ihrer kompletten Funktionalität ist implementiert und einsatzbereit. Jegliche Beziehungen und Trigger für die Tabellen untereinander sind mit dem OPC Router zu konfigurieren. Daher sind auf dem ER-Modell keine Beziehungen verzeichnet, lediglich die Schlüssel-Symbole geben an, dass auf einer oder mehreren Spalten ein Unique-Key vergeben ist.



Abbildung 27: ER-Modell der Datenbank

## 4.5 Installation und Konfiguration des OPC Router

Nachdem auf die Forderung nach einem Download-Link für OPC Router geantwortet wurde, ist die Anwendung herunterzuladen und auf dem Hostsystem zu installieren. Die Installation unter Windows verlangt die Rechte des Administrators. Ist Software installiert, geht es an die eigentliche Konfiguration der Plattform. Auf der linken Seite des Bildschirmfensters befindet sich ein Auswahlmenü namens *Plug-ins*. In diesem können, je nach Menüpunkt, verschiedene Andockstellen konfiguriert werden.

### 4.5.1 Konfiguration des OPC-UA-Plug-ins

Zunächst ist im Plug-ins-Menü der Punkt *OPC-UA-Server* auszuwählen, um die Konfiguration jener Schnittstelle vorzunehmen. Durch Mausklick auf das *Plus-Symbol* lassen sich neue *OPC-UA-Server-Plug-in-Instanzen* erstellen. In dem sich öffnenden Fenster sind die Rahmendaten des OPC-UA-Demo-Servers einzutragen. Durch anschließenden Mausklick auf den Button *Verbindung testen* verifizieren Sie die Verbindung. Mit darauf folgendem Klick auf *OK* bestätigen Sie die Konfiguration und schließen diese somit ab.

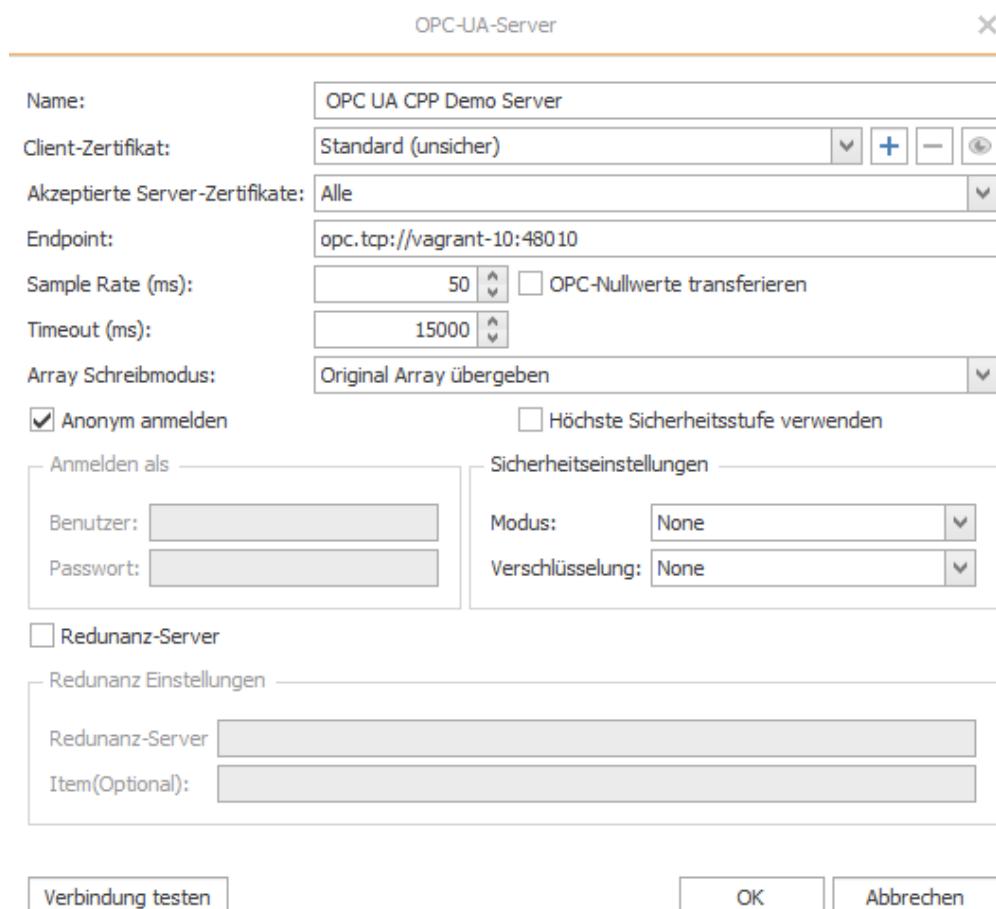


Abbildung 25: Konfiguration der Serverinstanz

#### 4.5.2 Konfiguration des MySQL-6-Plug-ins

Nachdem die Schnittstelle zum OPC-UA-Demo-Server konfiguriert ist, ist jene zur relationalen Datenbank einzustellen. Hierfür ist im Plug-ins-Menü der Punkt *MySQL 6* auszuwählen. Auch hier ist über das jeweilige *Plus-Symbol* eine neue *Plug-in-Instanz* zu erstellen und diese zu konfigurieren. Hier sind Name der Datenbank, IP-Adresse, Portnummer und der Benutzer anzugeben. Die Verbindung ist mit Klick auf *Verbindung testen* zu verifizieren und die Konfiguration mit Klick auf *OK* abzuschließen.

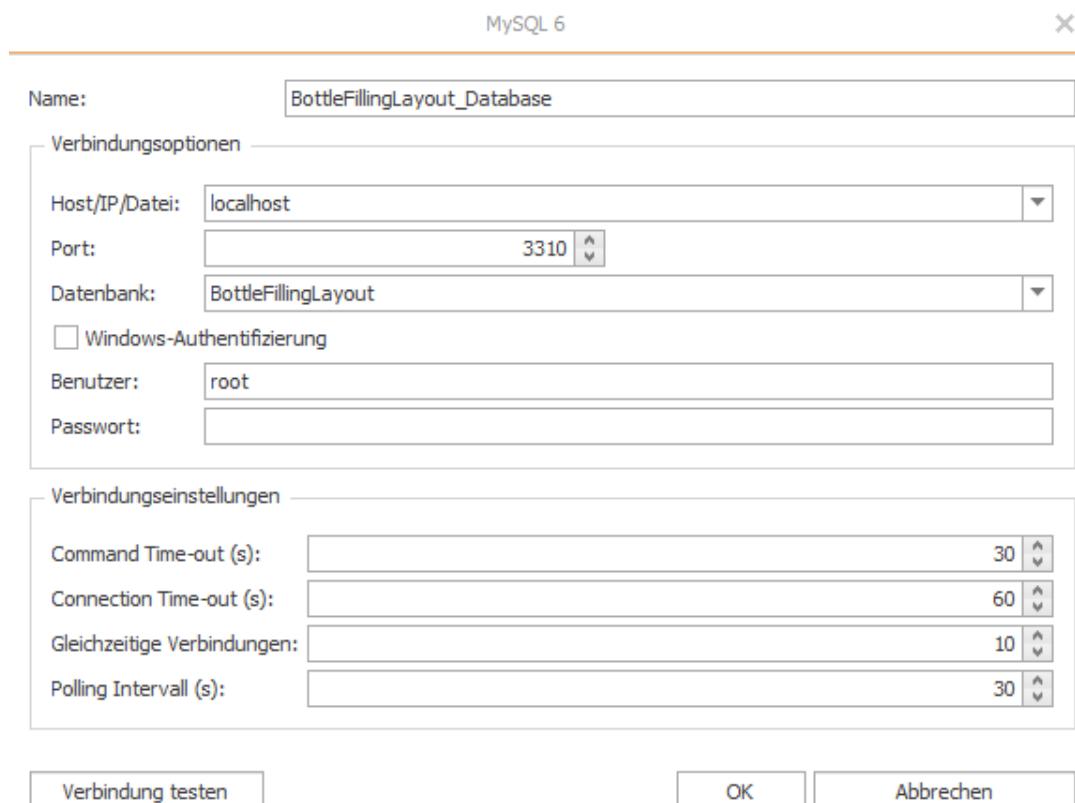


Abbildung 26: Konfiguration der Datenbankinstanz

#### 4.5.3 Konfiguration der Verbindung „StorageRequest\_Produziert“ mit Trigger

Damit die erfassten Daten aus der Limonaden-Abfüllfabrik über den OPC-UA-Demo-Server in die Datenbank eingepflegt werden können, ist es nötig eine Verbindung zu erstellen. Dies geschieht über den Menüpunkt *Verbindungen* auf der linken Seite. Im Menü angekommen ist ein *Ordner* für mehrere Verbindungen anzulegen. Durch Mausklick auf das orangene *Pfeil-Symbol* wird eine *neue Verbindung* erstellt. Auf der rechten Seite befindet sich ein weiteres Menü mit *Transferobjekten*. Aus ihm werden die Transferobjekte *OPC UA/DA* und *Datenbank* per drag and drop in den Editierbereich, welcher sich in der Mitte befindet, gezogen. Nachdem sich per Doppelklick auf das *OPC UA/DA Transferobjekt* im Editierbereich ein Fenster öffnet, ist in dieses der OPC-UA-Demo-

Server als Anbindung einzutragen. Über den Button *Tag-Browser* besteht die Möglichkeit die Serverhierarchie einzusehen und beliebige Variablen zu verwenden. Hier werden nun die in Kapitel 4.2.5 beschriebenen Variablen mit ihren Node-IDs eingetragen.

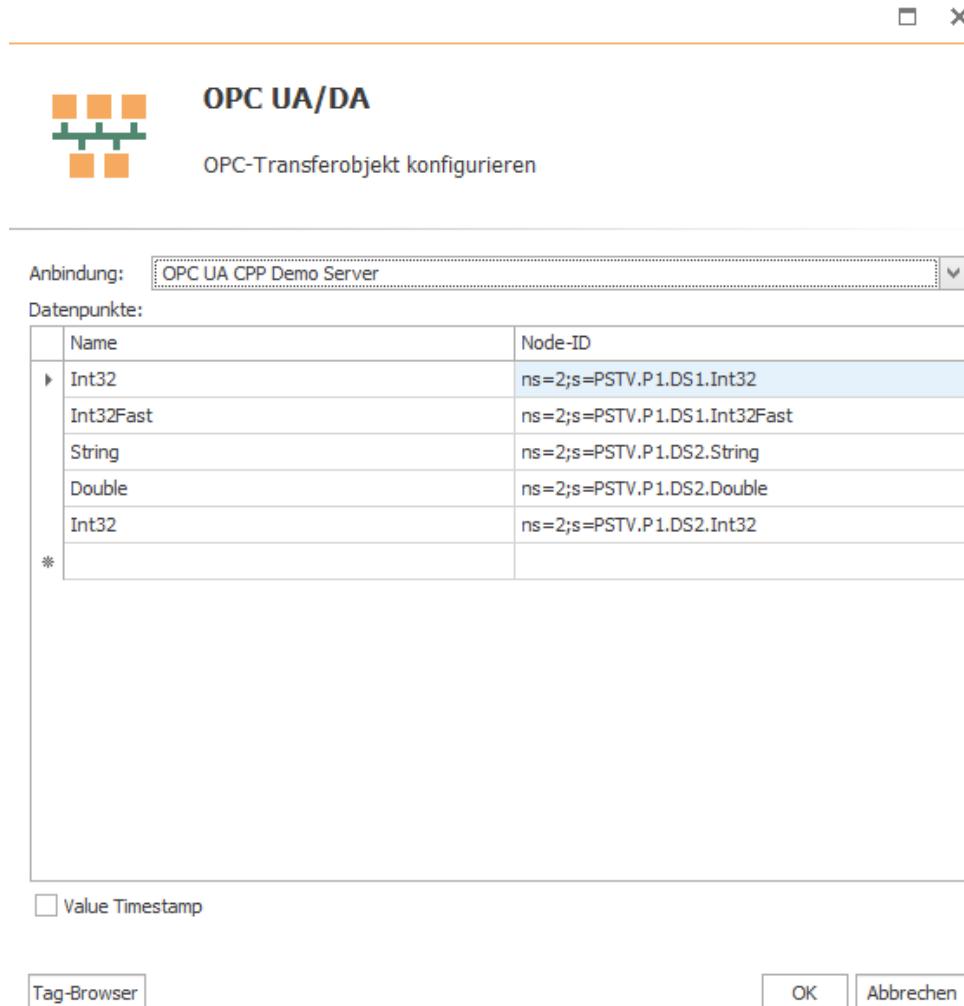


Abbildung 27: Konfiguration des OPC-Transferobjekts

Nachdem sich per Doppelklick auf das *Datenbank-Transferobjekt* ebenfalls dessen Konfigurationsfenster öffnet, sind auch dort die DB-Anbindung samt den zu verwendenden Tabellen und Spalten anzugeben. Des Weiteren ist ein sogenannter Typ einzustellen. Hier kann zwischen den MySQL-Statements `INSERT`, `UPDATE`, `SELECT`, `DELETE` oder `STOREDPROCEDURE` gewählt werden. Außerdem kann eine Angabe über die erwarteten Datensätze angegeben werden. Diese Einstellung ist allerdings nicht zwingend notwendig. Für die Produktivschaltung der Verbindung `StorageRequest_Produziert` ist die Angabe der Tabelle `Limonaden_Produziert` samt ihrer Spalten erforderlich. Mit Mausklick auf den *OK-Button* ist die Konfiguration abgeschlossen.

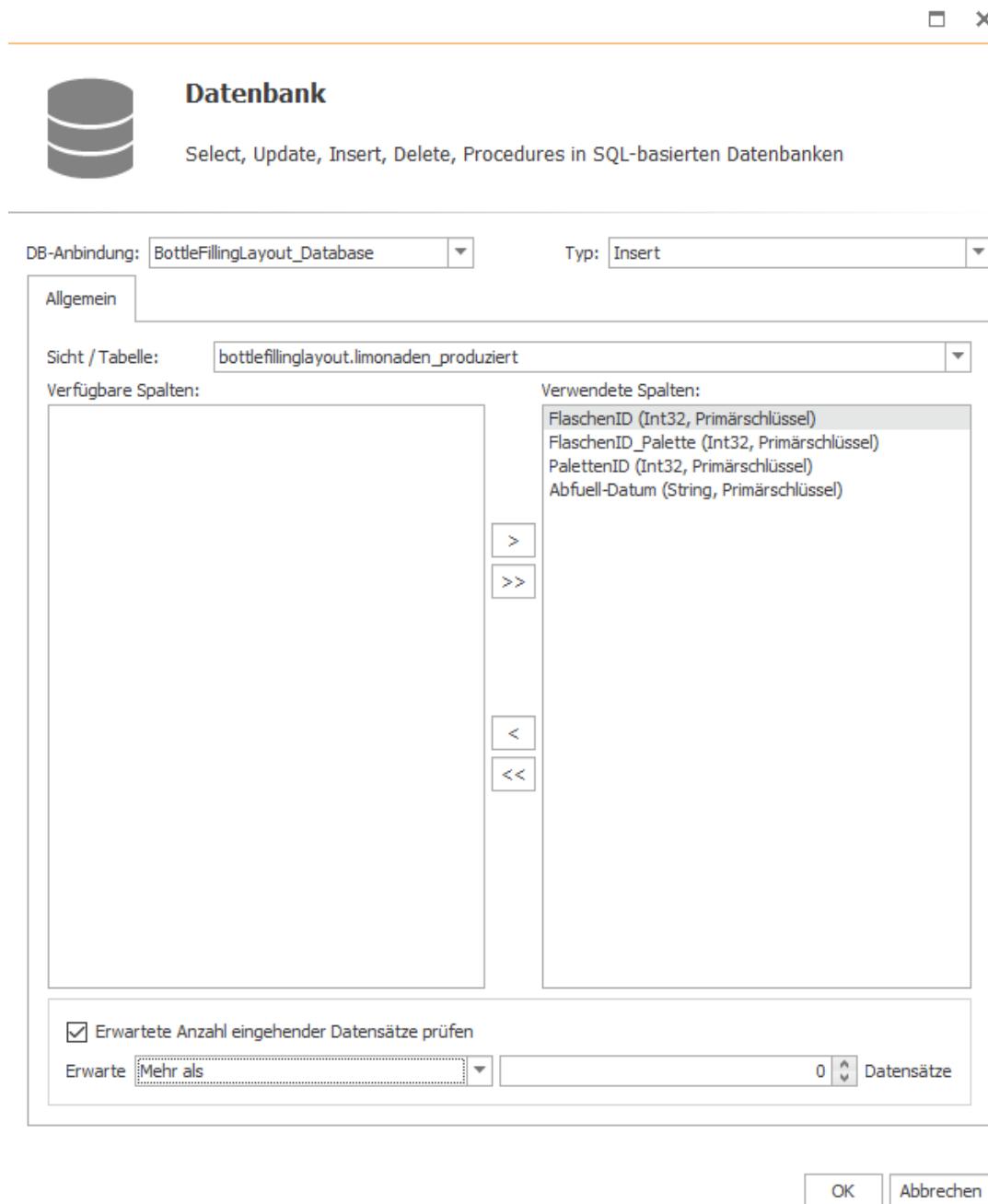


Abbildung 28: Konfiguration des Datenbank-Transferobjekts

Nachdem die beiden Transferobjekte mit ihren jeweiligen Variablen und Spalten konfiguriert sind, sind diese mittels Cursors miteinander zu verbinden. Zu jeder Variable im OPC-UA-Server ist ein *Pfeil* zu einer Spalte der Datenbank zu ziehen. Damit der OPC-UA-Demo-Server weiß, wann er der Datenbank Datensätze zum Einpflegen übergeben soll, ist ein *OPC-Datachange-Trigger* aus dem Transferobjektemenü zu entnehmen und zu konfigurieren. In dessen Konfigurationsmenü ist die Variable von Typ Int32, welche für die Zählung der produzierten Flaschen zuständig ist, als Triggervariable einzustellen. Jedes Mal, wenn sich der Wert dieser Variable ändert, bekommt der OPC-UA-Demo-Server die Anweisung, der Datenbank einen Datensatz zu übergeben. Diese

Wertveränderung findet immer dann statt, wenn eine Flasche den entsprechenden Sensor überquert.

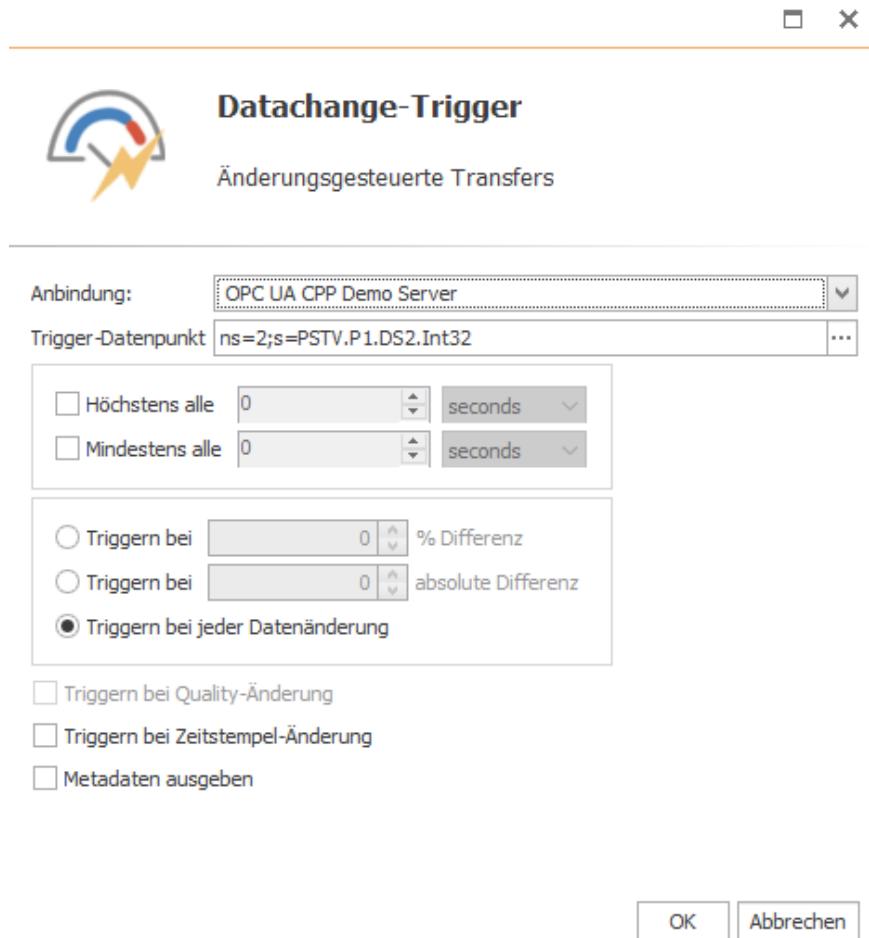


Abbildung 29: Konfiguration des Datachange-Triggers

Nach der Triggerkonfiguration sind die Einstellungsarbeiten für die Verbindung StorageRequest\_Produziert abgeschlossen. Diese kann nun produktiv geschalten werden und in Echtzeit arbeiten.

#### 4.5.4 Konfiguration der Verbindung „StorageRequest\_Defekt“ mit Trigger

Wie im vorherigen Kapitel die Verbindung StorageRequest\_Produziert konfiguriert ist, ist auch die Verbindung StorageRequest\_Defekt ähnlich einzustellen. Auch hier ist es wieder nötig, Transferobjekte vom Typ OPC UA/DA, Datachange-Trigger und Datenbank mittels Cursors in den Editerbereich zu ziehen. Für das OPC-Transferobjekt sind die Variablen aus Kapitel 4.2.5 zu verwenden. Für defekt produzierte Flaschen genügt es eine Int64- und eine String-Variable zu verwenden, da für diese Verbindung nur FlaschenID und Abfülldatum benötigt werden. Die Variablen sind über den Tag-Browser zu wählen.

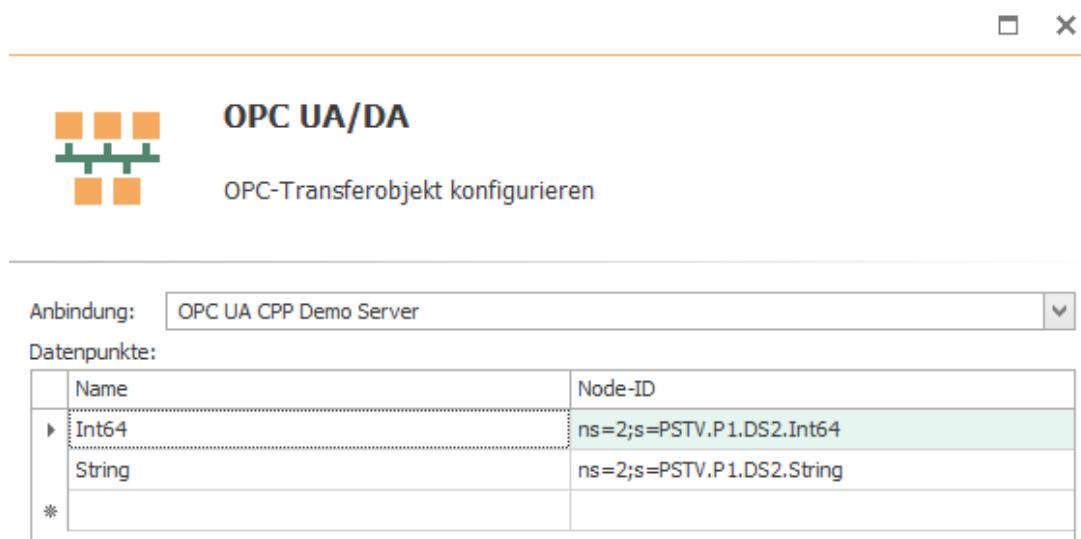


Abbildung 30: Konfiguration des OPC-Transferobjekts

Ist die Konfiguration des OPC-Transferobjekts abgeschlossen, ist die Einstellung des Datenbank-Transferobjekts durchzuführen. Hierfür ist die Tabelle `Limonaden_Defekt` samt den Spalten `FlaschenID` und `Abfuell-Datum` zu verwenden. Auch hier ist die Einstellung der erwarteten Datensätze nicht erforderlich.

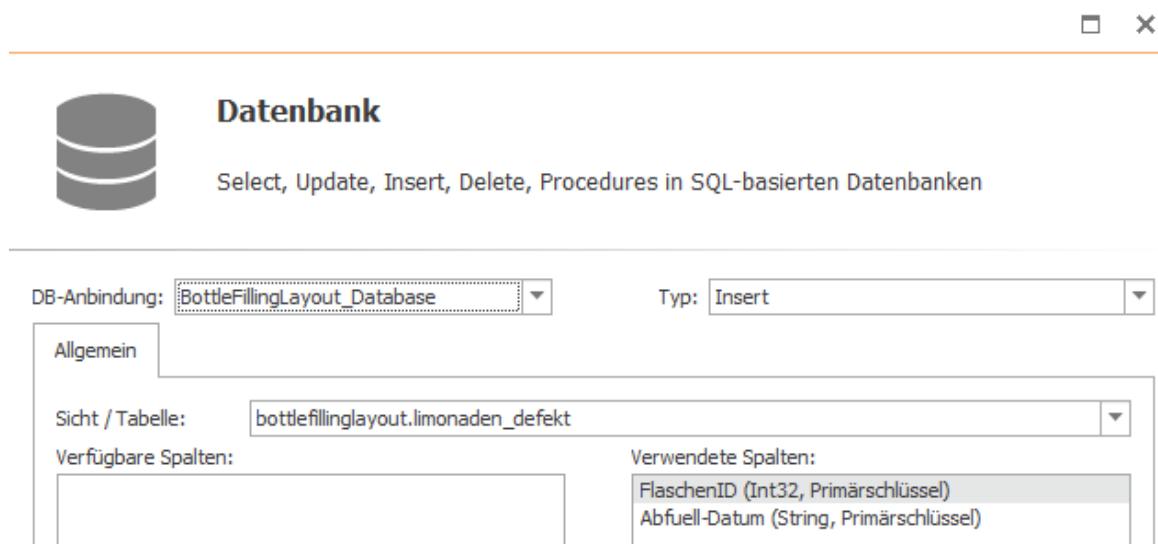


Abbildung 31: Konfiguration des Datenbank-Transferobjekts

Letztendlich ist noch der Datachange-Trigger zu konfigurieren. Hier ist die Variable vom Typ Int64 zu verwenden. Die Einstellung bezweckt, dass bei Wertänderung der Variable, dem Server ein Signal zur Datenübertragung gegeben wird. Nach Abschluss der Triggereinstellungen sind noch die Variablen des Servers per Cursor mit den Spalten der Tabelle zu verbinden.

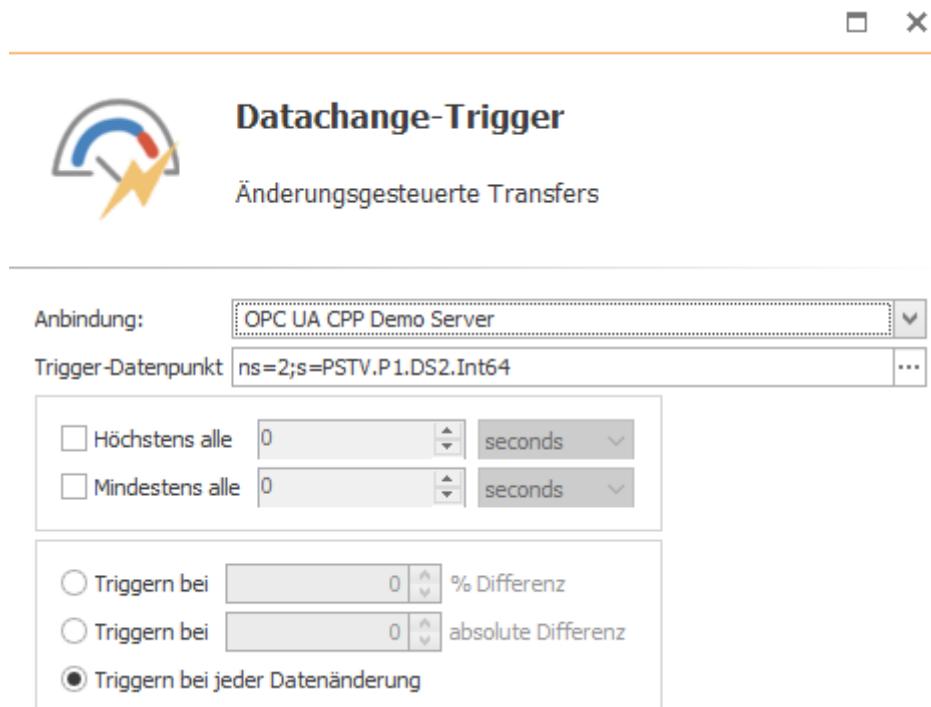


Abbildung 32: Konfiguration des Datachange-Triggers

#### 4.5.5 Produktivschaltung und Test der Verbindung

Sind die beiden Verbindungen komplett konfiguriert, ist über den sich oben befindlichen *Verbindung-produktiv-schalten-Button*, diese für den laufenden Betrieb freizuschalten. Diese können jeweils einzeln, oder aber durch vorheriges markieren zusammen produktiv geschalten werden. Durch Mausklick auf den Button *Status* auf der linken Seite, ist es möglich die Verbindungen im Monitoring-Verfahren zu überwachen. Hier werden auch, falls aufgetreten, Konfigurationsprobleme angezeigt. Die Übertragung der Datensätze vom OPC-UA-Demo-Server zur relationalen Datenbank können hier in Echtzeit verfolgt werden. Für jeden Datensatz ist eine genaue Dauer dessen Einpflege zu entnehmen. Dies ist nützlich, um auf eventuelle Fehler in der Datenbank oder dem Server aufmerksam zu werden. Über das linke Auswahlmenü ist zwischen den beiden Verbindungen zu wählen.

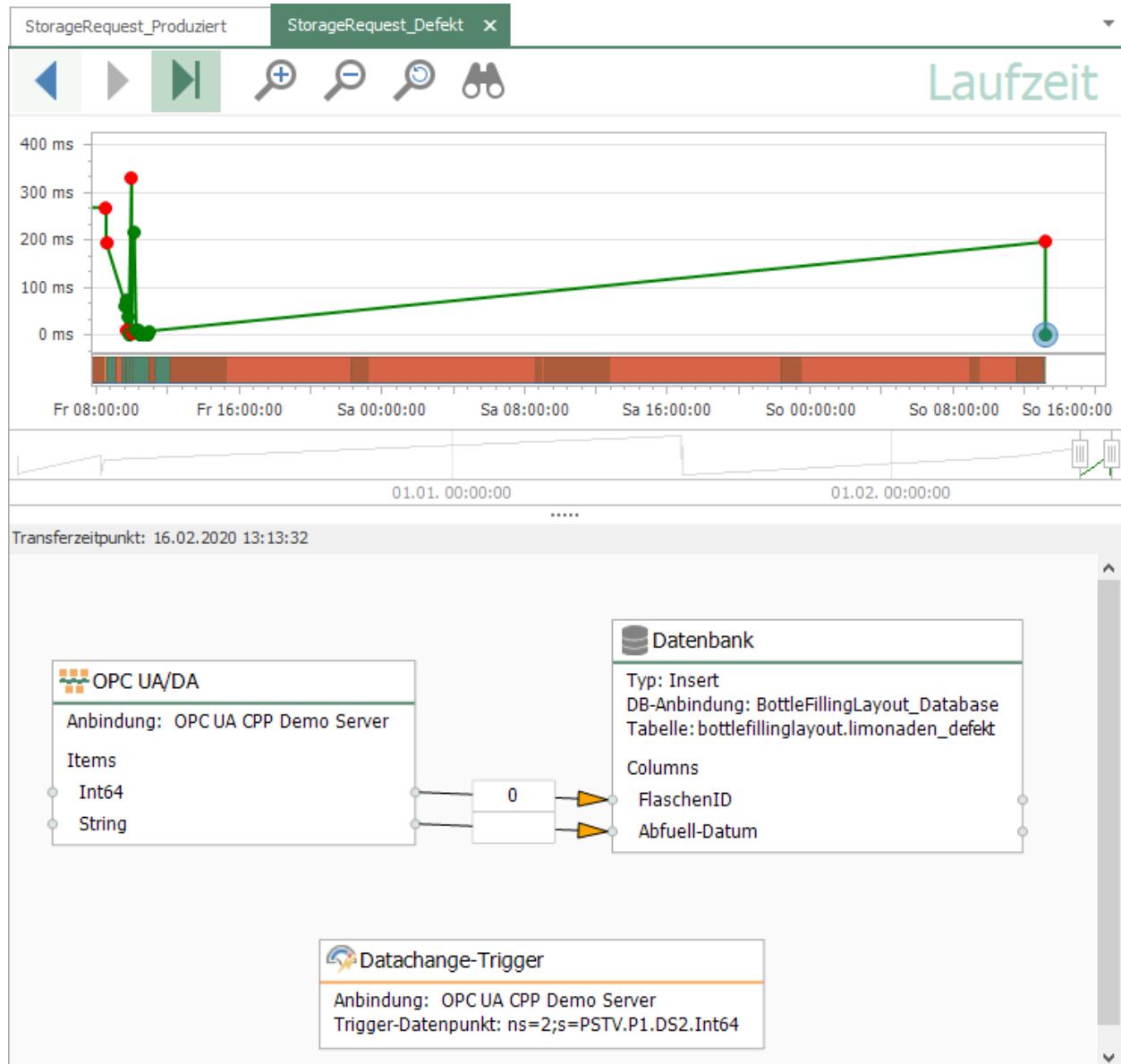


Abbildung 33: Die Verbindung StorageRequest\_Defekt beim Monitoring

## 4.6 Installation und Konfiguration von node-RED

Zuerst muss das node-RED-Installationspaket heruntergeladen werden. Danach muss für die Windowsinstallation in der Eingabekonsole folgender Befehl ausgeführt werden:

```
npm install -g --unsafe-perm node-red
```

Als nächstes muss der node-RED-Dienst mit folgendem Konsolenbefehl gestartet werden:

```
node-red
```

Die grafische Konfigurationsoberfläche ist nach der Standardinstallation unter folgender Defaultadresse im Internetbrowser zu erreichen:

`http://127.0.0.1:1880/`

#### 4.6.1 Konfiguration der nodes

Zwar sind viele Bausteine für die gängigsten Dienste und Technologien im IoT-Bereich nach der Installation vorhanden, jedoch müssen für unsere Zwecke weitere Baustein-pakete installiert werden. Zur Erfüllung des Projektziels müssen unsere generierten Daten aus der Fabrik grafisch dargestellt werden. Dazu werden Nodes benötigt, die eine Verbindung zu der MySQL-Datenbank herstellen, die Datensätze aus Datenbank importieren, weiterverarbeiten und danach grafisch visualisieren können.

Es besteht auch die Möglichkeit, die Fabrikdaten direkt in node-RED aus dem OPC UA -Server zu importieren, ohne sie zuerst in Datenbanken abzuspeichern. Diese Option kann als Echtzeitauswertung genutzt werden, da hier keine Datensätze gespeichert und möglicherweise alte Einträge wieder abgerufen werden. Nach jedem Neustart der Fabrik werden die Daten und damit verbundene grafische Darstellungen zurückgesetzt. Diese Konfiguration (Punkt 1.1.2 ) ist kein Teil der Projektvereinbarung.

Folgende Pakete wurden nachinstalliert:

node-RED-dashboard	Das Dashboard-Modul bietet eine Reihe von Knoten, um ein Live-Daten-Dashboard zu erstellen. Dieses zeigt die Daten der Nodes grafisch in Diagrammen in Echtzeit an. Konsolenbefehl: <code>npm install node-RED-dashboard</code>
node-RED-node-mysql	Dieses Paket ermöglicht den grundlegenden Zugriff auf eine MySQL-Datenbank. Dieser Knoten verwendet die Abfrageoperation gegen die konfigurierte Datenbank. Dies erlaubt sowohl INSERT- als auch DELETE-Befehle. Konsolenbefehl: <code>npm install node-RED-node-mysql</code>
node-RED-contrib-opcua	Node-RED-Knoten zur Kommunikation oder Bedienung über OPC UA. Es kann verwendet werden, um Variablen zu definieren. Der OpcUa-Client-Node kann zum Lesen, Schreiben, Abonnieren, Browsen des OPC UA Servers verwendet werden. Konsolenbefehl: <code>npm install node-RED-contrib-opcua</code>

#### 4.6.2 Konfiguration der Verbindung zu Visual Components

Für die Echtzeit- Auswertung der Fabrikdaten im Browser muss eine direkte Verbindung zwischen node-RED und dem OPC-UA-Server hergestellt werden und sog. widget-Nodes für die visuelle Darstellung der Werte eingerichtet werden. Mittels des OPC-UA-Browser – Nodes kann eine Verbindung zum OPC-UA-Server hergestellt werden und die benutzten OPC-UA-Adressen herausgelesen werden.

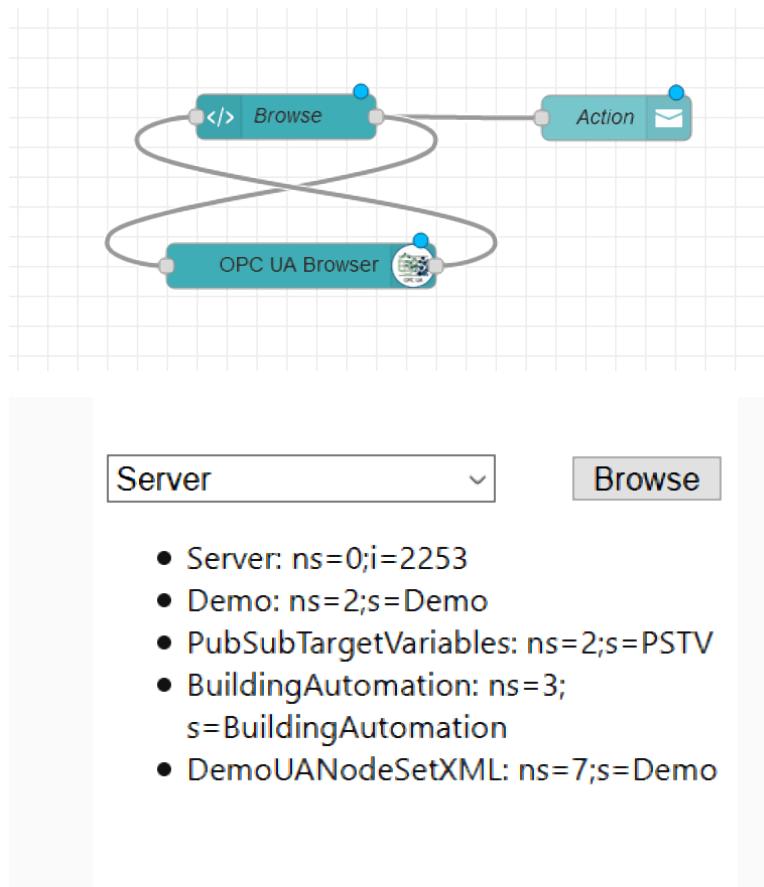


Abbildung 28: Flow für die Wertauslesung der Variablen des OPC-UA-Servers

Die Visual-Components-Variablen sind für den inject-Node notwendig. Dieser startet den Flow, mit der eingetragener OPC-UA-Adresse als optionale Eigenschaft (sog. Topic). Die Adresse wird als nächstes an den OPC-UA-Client-Node weitergegeben, der die Verbindung zum OPC-UA-Server herstellt und den Wert der Adresse auf dem OPC-UA-Server herausliest. Dieser Wert wird anschließend an den letzten, sog. Widget-Node weitergegeben, der den Wert in visueller Form darstellt. Für die Darstellung stehen 4 Modi zur Verfügung: Standard (einfacher Pegel), Donut (komplette 360°), Kompass und Welle. Der gesamte Prozess wird jede Sekunde wiederholt, damit die Grafik sekundenweise aktualisiert wird.

Insgesamt sind drei solche Flows konfiguriert: für befüllte Paletten, Flaschen auf einer Palette und aussortierten Flaschen.

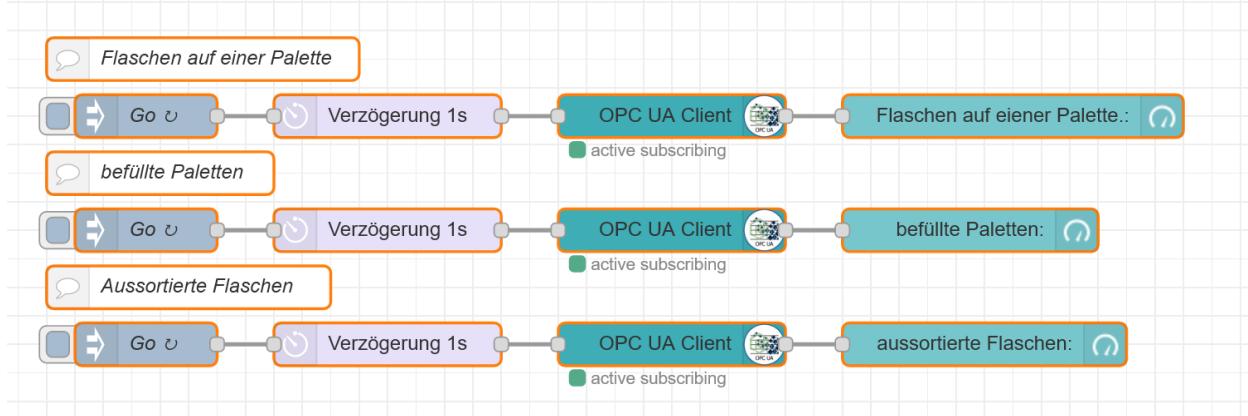


Abbildung 29: Nodes für den Live-View

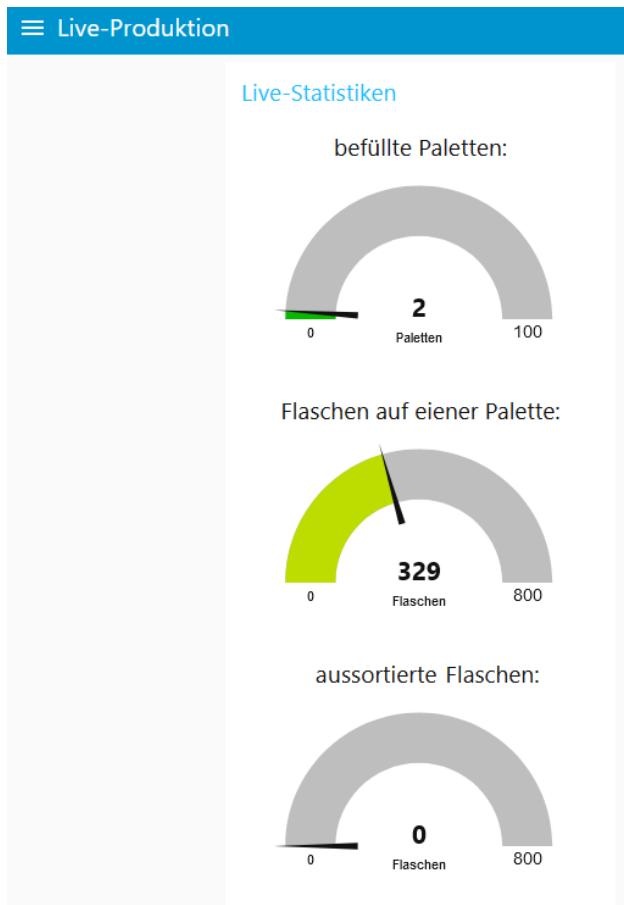


Abbildung 30: Echtzeitstatistiken im Dashboard

Nicht gelöstes Problem:

Obwohl der OPC-UA-Server-Variablenpfad richtig eingetragen ist, wird die Anzahl der aussortierten Flaschen nicht angezeigt.

#### 4.6.3 Konfiguration der Verbindung zur relationalen Datenbank

Einer der primären Ziele des Projekts beinhaltet die grafische Darstellung der exportierten Daten in Verbindung mit Datenbanken. Diese Konfiguration ist mit folgendem Setup realisiert:

Es ist ein inject-Node eingerichtet, der zwar keine Nutzerdaten hat, der aber alle zwei Sekunden ausgelöst wird, um die visuelle Anzeige zu aktualisieren. Der inject-Node ist mit dem function-Node verbunden, welcher für die Datenbankabfrage mit folgendem Befehl als Beispiel zuständig ist:

```
msg.topic = 'SELECT MAX(PalettenID)  
FROM `bottlefillinglayout`.`limonaden_produziert`;  
return msg;
```

msg.topic enthält die Abfrage für die Datenbank. Die Abfrage, also msg.topic, geht als nächstes zum mysql-Node und das Resultat der Abfrage wird als msg.payload an den widget-Node weitergegeben. Dieser letzter Node der Kette durchsucht msg.payload nach einem numerischen Wert und formatiert ihn entsprechend dem definiertem WertefORMAT.

Insgesamt sind fünf solche Flows konfiguriert: für volle Paletten, hergestellte Flaschen, defekte Flaschen, verbleibende Flaschen und verbleibende Flaschendeckel.

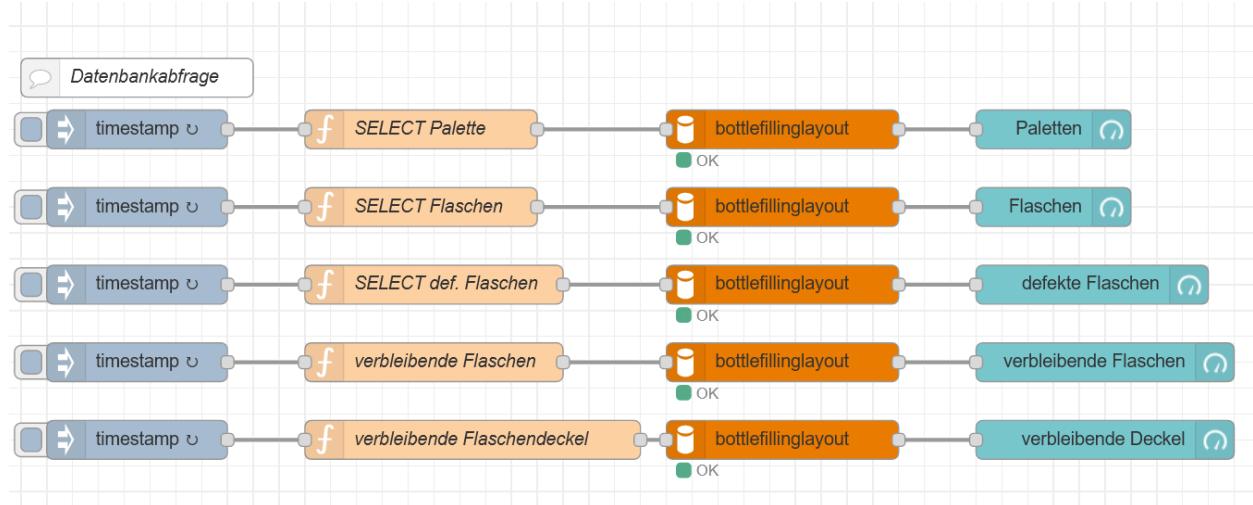


Abbildung 31: Flow zur Abfrage der Datenbank

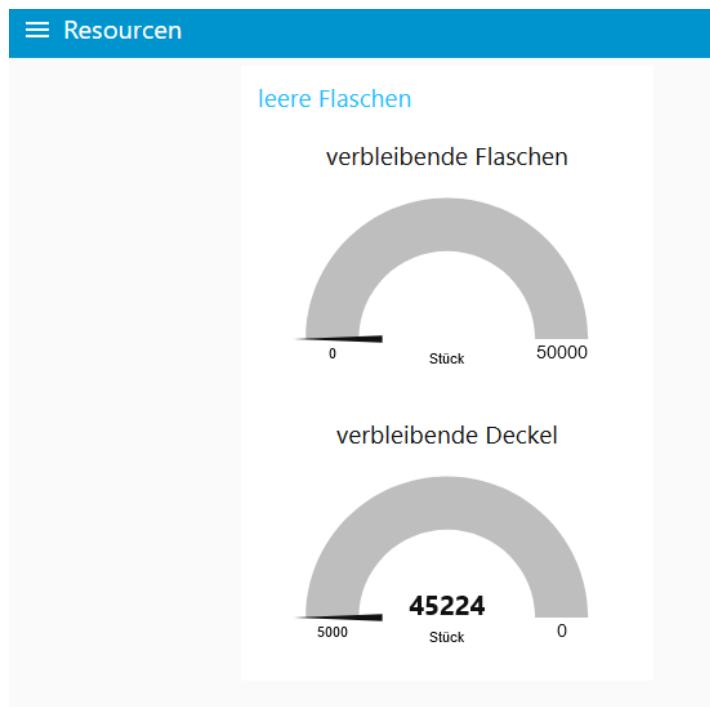


Abbildung 32: Ressourcenbestand im Dashboard

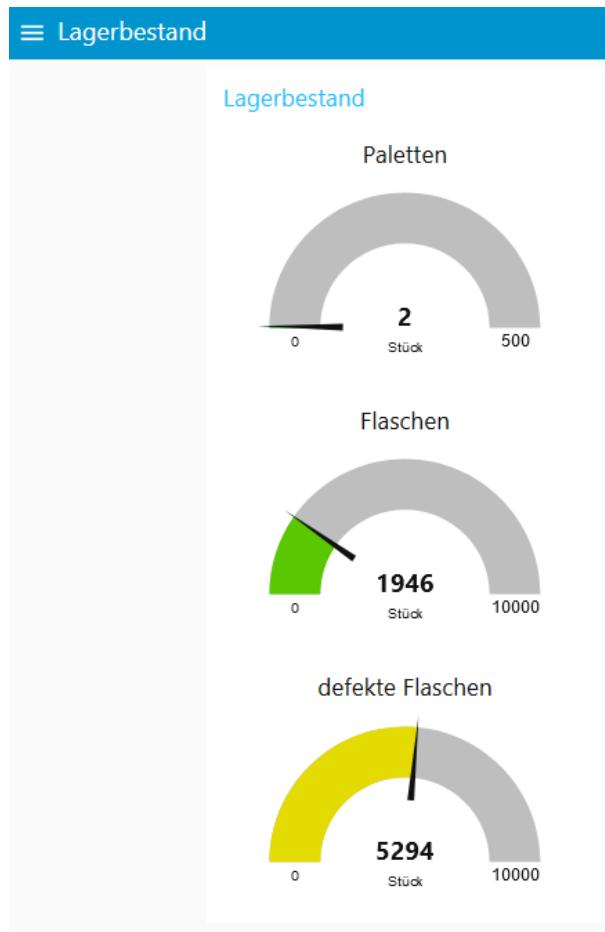


Abbildung 33: Echtzeitstatistik über die Limonadenproduktion im Dashboard

#### 4.6.4 Erstellung eines Dashboards für Bestellung und Bedienung

Damit auch Bestellungen entgegengenommen werden kann, ist im Dashboard ein Formular eingerichtet, in dem der Kunde die Möglichkeit hat, seinen Namen und die Palettenanzahl anzugeben. Dazu wurden folgenden Widgets konfiguriert:

Das Texteingabefeld, mit optionaler Beschriftung, kann auch den Passwort-, E-Mail- und Farbmodus unterstützen. Hier wird der Name des Bestellenden eingetragen

Einem Dropdown-Auswahl-Widget können mehrere Werte hinzugefügt werden. In dem Dropdownmenü „Getränk“ stehen die Werte Cola und Fanta zur Verfügung.

Beim Klicken auf den Button „Save“ wird eine Nachricht als `msg.payload` generiert und in das Payload-Feld eingetragen. Nach Möglichkeit kann für den Save-Button ein Fa-Icons eingestellt werden - die Farbe und die Hintergrundfarbe können ebenfalls eingestellt werden.

The screenshot shows a simple web form titled "Bestellung aufgeben". It contains two input fields: "Name" and "Palettenanzahl" (Quantity). The "Palettenanzahl" field includes a small dropdown arrow icon. Below the fields is a large blue button labeled "SAVE".

Abbildung 34: Bestellformular im Webbrowser

Alle eingetragenen Bestellungswerte im Webbrowser müssen zu einer Nachricht verbunden werden. Dies geschieht in dem join-Node, mit dem alle vier Widgets (Texteingabe, Dropdown-Feld, Zahleingabe, Save-Button) verbunden sind und der alle einge hende Sequenzen in eine Nachricht, sogenanntes `msg.payload` kombiniert. Der „Save-Button“ löste den Flow aus:

Die Bestellungsdaten werden im Flow weiter an den switch-Node weitergereicht. In diesem ist eine Regel definiert, die das Weiterreichen der zusammengefassten Nachricht solange verhindert, bis auf den „Save-Button“ geklickt wird. Die Bestellwerte werden weiter an den function-Node weitergereicht. Dieser erwartet eine Eigenschaft `msg.payload` und gibt ein `msg` Nachrichtenobjekt zurück. In dem function-Node steht folgender MySQL-Befehl:

```
msg.topic="INSERT INTO bottlefillinglay-
out.bestellungen (Name, Palettenanzahl) VALUES (?, ?)";
msg.payload=[msg.payload.name, msg.payload.code];
return msg;
```

Das obige SQL-Statement wird in den `msg.topic` verpackt und weiter an den mysql-Node weitergereicht. Nun wird der `INSERT`-Befehl aus dem `msg.topic` ausgeführt und die Daten werden in die Datenbank eingetragen. Nur nach einem erfolgreichen Schreibvorgang in die Datenbank, erscheint eine Bestellbestätigungsmeldung die nur mit „OK“ bestätigt werden kann. Schließlich werden alle Eingabefelder mit dem change-Node automatisch geleert und es kann neue Bestellung aufgegeben werden.

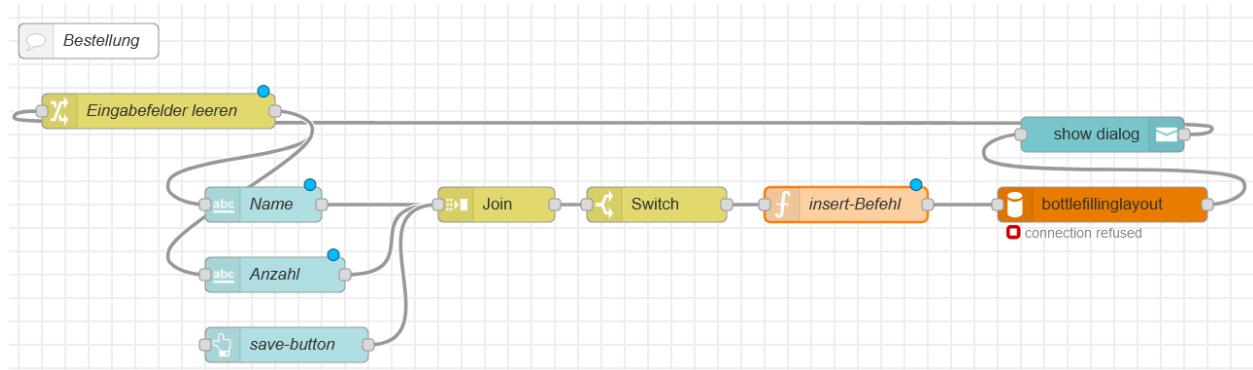


Abbildung 35: Flow für Bestellungen

### Aktor für die Flaschenproduktion

Die Konfiguration der Aktorenansteuerung, war der zeitaufwendigste Bestandteil der gesamten Dashboard-Einrichtung. Die Flows für die Akteure konnten jedoch schließlich mit Hilfe von vielen Forumbeitragsfragmenten erstellt werden.

Zu Abbildung 43:

Der Flow wird durch ein Klicken auf den Node-Button ausgelöst. Er hat im Payload den booleschen Wert false als Standardwert eingetragen. Am schwitcch-Node angelangt, wird der Payload von diesem entweder durch Ausgang 1 bei einem true-Wert weitergeleitet, oder durch Ausgang 2 bei einem false-Wert. Der Payload wird nun im nächsten change-Node auf den Umkehrwert gesetzt. Kommt beispielsweise ein true-Wert durch den Ausgang 1 des switch-Ports an dem change-Node an, so wird der Payload auf false gesetzt.

Zur nächsten switch-Node Ebene:

Node-RED bietet eine Möglichkeit, Informationen zu speichern, die von verschiedenen Knoten gemeinsam genutzt werden können, ohne die Nachrichten zu verwenden, die einen Fluss durchlaufen. Dies wird als 'Kontext' bezeichnet. Kontext erlaubt die gemeinsame Nutzung eines Zustands durch mehrere Knoten. Die Regel in der Abbildung 42, für den Knoten Change, speichert beispielsweise den Wert von msg.payload in flow context unter dem Schlüssel von „toggle-me“:

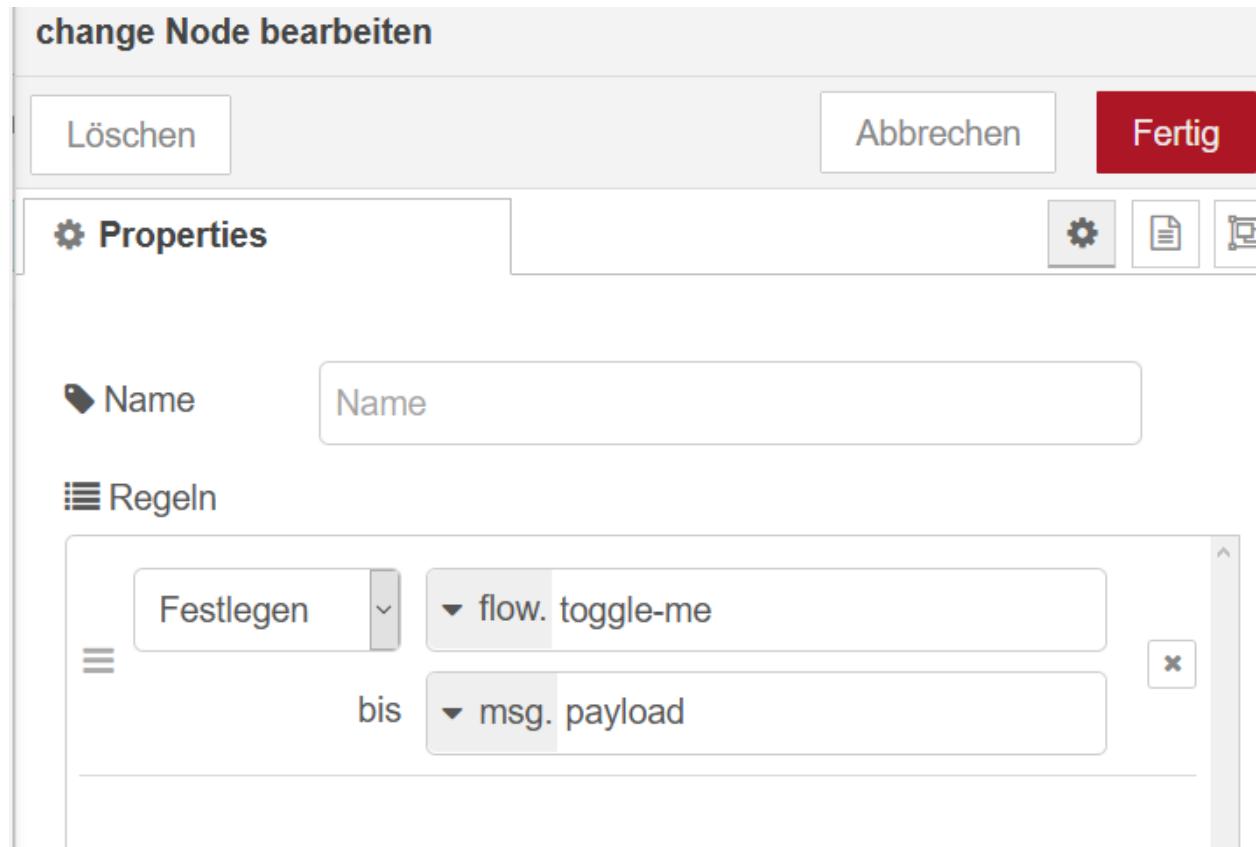


Abbildung 36: Konfiguration des change-Nodes

Durch das Speichern des booleschen Wertes im Kontext, ist er dann für die Rückkehr des HTTP-Flows verfügbar. Den true- oder false-Werten wird in dem OPC-UA-item-Node der Variablenpfad auf dem Server zugeordnet und schließlich in dem Letzen Node der Kette auf den Server geschrieben. Zugleich werden die booleschen Werte mit Hilfe des text-input-Nodes im Webbrowser ausgegeben. So ist nach jedem Klick im Browser „true“ oder „false“ sichtbar.

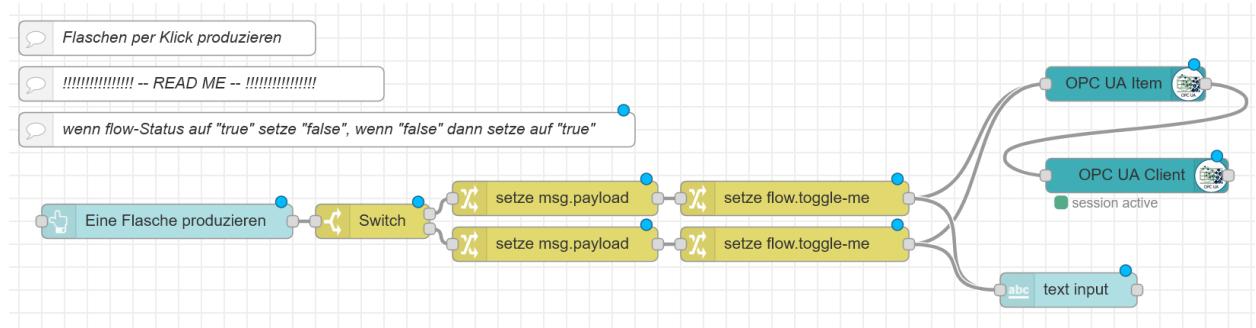


Abbildung 37: Flow für die manuelle Flaschenproduktion

Damit die Fabrik auf das manuelle Eingreifen in die Flaschenproduktion reagiert, muss dem „ReferenceFeeder“ (den Ursprung der Flaschen) mitgeteilt werden, nur bei einem Eingangssignal eine Flasche zu kreieren. Dazu muss in den Eigenschaften des Feeders, ein Hacken beim „CreateOnSignal“ – Wert (in Abbildung 38 gelb markiert) gesetzt werden. Ohne diesen, kreiert der Feeder automatisch unendlich viele Flaschen. Bei der Zurückstellung auf automatische Produktion, also beim Auschecken des Hackens, muss die Produktion neugestartet werden.

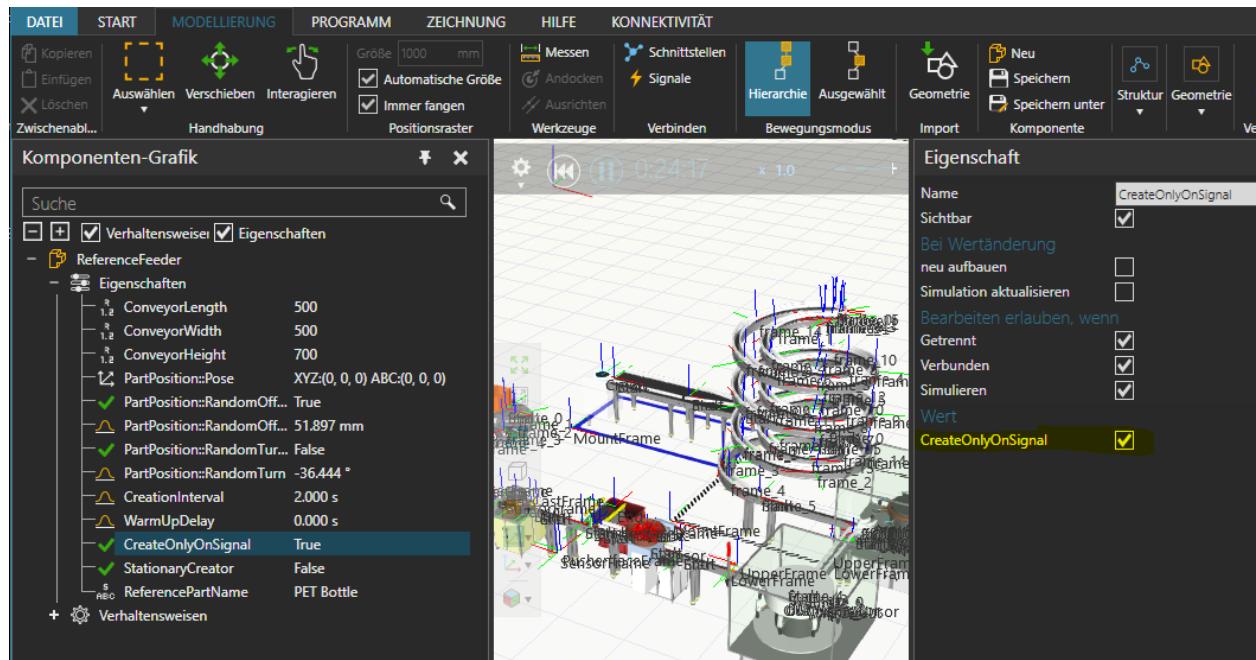


Abbildung 38: Notwendige Konfiguration für die manuelle Flaschenproduktion

### Aktor für die Regelung der Schraubverschlussmaschine

Der erste Node erlaubt es dem Benutzer im Webbrowser einen Wert zwischen 1 bis 5 zu definieren, da die Maschine bis zu 5 Flaschen gleichzeitig zuschrauben kann. Bei einem Testlauf führte der Wert > 5 zu einem Stillstand der Schraubverschlussmaschine. Die Flaschen wurden nicht mehr zugeschraubt und blieben unter der Maschine stehen. Als nächstes wird der Integer-Wert der Variablenpfad auf dem Server zugeordnet, bevor er im letzten Node eingetragen wird.

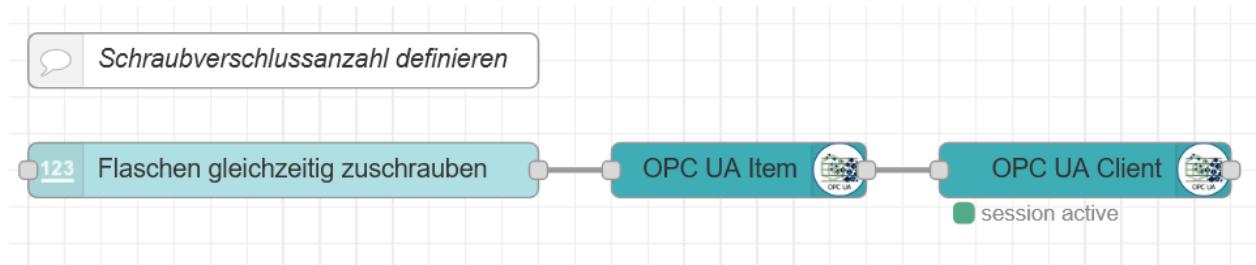


Abbildung 39: Flow für die Steuerung der Schraubverschlussmaschine

Vereinzelt kommt es zu Fehlern bei der Manipulation des Schreibverschlusswertes während einer laufenden Produktion. Entweder stoppt die Maschine den Schraubverschlussvorgang, was einen Flaschenstau zur Folge hat, oder sie lässt teilweise unverschraubte Flaschen durch. Wie im Punkt 4.3.3 bereits erwähnt, sind Flaschen mit äußersten Mängeln nicht als defekt deklariert, sondern werden von Programm selbst sporadisch ausgewählt und als „defekt“ markiert.

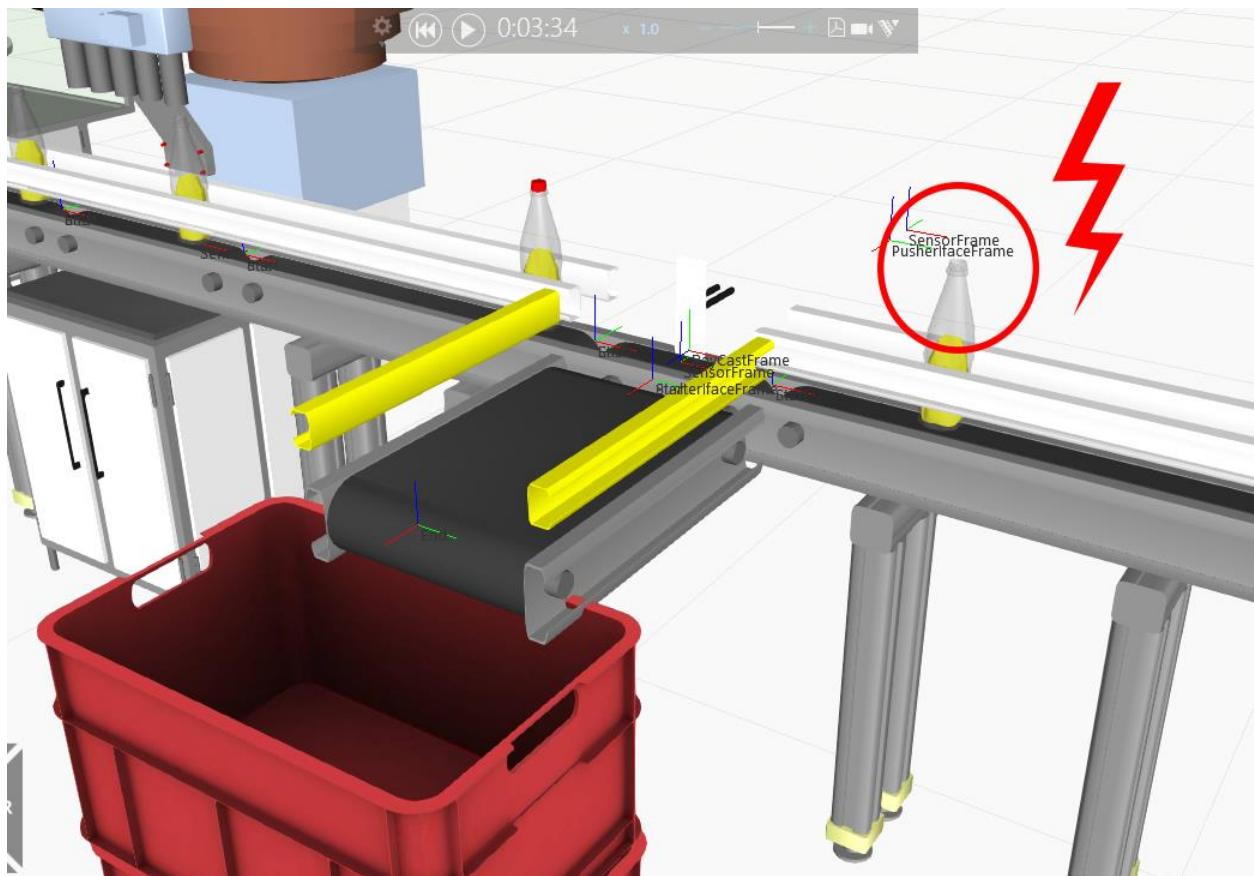


Abbildung 40: Unverschlossene Flasche nach Änderung des Wertes der Schraubverschlussmaschine

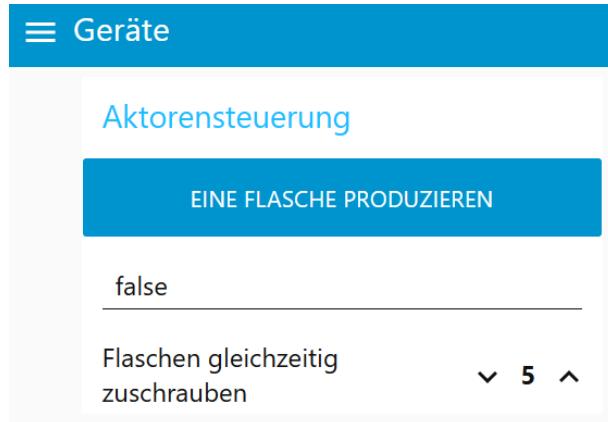


Abbildung 41: Bedienelement für die Aktoen

## 5 Bewertung des Projektprozesses

Im letzten Kapitel der Dokumentation wird ein Fazit aus dem Projekt gezogen und die Kognition erörtert. Anschließend wird das Potential des Projekts für mögliche Erweiterungen diskutiert.

### 5.1 Reflexion der Machbarkeitsanalyse

Dieses Kapitel beschreibt die Reflexion der zur Projektplanung erstellten Machbarkeitsanalyse. Aufgrund mangelnder Kenntnisse und der Unwissenheit über die Existenz bestimmter Anwendungssoftware, wurden einige Punkte in der Analyse nicht berücksichtigt. Dabei war die größte Herausforderung in Erfahrung zu bringen, ob und wie die Realisierung einer Datenbankanbindung mit dem OPC-UA-Server möglich ist. Durch spätere Gespräche mit dem Projektbetreuer Herrn Doktor Hofmann, wurde für das Erreichen des Teilzieles der Einsatz des OPC Routers vorgeschlagen. Bis zu jenem Zeitpunkt bestand der Glaube selbst, einen Wrapper für eine Anbindung implementieren zu müssen. Die Verwendung der Software OPC Router erleichterte die komplette Vernetzung vom OPC-UA-Server, der Datenbank bis hin zu den konfigurierten Flows von node-RED. Ebenfalls hatten wir uns nicht mit der Frage beschäftigt, wie mögliche Trigger, die der Datenbank Anweisungen zur Dateneinpfliegung übergeben sollen, implementiert werden können. Diese Schwierigkeit konnte in der Projektumsetzung allerdings ebenfalls mit dem OPC Router umgesetzt werden. Die anderen Punkte der Machbarkeitsanalyse haben sich in der Projektumsetzung bewahrheitet.

## 5.2 Reflexion des Projektstrukturplans

Dieser Abschnitt beschreibt die Reflexion des Projektstrukturplans, welcher ebenfalls wie die Machbarkeitsanalyse aufgrund fehlender Erfahrung und Kenntnissen zu Beginn große Lücken aufwies. So bestand der aufgestellte Projektstrukturplan lediglich aus den drei Oberpunkten: Vorbereitung, Implementierung und Konnektivität. Diese waren jeweils mit den gröbsten Unterpunkten versehen. In einem Gespräch mit Herrn Köppel wurde auf die Mängel des Plans hingewiesen und daher wurde dieser ein weiteres Mal überarbeitet. Allerdings mit Hinblick auf die neu gewählten Oberpunkte Planung, Vorbereitung, Implementierung, Konnektivität und Abnahme. Nach etlichen Stunden Recherche sind nun die Arbeitspakete unter den Oberpunkten detaillreicher definiert. Allerdings fand das Gespräch mit dem Projektbetreuer Herrn Doktor Hofmann erst einige Wochen später statt, weshalb die Konfiguration des OPC Router und dessen Verbindungen nicht im Projektstrukturplan auftaucht.

## 5.3 Reflexion der Zeitplanung

In der Reflexion der Zeitplanung ist aufgezeigt, wie sich dieser im Laufe der Projektplanung bis hin zur Projektdurchführung variiert hat. Diese Änderungen passierten aufgrund neuer Erkenntnisse und aus organisatorischen Gründen. Da in der Vergangenheit noch keine Teamprojekte von dem Projektteam durchgeführt wurden, fehlte es an Knowhow im Organisationsvermögen. Daher wurde zum Beginn des Projektes jeweils an denselben Arbeitspaketen gearbeitet, was zeitlich und finanziell wenig effizient war. Ebenso war der erste Zeitplan nicht auf Stunden-, sondern Tagesbasis aufgestellt worden. Nach der ersten Überarbeitung des Plans musste dieser allerdings im späteren Verlauf noch einmal umstrukturiert werden. Dies geschah ebenfalls durch das Erfahren der Existenz der Software OPC Router. Dadurch mussten neue Arbeitspakete mit in die Zeitplanung einfließen. Die Aufgaben wurden gerecht auf die Projektleiter aufgeteilt und selbstständig ausgearbeitet. Diese Umstrukturierung des Plans steigerte die Effizienz enorm und ermöglichte neue Teilaufgaben mit in Betracht zu ziehen.

Bei der Planung kamen die schulischen Erfahrungen mit node-RED und MySQL zugute. Anhand der Erfahrungswerte konnte eine annähernde Stundenanzahl für den jeweiligen Projektabschnitt angegeben werden.

## 5.4 Reflexion der Projektziele

Ein direkter Vergleich der tatsächlichen Projektergebnissen und den Projektzielen aus der Projektvereinbarung legt dar, dass alle Erwartungen erfüllt werden konnten. In der

Visual Components Software sind vorgefertigte Sensoren benutzt worden, die die Anzahl der Flaschen erfassen, oder mit Herstellungsdatum versehen. Eine Verbindung aus Visual Components zum OPC-UA-Server ist hergestellt und die Daten der Sensoren werden an den Server übermittelt. Eine markante Vereinfachung des Datenaustausches zwischen dem OPC-UA-Server und MySQL konnte mit dem, von Dr. Hofmann empfohlenen OPC UA-Router erreicht werden. Dieser stellt nach wenigen Konfigurationsschritten eine Verbindung zwischen den beiden Komponenten her und alle Werte werden korrekt aus dem OPC-UA-Server in die SQL-Datenbank geschrieben.

Ebenfalls erfolgreich werden die, in node-RED aus der Datenbank abgefragten Werten, im Webbrowser abstrahiert. Zusätzlich werden die Sensorwerte direkt aus Visual Components visualisiert dargestellt. Beim Letzterem wurde die Verbindung und Kommunikation zwischen dem OPC-UA-Server und node-RED durch vorgefertigte OPCUA-Nodes merklich vereinfacht.

Auch die Funktionstüchtigkeit der Aktoren ist gewährleistet. Über den Browser lassen sich einzelne Flaschen per Mausklick produzieren bzw. befüllen und die Schraubverschlussmaschine bedienen. Dabei kann der Schraubverschlussmaschine die Anzahl der zu verschraubenden Flaschen gleichzeitig mitgeteilt werden.

## 6 Erkenntnisse

Schon während der Projektdurchführung, konnte das Potential der Vereinheitlichung in der Steuerungs- und Automatisierungswelt mittels OPC UA – Standards erahnt werden. Dieses Projekt beschäftigt sich zwar nur mit der Darstellung der Informationen und mit dem Zugriff auf Parameter einer Flaschenfabrik, doch bei der Recherchearbeit wurde auch die Nutzung der Security-Technologien, Safety- und Netzwerkmanagement erwähnt. OPC UA ist einer der weiteren und wichtigen Schritte für die adaptiven, benutzerfreundlichen und vor allem effiziente Produktion.

Da das Projekt von zwei Personen durchgeführt wurde, förderte anfangs die mangelnde Teammanagementerfahrung beider Teilnehmer die Missorganisation und unvorteilhafte Aufgabenverteilung, wodurch bis zur ersten Meilensteinsitzung teilweise uneffektiv gearbeitet wurde. So beschäftigten sich beispielsweise beide Teilnehmer mit den gleichen Aufgaben, was keinen Zeitvorteil mit sich brachte. Im zweiten Projektabschnitt stellte sich jedoch heraus, dass die verschiedenen Wissensstände, Herangehensweisen und Erfahrung aus den ersten Teilaufgaben einen positiven Effekt auf eine aufgetauchte Problemlösung hatten.

Die Kommunikation und Datenaustausch wurden anfänglich per Email kommuniziert. Relativ zeitnah kam es jedoch zu der Erkenntnis, dass dieser weg nur zum Dokumentenverlust und Kommunikationsproblemen bzw. Missverständnissen führt. Nach der negativen Erfahrung erfolgte später der Datenaustausch mittels GitHub und die Kommunikation wurde auf WhatsApp verlagert.

Bei der Projektdurchführung stellte während der Prüfungsphasen und dadurch entstandenen Leistungsdruck, sowohl die Unterstützung als auch die gegenseitige Motivation eine Herausforderung dar. Oft war ein besonderes Fingerspitzengefühl bei den Projekt-partnern gefragt.

Einer der größten Errungenschaften beider Teilnehmer ist die strukturierte Planung und Durchführung eines Projektes und die Kommunikation miteinander sowohl auf der technischen als auch auf der menschlichen Ebene.

## 6.1 Ausblick

Durch die zahlreichen Simulationsmöglichkeiten, bietet Visual Components eine breite Konfigurationspalette. Auf das Projekt angewandt, ist es denkbar, dass man eine Art Steuerdashboard für die Steuerung der einzelnen Stationen erstellt und damit ein manuelles Eingreifen auf die einzelnen Produktionsstationen und zugleich in die Prozessabläufe ermöglicht. Durch einzelne Bedienungselemente könnte die Produktionsgeschwindigkeit erhöht und die Maschinenauslastung angepasst werden. Das manuelle Eingreifen ist zwar nicht praxisfreundlich, da die Idee von vollautomatisierten Abläufen verfehlt wird, jedoch eine gute Übung um ein wenig die OPCUA-Welt kennen zulernen.

Ein sinnvolleres Scenario dagegen, könnte die Konfiguration komplexer Online-Bestellungen bieten. So könnte man im weiteren Projekt eine Limonaden-Web-Shop konfigurieren, über den es möglich wäre, neben der Flaschenanzahl, verschiedenen Arten von Getränken in verschiedenen Flaschenformen zu bestellen. Zu den Aufträgen könnte der Kunde zusätzlich einen spätesten Liefertermin definieren. Die vollautomatisierte Fabrik priorisiert den Zeitplan entsprechend den Aufträgen und steuert in diesem Fall die Auslastungskapazitäten der Maschinen selbst.

## Glossar

GitHub	GitHub ist eine Online-Software-Entwicklungsplattform, auf der Entwickler ihre Arbeitsergebnisse zusammenführen und die verwalten können.
FA-icon	Das Akronym FA leitet sich von der Webtypografie Font Awesome ab. Dabei werden kleine Grafiken unabhängig von den installierten Fonts des Besuchers auf einer Website in seinem Browser angezeigt.
Nodes	Knoten sog. Nodes, werden bei der Bereitstellung eines Datenflusses erstellt. Sie können Nachrichten senden und empfangen, während der Datenfluss läuft und sie werden gelöscht, wenn der nächste Datenfluss bereitgestellt wird.
parsen	maschinenlesbare Daten analysieren, segmentieren und codieren
Payload	Ein node-RED-Flow (Datenfluss) funktioniert nur durch die Weitergabe von Nachrichten zwischen den Knoten. Bei den Nachrichten handelt es sich um einfache JavaScript-Objekte, die eine beliebige Menge von Eigenschaften haben können. In der Regel haben Nachrichten eine Payload-Eigenschaft - dies ist die Standardeigenschaft, mit der die meisten Knoten arbeiten.

## Literaturverzeichnis

1. „Was ist Industrie 4.0?“, [Online], Available  
<https://www.plattform-i40.de/PI40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html> [Zugriff am 27. Februar 2020]
2. “Introduction to Vagrant”, [Online], Available  
<https://www.vagrantup.com/intro/index.html> [Zugriff am 03. Februar 2020]
3. “OPC Foundation”, [Online], Available  
<https://opcfoundation.org/members> [Zugriff am 28. Dezember 2019]
4. “C++ SDK Demo Server”, [Online], Available  
<https://documentation.unified-automation.com/uasdkcpp/1.5.3/html/L2DemoServer.html> [Zugriff am 03. Februar 2020]
5. “UaExpert – ein vollausgestatteter OPC UA Client“, [Online], Available  
<https://www.unified-automation.com/de/produkte/entwicklerwerkzeuge/uaexpert.html> [Zugriff am 03. Februar 2020]
6. “MySQL Workbench 6.3.6”, [Online], Available  
<https://www.heise.de/download/product/mysql-workbench-14226> [Zugriff am 03. Februar 2020]
7. “phpMyAdmin – Brining MySQL to the web”, [Online], Available  
<https://www.phpmyadmin.net/> [Zugriff am 03. Februar 2020]
8. “SharePoint – Ihr mobiles intelligentes Intranet”, [Online], Available  
<https://products.office.com/de-de/sharepoint/collaboration> [Zugriff am 04. Februar 2020]
9. “OPC Server”, [Online], Available  
<https://www.matrikonopc.de/opc-server/> [Zugriff am 04. Februar 2020]
10. “About Node.js”, [Online], Available  
<https://nodejs.org/en/about/> [Zugriff am 05. Februar 2020]
11. “Git Bash – Tools & Features”, [Online], Available  
<https://gitforwindows.org/> [05. Februar 2020]
12. "OPC Router – die Industrie 4.0-Software", [Online], Available:  
<https://www.inray.de/produkte/opc-router/>, [Zugriff am 15. Februar 2020]
13. "Design the factories of the future", [Online], Available:  
<https://www.visualcomponents.com>, [Zugriff am 15. Februar 2020]
14. "Node-RED", [Online], Available:  
<https://nodered.org/>, [Zugriff am 2. Februar 2020]

15. "Conveyors", [Online], Available:  
[http://download.visualcomponents.net/elib/3.1/EquipmentLibrary\\_HTML/1%20Simulation%20Models/Components/By%20Type/Conveyors/index.htm](http://download.visualcomponents.net/elib/3.1/EquipmentLibrary_HTML/1%20Simulation%20Models/Components/By%20Type/Conveyors/index.htm),  
[Zugriff am 28 Dezember 2019]
16. "Lessons", [Online], Available:  
<https://academy.visualcomponents.com/lessons/>, [Zugriff am 15. Dezember 2019]
17. "Create New Tasks in Works Library | Visual Components 4.1", [Online], Available:  
[https://www.youtube.com/watch?v=uRL7m\\_cInCA](https://www.youtube.com/watch?v=uRL7m_cInCA), [Zugriff am 17. Dezember 2019]
18. "Welcome to the Node-RED Forum!", [Online], Available:  
<https://discourse.nodered.org/>, [Zugriff am 02. Februar 2019]
19. "Steve Cope", [Online], Available:  
<https://www.youtube.com/channel/UCt12dTP8xvHM02Gc8HdGOcQ>,  
[Zugriff am 27. Januar 2019]
20. "Open Source SCADA: Node-RED, OPC UA & MySQL on Raspberry Pi",  
[Online], Available:  
<https://www.youtube.com/watch?v=LaUmhhMdoyY>, [Zugriff am 25. Januar 2019]

## Abbildungsverzeichnis

Abbildung 1: OPC-UA Kommunikationsstack .....	15
Abbildung 2: Systemkonfiguration der virtuellen Arbeitsumgebung .....	21
Abbildung 3: Festlegung der eingehenden Regel.....	21
Abbildung 4: Deklaration des Servers und Erstellung einer Instanz .....	25
Abbildung 5: Erstellung des AddressSpace samt Objekten .....	26
Abbildung 6: Definieren der Variablen für das Objekt sensor1.....	27
Abbildung 7: Definieren der Variablen für das Objekt sensor2.....	28
Abbildung 8: Definieren der Variablen für das Objekt sensor3.....	29
Abbildung 9: Asynchrone Startmethode des Servers .....	30
Abbildung 10: Quellcode des Addons messageHelper .....	37
Abbildung 11: Pythonskript der Init-Datei.....	38
Abbildung 12: Quellcode des Volumensensors .....	40
Abbildung 13: Hinzugefügte und konfigurierte Sensoren .....	41
Abbildung 14: Inkorrektar Aussortievorgang. Defekte Flaschen werden nicht aussortiert.....	42
Abbildung 15: Eigenschaften einer Flasche beim Labelvorgang.....	43
Abbildung 16: Eigenschaften des Palettenzählers .....	44
Abbildung 17: Flaschenverschlussmaschine .....	45
Abbildung 18: ReferenceFeeder beim Erstellen leerer Flaschen .....	46
Abbildung 19: Server im Reiter "Konnektivität" .....	47
Abbildung 20: Bilden der Variablenpaare .....	47
Abbildung 21: Verwendete Variablen für Sensoren und Aktoren .....	48
Abbildung 22: Verwendete Variablen für Sensoren und Aktoren in UaExpert.....	48
Abbildung 23: Auffüllen des Vorratslagers .....	50
Abbildung 24: Vergabe der Unique-Schlüssel .....	51

Abbildung 25: Trigger für die Tabelle Vorratslager .....	51
Abbildung 26: Anlegen des Triggers für Bestellungen .....	52
Abbildung 27: ER-Modell der Datenbank.....	52
Abbildung 28: Flow für die Wertauslesung der Variablen des OPC-UA-Servers.....	62
Abbildung 29: Nodes für den Live-View.....	63
Abbildung 30: Echtzeitstatistiken im Dashboard .....	63
Abbildung 31: Flow zur Abfrage der Datenbank .....	65
Abbildung 32: Ressourcenbestand im Dashboard.....	65
Abbildung 33: Echtzeitstatistik über die Limonadenproduktion im Dashboard .....	66
Abbildung 34: Bestellformular im Webbrowser .....	67
Abbildung 35: Flow für Bestellungen .....	68
Abbildung 36: Konfiguration des change-Nodes.....	69
Abbildung 37: Flow für die manuelle Flaschenproduktion.....	69
Abbildung 38: Notwendige Konfiguration für die manuelle Flaschenproduktion .....	70
Abbildung 39: Flow für die Steuerung der Schraubverschlussmaschine.....	70
Abbildung 40: Unverschlossene Flasche nach Änderung des Wertes der Schraubverschlussmaschine .....	71
Abbildung 41: Bedienelement für die Aktoren.....	72

## Quellcodeverzeichnis

Quellcode 1: Deklaration des Servers und Erstellung einer Instanz.....	22
Quellcode 2: Erstellung des AddressSpace samt Objekten.....	23
Quellcode 3: Definieren der Variablen für das Objekt sensor1.....	24
Quellcode 4: Definieren der Variablen für das Objekt sensor2.....	25
Quellcode 5: Definieren der Variablen für das Objekt sensor3.....	26
Quellcode 6: Asynchrone Startmethode des Servers.....	27
Quellcode 8: Quellcode des Addons messageHelper.....	34
Quellcode 9: Pythonskript der Init-Datei.....	35

Quellcode 10: Quellcode des Volumensensors.....	36
Quellcode 11: Anlegen der Datenbank samt Tabellen.....	47
Quellcode 12: Auffüllen des Vorratslagers.....	47
Quellcode 13: Vergabe der Unique-Schlüssel.....	47
Quellcode 14: Trigger für die Tabelle Vorratslager.....	48
Quellcode 15: Anlegen des Triggers für Bestellungen.....	4

## **Tabellenverzeichnis**

Tabelle 1: Verwendete Variablen.....	32
Tabelle 2: Konfiguration der Aktoren.....	42

# Anlagenverzeichnis

## Anhang A (Projektmanagement)

### A.1 Machbarkeitsanalyse

Rudolf Diesel Fachschule

16.09.2019

#### Dokumentation der Machbarkeitstests

**Projekttitel:** Datenerfassung und -abstraktion von Industrieanlagen mittels OPC UA  
**Tester:** Michael Mölle, Ondrej Hruby  
**Datum:** 16.09.19

1. Die Software „Visual Components V1.7.0“ kann mit einem Fujitsu Lifebook mit einem Intel i5-Prozessor, 16 GB RAM und einer Intel HD Graphics 4400 betrieben werden.
2. In der Software „Visual Components V1.7.0“ ist es möglich, einen OPC-UA-fähigen Sensor an das Fließband zu installieren und mit einem Server zu verbinden (vgl. <http://academy.visualcomponents.com/lessons/connect-a-remote-opc-ua-server/>)
3. Der Unified Automation C++ Server kann auf einer virtuellen Windows 10 Maschine mit einem Intel i5-Prozessor und 4 GB RAM betrieben werden.
4. Die Software „UA Expert“ kann auf einer virtuellen Windows 10 Maschine mit einem Intel i5-Prozessor und 4 GB RAM betrieben werden.
5. Die Client-Server-Verbindung zwischen dem Sensor und dem OPC-UA-Server kann hergestellt werden (vgl. <http://academy.visualcomponents.com/lessons/connect-a-remote-opc-ua-server/>)
6. OPC-UA Modul-Pakete für node-red unter Windows sind online zum freien Download verfügbar (<https://node-opcua.github.io/>, <https://flows.nodered.org/node/node-red-contrib-opcua/>)
7. Die Abstraktion der erfassten Sensordaten im UA-Server kann mit node-red realisiert werden (vgl. <https://flows.nodered.org/node/node-red-contrib-iiot-opcua>, <https://flows.nodered.org/node/node-red-dashboard>)

Michael Mölle, Ondrej Hruby

## A.2 Projektvereinbarung

Rudolf Diesel Fachschule

16.09.2019

### Projektvereinbarung

<b>Projekttitel</b>	Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC-UA
<b>Projektteam</b>	Ondrej Hruby Michael Mölle
<b>Projektleiter</b>	Dr. Markus Hofmann
<b>Projektauftraggeber</b>	Dr. Markus Hofmann
<b>Projektkunden</b>	-
<b>Projektdauer</b>	<b>Geplanter Beginn:</b> 13.09.19 <b>Geplantes Ende:</b> 03.02.20
<b>Projektgesamtziel</b>	In einer fiktiven Limonaden-Abfüll-Fabrik in der Software namens „Visual Components“ sind diverse Akteure und Sensoren vorhanden. Ziel ist es, die Akteure anzusteuern und Sensordaten mittels OPC-UA zu gewinnen. Die erfassten Daten werden durch eine Anbindung in eine Datenbank eingepflegt und die bidirektionale Steuerung der Füllanlage über eine Webanbindung mittels node-red realisiert.
<b>Zu verarbeitende Daten</b>	Sensordaten
<b>Eingesetzte Techniken</b>	Für den Betrieb der Software „Visual Components“ soll ein Fujitsu-Life-Book mit einem Intel i5 Prozessor und 16 GB RAM verwendet werden. Beim OPC-UA-Server handelt es sich um einen lokalen OPC UA C++ Demo Server V1.7.0, der auf einer virtuellen Windows 10 Maschine betrieben werden soll. Die Abstraktion der Daten soll mittels node-red auf einer weiteren virtuellen Windows 10 Maschine realisiert werden.
<b>Projektziele- und -ereignisse</b>	<b>Teilziel:</b> Beschaffung der Software <b>Ergebnis:</b>

Michael Mölle, Ondrej Hruby

Rudolf Diesel Fachschule

16.09.2019

	<p><b>Teilziel:</b> Konfiguration der virtuellen Maschine  <b>Ergebnis:</b></p> <p><b>Teilziel:</b> Installation und Konfiguration des Fließbandsensors  <b>Ergebnis:</b></p> <p><b>Teilziel:</b> Einrichtung des OPC-UA-Servers  <b>Ergebnis:</b></p> <p><b>Teilziel:</b> Einrichtung von Client-Server-Verbindungen  <b>Ergebnis:</b></p> <p>Teilziel: Einrichtung einer Datenbankanbindung  <b>Ergebnis:</b></p> <p><b>Teilziel:</b> Datenauswertung mittels node-red  <b>Ergebnis:</b></p> <p>Teilziel Einrichtung einer Webanbindung zur bidirektionalen Steuerung  <b>Ergebnis:</b></p>
<b>Besondere Anforderungen</b>	Der Fließbandsensor muss die korrekte Füllmenge der Flaschen erkennen. Ist dies nicht der Fall, darf die Flasche nicht gewertet werden.
<b>Nicht-Ziele / Nicht-Inhalte</b>	

Michael Mölle, Ondrej Hruby

Rudolf Diesel Fachschule

16.09.2019

<b>Meilensteine</b>	Meilensteine:                    Datum: 1. Sitzung:                      25.10.19 In der Meilensteinsitzung sollen die Teilziele „Beschaffung der Software“, „Konfiguration der virtuellen Maschine“ und „Installation und Konfiguration des Fließbandsensors“ erreicht sein.  2. Sitzung:                      13.12.19 In der Meilensteinsitzung sollen die Teilziele „Einrichtung des OPC-UA-Servers“, „Einrichtung von Client-Server-Verbindungen“ und „Einrichtung einer Datenbankverbindung“ erreicht sein.  3. Sitzung                        21.02.19 In der Meilensteinsitzung soll das Ziel „Datenauswertung mittels node-red“ und „Einrichtung einer Webanbindung zur bidirektionalen Steuerung“.
<b>Randbedingungen und -projektkontext</b>	Software und dazugehörige Lizenzen müssen vorliegen
<b>Projektklassifizierung</b>	<b>Komplexität</b> hoch <b>Neuartigkeitsgrad</b> mittel <b>Projektumfang</b> hoch <b>Projektrisiko</b> mittel
<b>Projektressourcen</b>	<b>Ressourcen</b> <b>Menge</b> Notebook                        2 Visual Components V1.7.0     1 UA Expert                      1 UA C++ Demo-Server          1 UA C++ Demo-Client          1 Node-red                        1 OPC Router                     1

Michael Mölle, Ondrej Hruby

Rudolf Diesel Fachschule

16.09.2019

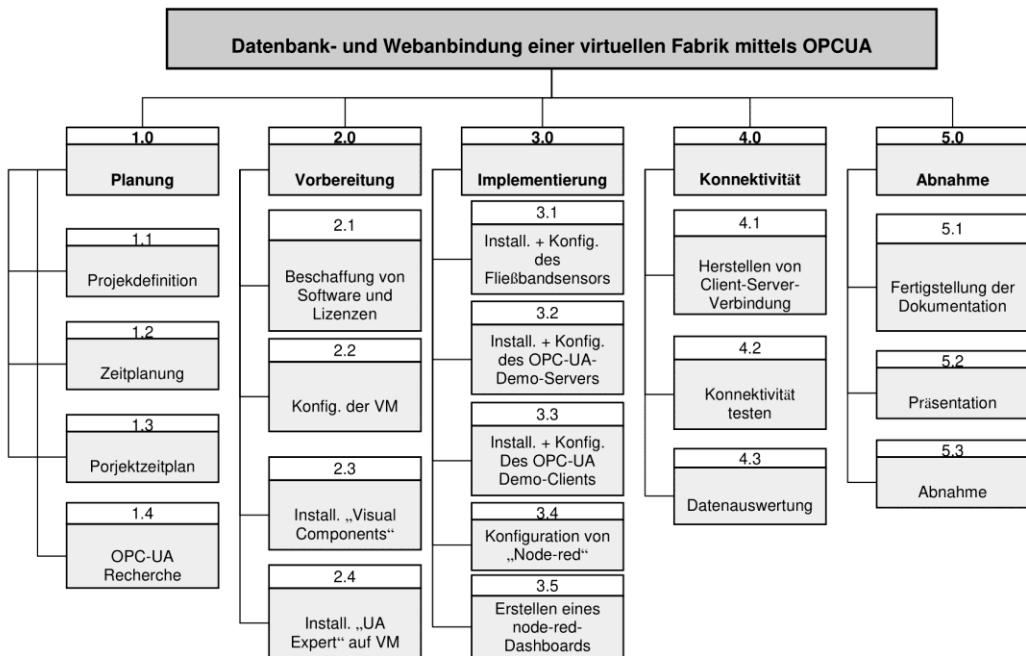
<b>Projektbudget</b>	-
<b>Projektentscheidung</b>	<b>Freigabe:</b> 13.09.19 <b>Freigabe am:</b>
<b>Anlagen</b>	- Zeitplanung - Machbarkeitsanalyse

Michael Mölle, Ondrej Hruby

## A.3 Projektstrukturplan

### Projektstrukturplan

Michael Mölle, Ondrej Hruby



## A.4 Zeitplanung

Beschreibung	Verantwortlicher	Start	Ende	Mölle / SOLL (in Stunden)	Mölle / IST (in Stunden)	Differenz	Hrubi / SOLL (in Stunden)	Hrubi / IST (in Stunden)	Differenz	Status	Kommentare
13.9.19                    25.10.19											
Recherche zu OPC UA	MöllerHrubi	13.9.19	17.9.19	10	10		10	10		Abgeschlossen	
Beschaffung von Software und Lizzenzen	MöllerHrubi	13.9.19	10.10.19	0,5	0,5		0,5	0,5		Abgeschlossen	Vor vorerst bis zum 30.09.2019 geplant. Bis heute (30.09.2019) weder Zugang zur Software vor Ort noch Lizenz erhalten
Recherche zur Vagrant und Virtual Box	MöllerHrubi	22.9.19	23.9.19	3	2,5		3	2,5		Abgeschlossen	
Konfiguration der virtuellen Maschine	MöllerHrubi	23.9.19	3.10.19	3	4,5		3	4		Abgeschlossen	
Installation der Software „Visual Components 17.0“	MöllerHrubi	10.10.19	11.10.19	1	0,5		1	0,5		Abgeschlossen	
Installation der Software „UA Expert“ (virtuelle Maschine)	MöllerHrubi	3.10.19	3.10.19	1	2		1	1		Abgeschlossen	
Recherche zu Visual Components und geeigneten Sensoren	MöllerHrubi	3.10.19	12.10.19	10	10		10	10		Abgeschlossen	
Installation/Konfiguration des Fließbandsensors	Mölle	12.10.19	24.10.19	15	12,75		0	7		Abgeschlossen	Grundkonfiguration (Fähigkeit gefüllte Flaschen zu erkennen und Anzahl festzuhalten) abgeschlossen. Es folgt weitere Recherche zur eventuellen Erweiterung/Spezifikation der Konfiguration.
Dokumentation der Teilziele	MöllerHrubi	13.9.19	17.9.19	10	7	Differenz:	10	2		Abgeschlossen	
				53,5	49,75	3,75	39,5	37,5	1		
25.10.19                    13.12.19											
Recherche zu OPC UA - Server	Mölle	25.10.19	27.10.19	20	12		0	0		Abgeschlossen	
Installation/Konfiguration des OPC-UA-Server (C++)	Mölle	27.10.19	11.11.19	50	25		0	0		Abgeschlossen	
Recherche zu OPC UA - Client	Hrubi	25.10.19	27.10.19	0	0		15	5		Abgeschlossen	
Installation/Konfiguration der OPC-UA-Clients, Akten und Sensoren	Hrubi	27.10.19	11.11.19	0	0		50	48,45		Abgeschlossen	
Recherche für Datenbank und Datenbankanbindung	Mölle	11.11.19	12.11.19	10	12		0	0		Abgeschlossen	
Anlegen einer Datenbank / Konfiguration der Tabellen	Mölle	12.11.19	15.11.19	10	10		0	0		Abgeschlossen	
Recherche zu OPC Router	Hrubi/Mölle	11.11.19	13.11.19	10	8		5	3		Abgeschlossen	
Installation / Konfiguration eines OPC Routers	Hrubi/Mölle	13.11.19	14.11.19	3	3		3	2		Abgeschlossen	
Recherche Verbindungsraubau zwischen OPC UA Server und Datenbank	Mölle	14.11.19	15.11.19	20	5		0	0		Abgeschlossen	
Datenbankanbindung	Mölle	16.11.19	1.12.19	40	40		0	0		Abgeschlossen	
Dokumentation der Teilziele	MöllerHrubi	1.12.19	13.12.19	10	20	Differenz:	10	17		Abgeschlossen	
				153	135	18	83	76,45	6,55		
13.12.19                    21.2.20											
Recherche zu Node-Red	Hrubi	13.12.19	16.12.19	0	0		10	4,5		Abgeschlossen	
Konfiguration von node-red	Hrubi	15.12.19	16.12.19	0	0		2	4		Abgeschlossen	
Erstellen eines node-red-Dashboards	Hrubi	16.12.19	25.12.19	0	0		40	47,5		Abgeschlossen	
Recherche zu Vebanbindung	Hrubi	25.12.19	27.12.19	0	0		10	4		Abgeschlossen	
Vebanbindung	Hrubi	27.12.19	31.12.20	0	0		30	0		Abgeschlossen	
Dokumentation der Teilziele	MöllerHrubi	1.2.20	10.2.20	10	20		15	22		Abgeschlossen	
Fertigstellung Dokumentation	MöllerHrubi	11.2.20	21.2.20	20	40	Differenz:	20	36	-1	Abgeschlossen	
				30	60	-30	117	118	-1		

## A.5 Meilensteinprotokoll Nr. 1 vom 25.10.2019

03.11.2019

### Protokoll

Erste Meilensteinsitzung zum Projekt „Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC UA“

25.10.2019, 10:30 Uhr bis 11:45 Uhr, Rudolf-Diesel-Fachschule, Nürnberg

#### Anwesend:

Michael Mölle	(Sitzungsleiter)
Ondrej Hruby	(Sitzungsleiter)
Markus Hofmann	(Projektleiter)
Konstantin Fahner	(Lehrer, Rudolf-Diesel-Fachschule)
Sandra Kroher	(Lehrer, Rudolf-Diesel-Fachschule)

#### Abwesend:

Johannes Köppel	(entschuldigt)
Rudolf Schneider	(entschuldigt)

#### Verteiler:

Teilnehmer

#### TOP 1: Vorstellung des Projektthemas

Informationen durch Sitzungsleiter Ondrej Hruby und ergänzend durch Michael Mölle:

- Erklären und Vorstellen der Idee und des Plans für die Projektarbeit
- Vorstellung der verwendeten Software (Visual Components, node-red, UAExpert, vagrant, Oracle VirtualBox, OPC UA C++ Demo-Server)
- Instruktion, welche Bedeutung der OPC UA Standard für das Projekt hat
- Klärung von Verständnisproblemen hinsichtlich der anzusteuern den Sensoren und Aktoren

03.11.2019

## TOP 2: Vorstellung der erreichten Teilziele

Teilziel	Bemerkung
Beschaffung der Software	<ul style="list-style-type: none"> <li>Die Einzellizenzen für die Software „Visual Components“ erhielten wir erst am 03.10.2019, statt wie geplant am 30.09.2019</li> <li>Vagrant, VirtualBox, node-red und UAExpert konnten ohne Probleme besorgt und installiert werden</li> <li>Die geladenen Teilnehmer akzeptieren die Verspätung</li> </ul>
Konfiguration der virtuellen Maschine	<ul style="list-style-type: none"> <li>Die virtuelle Maschine (Windows 10 64 Bit) konnte erfolgreich installiert und geeignet für den OPC UA C++ Demo-Server konfiguriert werden</li> <li>Die geladenen Teilnehmer verstehen den Zweck zur Illustration eines Netzwerks mit virtuellen Maschinen</li> </ul>
Installation/ Konfiguration des Fließbandsensors	<ul style="list-style-type: none"> <li>Volumensensor erfolgreich an Conveyor #9 installiert und konfiguriert (Anzahl der Flaschen wird erfasst)</li> <li>Des weiteren schläft der Projektleiter eine Funktions- und Messdatenerweiterung der Senoren und Aktoren vor.</li> </ul>

## TOP 3: Vorstellung der Zeitplanung

Auf Wunsch der geladenen Teilnehmer wird der Zeitplan folgendermaßen umstrukturiert:

- Vor jeder Durchführung (Arbeitsschritt) soll eine dazugehörige Recherchezeit eingepflegt werden
- Für den Durchführungspunkt „Beschaffung der Software“ wird eine Stundenanzahl einkalkuliert (vorher keine Angaben)
- Die Integration des „OPC Router“ in den Zeitplan
- Änderungen des Zeitplans und etwaige Auswirkungen auf den Projektverlauf sollen dokumentiert werden

03.11.2019

#### **TOP 4: SOLL-IST-Analyse**

- Der Projektleiter und die anderen Teilnehmer der Meilensteinsitzung sind mit den Ergebnissen der Analyse zufrieden. Die Erfüllung der Teilziele geschieht im zeitlichen Rahmen.

#### **TOP 5: Weiteres Vorgehen und Ausblick**

Informationen durch Sitzungsleiter Michael Mölle:

- Instruktion der von uns gesteckten Ziele zur zweiten Meilensteinsitzung
- Der Projektleiter Markus Hofmann äußert Bedenken zum pünktlichen Erreichen der Teilziele
- Ebenfalls möchte der Projektleiter Markus Hofmann, dass in der angebundenen Datenbank Produktionsdaten über erfasste Sensordaten angepasst werden
- Die interne Zusammenarbeit (Verbesserung der Kommunikation) wird dokumentiert

#### **TOP 6: Offene Fragerunde**

- Beantwortung der Fragen während der Präsentation
- Es bestehen keine weiteren Fragen zum Projekt

#### **Maßnahmen, die sich aus der Sitzung ergeben haben**

<b>Maßnahme</b>	<b>Verantwortlicher</b>	<b>Bis</b>
Überarbeitung des Zeitplans	Mölle/ Hruby	08.11.19
Organisieren des OPC Routers	Mölle/ Hruby	14.11.19
Dokumentieren der internen Zusammenarbeit	Mölle/ Hruby	13.12.19
Dokumentieren der Zeitplanänderungen und deren Auswirkungen	Mölle/ Hruby	21.02.20
Anpassung der Produktionsdaten in der Datenbank über Sensordaten	Mölle/ Hruby	13.12.19

Die nächste Meilensteinsitzung findet am 13.12.2019 statt.

**Unterschrift Protokollant/-in:**

**Unterschrift Projektleiter/-in:**

## **A.6 Meilensteinprotokoll Nr. 2 vom 19.12.2019**

17.01.2020

### **Protokoll**

Erste Meilensteinsitzung zum Projekt „*Datenbank- und Webanbindung einer virtuellen Fabrik mittels OPC UA*“

17.01.2020, 08:30 Uhr bis 09:45 Uhr, Rudolf-Diesel-Fachschule, Nürnberg

#### **Anwesend:**

Michael Mölle	(Sitzungsleiter)
Ondrej Hruby	(Sitzungsleiter)
Markus Hofmann	(Projektleiter)

#### **Abwesend:**

Johannes Köppel	(entschuldigt)
Rudolf Schneider	(entschuldigt)

#### **Verteiler:**

Teilnehmer

17.01.2020

### **TOP 1: Vorstellung der erreichten Teilziele**

<b>Teilziel</b>	<b>Bemerkung</b>
Installation / Konfiguration der OPC-UA-Clients, Aktoren und Sensoren	<ul style="list-style-type: none"> <li>Dieses Teilziel wird in einer Livepräsentation mittels der Visual Components-Fabrik veranschaulicht.</li> <li>Es wurden insgesamt drei Sensoren und zwei Aktoren installiert und konfiguriert.</li> <li>Dem Projektleiter werden diese vorgestellt und die Funktionsweisen erläutert.</li> <li>Besprechung der Aufgaben der einzelnen Sensor- und Aktorvariablen.</li> <li>Sensoren geben die Daten aus Visual Components an den OPC UA – Client weiter, jedoch nicht die Aktoren.</li> </ul>
Anlegen der Datenbank / Konfiguration der Tabellen	<ul style="list-style-type: none"> <li>In MySQL wurden Tabellen für die exportierten Daten aus Visual Components angelegt.</li> <li>In der Livedemonstration werden Daten in die Datenbank erfolgreich aus dem OPC UA Client importiert und gespeichert.</li> <li>Der Projektleiter schlägt eine zusätzliche Tabelle für die Lagerbestandteile vor.</li> </ul>
Installation / Konfiguration des OPC Routers	<ul style="list-style-type: none"> <li>OPC Router – Konfiguration erläutert und GUI vorgestellt.</li> <li>Statistik der Datenbankzugriffe im OPC Router veranschaulicht.</li> </ul>

### **TOP 2: Weiteres Vorgehen und Ausblick**

Informationen durch Sitzungsleiter Ondrej Hruby:

- Instruktion der von uns gesteckten Ziele zur dritten Meilensteinsitzung
- Der Projektleiter Markus Hofmann äußert keine Bedenken zum pünktlichen Erreichen der Teilziele und ist mit dem bisherigen Verlauf zufrieden
- Ebenfalls möchte der Projektleiter Markus Hofmann, dass die Datenbank um eine Tabelle für Lagerbestände erweitert wird und diese auch in node red ausgewertet wird

### **TOP 3: Offene Fragerunde**

- Beantwortung der Fragen während der Präsentation
- Es bestehen keine weiteren Fragen zum Projekt

17.01.2020

**Maßnahmen, die sich aus der Sitzung ergeben haben**

<b>Maßnahme</b>	<b>Verantwortlicher</b>	<b>Bis</b>
Erweiterung der Datenbank um eine Tabelle für Lagerbestände mit zusätzlicher Auswertung in node red	Mölle/ Hruby	21.02.19

Die nächste Meilensteinsitzung findet am 21.02.2020 statt.

**Unterschrift Protokollant/-in:**

**Unterschrift Projektleiter/-in:**

## A.7 Meilensteinprotokoll Nr. 3 vom

## A.8 Tätigkeitsnachweise

Tätigkeitsnachweise Michael Mölle:

### Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
16.09.2019	14:00	16:00	FAU	Projektvereinbarung	2h
16.09.2019	16:00	17:00	FAU	Machbarkeitsanalyse	1h
17.09.2019	09:00	13:00	RDF	Installation virtuelle Maschine	4h
27.09.2019	11:00	14:00	RDF	Kein Zugang EU137 / Lizenzserver ausgefallen	3h
27.09.2019	15:00	16:00	FAU	Projektstrukturplan	1h
27.09.2019	16:00	17:00	FAU	Konfiguration virtuelle Maschine	1h
27.09.2019	18:00	20:00	FAU	Projektzeitplan	2h
29.09.2019	14:30	16:30	Heimat	Konfiguration virtuelle Maschine	2h
29.09.2019	18:00	18:30	Heimat	Dokumentation	0.5h
30.09.2019	17:00	19:00	Ohm-Bibliothek	Simplen OPC-UA-Client in JavaScript zu Testzwecken geschrieben	2h
30.09.2019	19:00	20:00	Ohm-Bibliothek	Verbindung zwischen OPC-UA Demo-Server und Testclient erfolgreich hergestellt	1h
04.10.2019	10:30	10:45	RDF	Erhalt der Einzellizenz für die Software „Visual Components“	0.25h
04.10.2019	11:00	12:45	RDF	Im Raum EU137 erste Eindrücke in „Visual Components“ mit dem „Bottle Filling Layout“ gesammelt	1.75h
04.10.2019	14:00	15:00	Heimat	Einführendes Tutorial für „Visual Components“ durchgearbeitet	1h
04.10.2019	15:00	18:00	Heimat	Recherche betrieben und Tutorials durchgearbeitet zum Thema „Sensor-Konfiguration“ in „Visual Components“	3h
07.10.2019	12:30	13:30	RDF	Für „Visual Components“ ein AddOn in Python geschrieben, welches es ermöglicht Messages im Output-Feld der Software zu löschen oder in einer separaten Textdatei zu speichern.	1h
10.10.2019	09:30	11:30	Heimat	Durcharbeiten des Tutorials „Component Scripting“ der Visual-Components-Academy	2h
10.10.2019	15:00	16:00	Heimat	Durcharbeiten des Tutorials „Create New Tasks in Works Library“ der Visual-Components-Academy	1h
10.10.2019	16:00	17:00	Heimat	Durcharbeiten des Tutorials „Conditions and Signals“ der Visual-Components-Academy	1h
10.10.2019	17:00	18:00	Heimat	Durcharbeiten des Tutorials „Model a Volume Sensor“ der Visual-Components-Academy	1h
10.10.2019	19:00	20:00	Heimat	Erfolgreiche Installation und Grundkonfiguration des Volumensensors am Förderband #9 in „Visual Components“	1h
11.10.2019	10:00	10:30	RDF	Verfassung einer Einladung zur ersten Meilensteinsitzung	0.5h
11.10.2019	10:30	11:30	RDF	Durcharbeiten des Tutorials „Hello World“ von Github und Erstellung eines Github-Repository, zur Verwaltung und Austausch sämtlicher Projektdateien	1h
16.10.2019	16:30	17:00	Ohm-Bibliothek	Fertigstellung des Einladungsschreibens zur ersten Meilensteinsitzung	0.5h
16.10.2019	17:00	18:30	Ohm-Bibliothek	Planung der ersten Meilensteinsitzung	1.5h
16.10.2019	21:30	22:30	Heimat	Dokumentation Konfiguration der virtuellen Maschine	1h
				Gesamt:	36.5h

**Tätigkeitsnachweis***Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
17.10.2019	19:15	20:15	Heimat	SOLL-IST-ANALYSE (Rohfassung) verfasst	1h
18.10.2019	08:45	09:15	RDF	Räumlichkeiten für die 1. Meilensteinsitzung organisiert	0.5h
18.10.2019	09:15	11:45	RDF	Dokumentation „Konfiguration der virtuellen Maschine“	2.5h
18.10.2019	11:45	12:45	RDF	Termin mit Herrn Hofmann	1h
18.10.2019	12:45	15:30	RDF	Überarbeitung der Projektvereinbarung. Überarbeitung des Zeitplans	2.75h
18.10.2019	16:30	17:30	Heimat	Screenshots der „erreichten Teilziele“ für die Präsentation bei der 1. MSS, archiviert und Repository aktualisiert.	1h
18.10.2019	17:30	17:45	Heimat	Fertigstellung des Einladungsschreibens und Versenden an Herrn Hofmann und Herrn Köppel	
20.10.2019	20:00	22:00	Heimat	Dokumentation „Installation/ Konfiguration des Volumensors“ geschrieben und im Repository hochgeladen	2h

21.10.2019	14:00	15:00	RDF	Dokumentation „MessageHelper“ verfasst	1h
21.10.2019	20:00	20:30	Heimat	Überarbeitung des Zeitplans und Upload ins Repository	0.5h
21.10.2019	20:30	22:00	Heimat	Handout für die 1. MSS verfasst	1.5h
22.10.2019	10:00	11:00	RDF	Besprechung des Ablaufs der Präsentation der 1. MSS	1h
22.10.2019	11:45	12:15	RDF	Fertigstellung Dokumentation „AddOn MessageHelper“	0.5h
22.10.2019	12:15	12:45	RDF	Fertigstellung Dokumentation „Konfiguration der VM“	0.5h
22.10.2019	12:45	13:00	RDF	Fertigstellung Dokumentation „Installation/ Konfiguration des Volumensors“	0.25h
22.10.2019	13:00	13:30	RDF	Fertigstellung des Handouts zur 1. MSS	0.5h
24.10.2019	14:00	16:00	RDF	Vorbereitung/ Proben auf 1. MSS	2h
25.10.2019	09:00	12:00	RDF	Vorbereiten des Raums, in dem die 1. MSS stattfand + Durchführung der 1. MSS + Nachfolgende Besprechung	3h
27.10.2019	10:45	11:00	Heimat	Erfassung der Daten für das Protokoll	0.25h
27.10.2019	11:00	12:00	Heimat	Recherche OPC UA Server	1.0h
27.10.2019	12:00	12:15	Heimat	Konnektivität (OPC UA Server mit Visual Components) hergestellt	0.25h
27.10.2019	12:15	13:00	Heimat	In Visual Components gemeinsame Variablenpaare festgelegt	0.75h
27.10.2019	13:00	13:45	Heimat	Recherche OPC Router	0.75h
27.10.2019	13:45	14:00	Heimat	Erinnerungsschreiben an Herrn Hofmann und Downloadanforderung an die Firma inray gesendet	0.25h
28.10.2019	09:45	10:00	Heimat	Download und Installation der Software „OPC Router“	0.25h
28.10.2019	10:00	10:15	Heimat	Upload der aktuellen „Bottle Filling Layout“ von Visual Components ins Repository	0.25h
28.10.2019	10:15	10:30	Heimat	Download/ Installation der Software „UAGateway“	0.25h
28.10.2019	10:30	11:00	Heimat	Durcharbeiten eines Tutorials zum Umgang mit UAGateway	0.5h
28.10.2019	11:00	11:30	Heimat	Einarbeitung in das UA Configuration Tool	0.5h
28.10.2019	11:30	12:30	Heimat	Recherche OPC UA Server Konfiguration	1.0h
28.10.2019	12:30	13:00	Heimat	Erstelltes Variablenpaar in Visual Components bearbeitet. Im UAExpert im Data Access View einen Node angelegt, welche die Variable des Paares referenziert.	0.5h
28.10.2019	13:00	13:15	Heimat	Im Monitoring-Verfahren konnte die Zustandsänderung des booleschen Signals des Sensors erfolgreich festgestellt werden	0.25h
					28.25h

## Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
29.10.2019	12:00	14:00	Hruby	Überarbeitung des Zeitplans	2h
03.11.2019	10:30	12:15	Heimat	Verfassen des Protokolls zur ersten Meilensteinsitzung	1.75h
08.11.2019	08:45	09:15	Heimat	Fertigstellung des Ereignisprotokolls und Zusammenstellung der Abgabedokumente	0.5h
09.11.2019	10:00	11:00	Heimat	Recherche OPC UA Server	1h

## Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
14.11.2019	15:15	17:00	Heimat	Recherche zum OPC UA Server	1.75h
14.11.2019	17:00	18:00	Heimat	Einarbeitung in das Hierarchiesystem des OPC UA C++ Demo-Server	1h
14.11.2019	18:00	18:30	Heimat	In Visual Components zu Testzwecken ein Variablenpaar (Int32) zusammen mit dem OPC UA C++ Demo-Server angelegt (erfolglos)	0.5h
15.11.2019	08:00	09:00	RDF	Recherche zum OPC UA Server	1h
15.11.2019	09:00	10:30	RDF	Implementierung eines simplen OPC UA Servers in JavaScript (nodeJS), Definieren von Objekten (Sensoren) und deren Variablen	1.5h
15.11.2019	11:30	11:45	RDF	OPC UA Server (JavaScript) erfolgreich mit UAExpert verbunden	0.25h
15.11.2019	11:45	12:00	RDF	OPC UA Server (JavaScript) erfolgreich mit Visual Components verbunden	0.25h
15.11.2019	12:00	13:30	RDF	In Visual Components an das Fließband einen Sensor (vorkonfiguriert zum Flaschenzählern) installiert. Dessen CounterSignal-Variablen (Int32) mit einer Variable des selben Datentyps im Server verbunden	1.5h

## Tätigkeitsnachweis

*Mölle, Michael*

## Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tatigkeit	Zeit
10.12.2019	16:00	17:00	Heimat	E-Mail-Verkehr mit Herrn Hofmann und Herrn Schuler	1h
12.12.2019	08:00	09:00	RDF	Dokumentation "Konfiguration der Datenbank"	1h
12.12.2019	09:00	09:15	RDF	Einladungsmail zur zweiten Meilensteinsitzung verfasst	0.25h
16.12.2019	09:00	10:00	RDF	Handout von Hruby gelesen und uberarbeitet	1h
26.12.2019	10:00	13:30	Heimat	Konfiguration der Datenbank	3.5h
26.12.2019	13:30	14:30	Heimat	Aktualisierung des Zeitplans, des Repositorys und Mail an Hruby	1h
27.12.2019	13:30	15:00	Heimat	Konfiguration der Datenbank	1.5h
27.12.2019	15:00	15:30	Heimat	Hruby eingewiesen	0.5h

## Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
07.01.2020	08:00	09:30	RDF	Besprechung der Abschlussdokumentation	1.5h
07.01.2020	15:00	18:00	Heimat	Abschlussdokumentation	3h
10.01.2020	09:30	11:45	RDF	Abschlussdokumentation	2.25h
14.01.2020	17:00	19:00	Heimat	Planung der zweiten Meilensteinsitzung	2h
17.01.2020	08:00	11:00	RDF	Zweite Meilensteinsitzung + Vorführung der Projektarbeit für neue Schüler	3h
19.01.2020	10:00	12:00	Heimat	Abschlussdokumentation	2h

13.75h

## Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
20.01.2020	15:00	17:00	Heimat	Abschlussdokumentation	2h
24.01.2020	20:00	22:00	Heimat	Abschlussdokumentation	2h
31.01.2020	08:00	12:00	Ohm Bibliothek	Abschlussdokumentation	3h
01.02.2020	10:00	11:00	Heimat	Abschlussdokumentation	1h
03.02.2020	17:00	21:00	Heimat	Abschlussdokumentation	4h

12h

## Tätigkeitsnachweis

*Mölle, Michael*

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
04.02.2020	16:00	18:00	Heimat	Abschlussdokumentation	2h
06.02.2020	18:00	20:00	Heimat	Abschlussdokumentation	2h
08.02.2020	18:00	20:00	Heimat	Abschlussdokumentation	2h
11.02.2020	10:00	12:00	RDF	Abschlussdokumentation	2h
12.02.2020	17:00	19:00	Heimat	Abschlussdokumentation	2h
13.02.2020	17:00	19:00	Heimat	Abschlussdokumentation	2h
14.02.2020	14:00	17:00	Heimat	Abschlussdokumentation	3h
15.02.2020	15:00	18:00	Heimat	Abschlussdokumentation	3h

## Tätigkeitsnachweise Ondrej Hruby:

**Tätigkeitsnachweis****Hruby, Ondrej**

Datum	Start	Ende	Ort	Beschreibung der Tätigkeit	Zeit
16.09.2019	14:00	16:00	zu Hause	Projektvereinbarung + Machbarkeitsanalyse	02:00
17.09.2019	12:00	14:00	RDF	Installation der virtuellen Maschine	02:00
17.09.2019	17:00	20:00	zu Hause	Konfiguration der virtuellen Maschine	03:00
19.09.2019	20:00	22:00	zu Hause	node-red und OA-Expert Installation	02:00
27.09.2019	14:30	18:00	FAU	UA-Demo-Server-Installation, Ausarbeitung der Projektzeitplanung.	03:30
27.09.2019	17:30	18:30	zu Hause	Prosys-OPC-UA-Client Installation	01:00
30.09.2019	14:00	14:30	FAU	Dokumentation	00:30
04.10.2019	10:00	11:00	RDF	Visual Components - Lizenz erhalten	01:00
04.10.2019	11:00	12:30	RDF	erste Eindrücke mit Visual Components gesammelt	01:30
06.10.2019	19:00	20:00	zu Hause	Visual Components Installation	01:00
09.10.2019	18:30	20:30	zu Hause	Informationsbeschaffung über OPC UA,	02:00
10.10.2019	22:00	02:00	zu Hause	Zählsensoreinrichtung in VC + Konfiguration (unvollständ.)	
				Verbindung von VC (auf PC1) zu OPC UA -Server (VM auf PC2)	04:00
12.10.2019	22:30	23:30	zu Hause	Projektstrukturplan überarbeitet	01:00
12.10.2019	23:30	01:30	zu Hause	Zählsensoreinrichtung in VC + Konfiguration (unvollständ.)	02:00
16.10.2019	16:30	18:30	TIM	Vorbereitung des Vortrags für die 1. Meilensteinsitzung	02:00
16.10.2019	19:30	21:00	zu Hause	Verbesserung des Zeitplans + Präsentation für 1.Mss.	01:30
18.10.2019	10:30	14:30	RDF	Recherche für OPCUA Server Kommunikation, Besprechung und Neugestaltung der Machbarkeitsanalyse + Zeitplans	04:00
18.10.2019	18:30	20:00	RDF	Ergänzung des Zeitplans	01:30
18.10.2019	20:00	21:30	zu Hause	Erstellung der Präsentation für 1. Mss.	01:30
21.10.2019	20:00	21:30	zu Hause	Bearbeitung der Präsentation / des Zeitplans	01:30
21.10.2019	22:00	23:30	zu Hause	Recherche für OPCUA Server Kommunikation	01:30
23.10.2019	23:00	00:00	zu Hause	Bearbeitung der Präsentation für 1. Mss	01:00
28.10.2019	12:00	13:30	zu Hause	Überarbeitung des Zeitplans	01:30
29.10.2019	15:30	17:30	zu Hause	Recherche über Konfiguration des OPC UA Clients	02:00
29.10.2019	12:00	14:00	zu Hause	Überarbeitung des Zeitplans	02:00
01.11.2019	10:30	18:00	zu Hause	Recherche über Konfiguration des OPC UA Clients	07:30
				Kurzarbeitsvorbereitung	
21.11.2019	21:00	01:30	zu Hause	Fabrik-Templates analysiert	04:30
22.11.2019	13:00	18:00	zu Hause	Recherche + Configuration einer Batchvariable	05:00
24.11.2019	21:00	01:00	zu Hause	Configuration eines Time-Stamp	04:00
25.11.2019	17:00	20:00	zu Hause	Filling Maschine analysiert, Feinschliff	03:00
28.11.2019	22:00	02:00	zu Hause	Sortieranlage konfigurieren + Recherche	04:00
29.11.2019	12:00	21:00	zu Hause	Sortieranlage konfigurieren + Recherche	09:00
29.00.2019	14:15	14:30	zu Hause	Telefonat mit Herrn Meyer bezüglich VC (Support)	00:15
30.11.2019	22:00	01:00	zu Hause	Auswertung der Daten, Anpassung der Sensoren, 1. Aktorconfiguration	03:00
07.12.2019	11:00	12:00	zu Hause	Treffen mit Projektpartner (Meilensteinbesprechung, Informationsaustausch)	01:00
07.12.2019	17:00	21:30	zu Hause	Dokumentation	04:30
15.12.2019	20:00	21:00	zu Hause	Hand Out Vorbereitung	01:00
17.12.2019	21:00	21:45	zu Hause	Hand Out Vorbereitung	00:45
27.12.2019	21:00	1:00	zu Hause	Node-red Installation und Konfiguration + Node Red Recherche	04:00
29.12.2019	18:30	21:00	zu Hause	Einrichtung der Node-Red-Verbindungen zu OPCUA Client und der Datenbank	02:30
30.12.2019	10:30	11:00	zu Hause	Treffen mit Projektpartner (Meilensteinbesprechung, Informationsaustausch)	00:30
04.01.2020	21:30	02:30	zu Hause	Einrichtung der Node-Red-Verbindungen zu OPCUA Client und der Datenbank	05:00
05.01.2020	10:30	15:00	FAU	Dokumentation	04:30
05.01.2020	22:00	02:45	zu Hause	Einrichtung der Node-Red-Verbindungen zu OPCUA Client und der Datenbank	04:45
14.01.2020	18:00	21:15	zu Hause	Fehlerbehebung bei Node Red: Node-Red startet nicht, Nodes funktionieren nicht mehr	03:15
15.01.2020	19:00	23:15	zu Hause	das gleiche Problem wie am Vortag, anschließende Neuinstallation von Node-Red	04:15
16.01.2020	21:30	00:45	zu Hause	Node Red: Datenbank INSERT-Befehl konfiguriert	03:15
17.01.2020	22:45	00:30	zu Hause	Node Red: Recherche über Boolean-Umwandlung betrieben	01:45
31.01.2020	09:00	12:00	TIM	Dokumentation	03:00
02.02.2020	14:00	18:00	FAU	Dokumentation	04:00

28.10.2019	12:00	13:30	zu Hause	Überarbeitung des Zeitplans	01:30
29.10.2019	15:30	17:30	zu Hause	Recherche über Konfiguration des OPC UA Clients	02:00
29.10.2019	12:00	14:00	zu Hause	Überarbeitung des Zeitplans	02:00
01.11.2019	10:30	18:00	zu Hause	Recherche über Konfiguration des OPC UA Clients	07:30
				Kurzarbeitvorbereitung	
21.11.2019	21:00	01:30	zu Hause	Fabrik-Templates analysiert	04:30
22.11.2019	13:00	18:00	zu Hause	Recherche + Configuration einer Batchvariable	05:00
24.11.2019	21:00	01:00	zu Hause	Configuration eines Time-Stamps	04:00
25.11.2019	17:00	20:00	zu Hause	Filling Maschine analysiert, Feinschliff	03:00
28.11.2019	22:00	02:00	zu Hause	Sortieranlage konfigurieren + Recherche	04:00
29.11.2019	12:00	21:00	zu Hause	Sortieranlage konfigurieren + Recherche	09:00
29.11.2019	14:15	14:30	zu Hause	Telefonat mit Herrn Meyer bezüglich VC (Support)	00:15
30.11.2019	22:00	01:00	zu Hause	Auswertung der Daten, Anpassung der Sensoren, 1. Aktorconfiguration	03:00
07.12.2019	11:00	12:00	zu Hause	Treffen mit Projektpartner (Meilensteinbesprechung, Informationsaustausch)	01:00
07.12.2019	17:00	21:30	zu Hause	Dokumentation	04:30
15.12.2019	20:00	21:00	zu Hause	Hand Out Vorbereitung	01:00
17.12.2019	21:00	21:45	zu Hause	Hand Out Vorbereitung	00:45
27.12.2019	21:00	1:00	zu Hause	Node-red Installation und Konfiguration + Node Red Recherche	04:00
29.12.2019	18:30	21:00	zu Hause	Einrichtung der Node-Red-Verbindungen zu OPCUA Client und der Datenbank	02:30
30.12.2019	10:30	11:00	zu Hause	Treffen mit Projektpartner (Meilensteinbesprechung, Informationsaustausch)	00:30
04.01.2020	21:30	02:30	zu Hause	Einrichtung der Node-Red-Verbindungen zu OPCUA Client und der Datenbank	05:00
05.01.2020	10:30	15:00	FAU	Dokumentation	04:30
05.01.2020	22:00	02:45	zu Hause	Einrichtung der Node-Red-Verbindungen zu OPCUA Client und der Datenbank	04:45
14.01.2020	18:00	21:15	zu Hause	Fehlerbehebung bei Node Red: Node-Red startet nicht, Nodes funktionieren nicht mehr	03:15
15.01.2020	19:00	23:15	zu Hause	das gleiche Problem wie am Vortag, anschließende Neuinstallation von Node-Red	04:15
16.01.2020	21:30	00:45	zu Hause	Node Red: Datenbank INSERT-Befehel konfiguriert	03:15
17.01.2020	22:45	00:30	zu Hause	Node Red: Recherche über Boolean-Umwandlung betrieben	01:45
31.01.2020	09:00	12:00	TIM	Dokumentation	03:00
02.02.2020	14:00	18:00	FAU	Dokumentation	04:00
03.02.2020	18:00	20:30	zu Hause	Dokumentation	02:30
12.02.2020	23:30	02:30	zu Hause	Node Red Bestellinterface angepasst (Paletten können noch nicht bestellt werden)	03:00
13.02.2020	15:30	20:00	zu Hause	Node Red Bestellinterface angepasst (Paletten können noch nicht bestellt werden)	04:30
				Versuch Kommunikation zwischen node red und einer upgedateten Datenbank herzustellen	
15.02.2020	17:30	23:30	zu Hause	Versuch Kommunikation zwischen node red und einer upgedateten Datenbank herzustellen	06:00
16.02.2020	10:00	16:00	FAU	Dokumentation	
16.02.2020	21:30	02:00	zu Hause	Node Red Bestellinterface angepasst (Paletten können noch nicht bestellt werden)	04:30
17.02.2020	13:00	18:00	zu Hause	Dokumentation	05:00
17.02.2020	09:00	10:30	zu Hause	Konfiguration der Akteure in node red	01:30
17.02.2020	16:00	21:00	zu Hause	Dokumentation	05:00
18.02.2020	19:00	20:15	zu Hause	Node Red Bestellinterface konfiguriert, Bestellung kann abgegeben werden	01:15
28.02.2020	12:00	17:00	zu Hause	Dokumentation	05:00
01.02.2020	16:00	21:30	zu Hause	Dokumentation	05:30

Gesamt: 178:30

## Anhang B (Projektmaterialien)

### Quellcode für den OPC-UA-Server in JavaScript

```
1  /*global require, setInterval, console */
2  const opcua = require("node-opcua");
3
4  // create an instance of OPCUAServer
5  const server = new opcua.OPCUAServer({
6      port: 4334, // the port of the listening socket of the server
7      resourcePath: "/UA/MyLittleServer", // this path will be added to the endpoint resource name
8      buildInfo : {
9          productName: "MySampleServer1",
10         buildNumber: "7658",
11         buildDate: new Date(2014,5,2)
12     }
13 });
14
15 function post_initialize() {
16     console.log("initialized");
17     function construct_my_address_space(server) {
18
19         const addressSpace = server.engine.addressSpace;
20         const namespace = addressSpace.getOwnNamespace();
21
22         // declare a new object
23         const sensor1 = namespace.addObject({
24             organizedBy: addressSpace.rootFolder.objects,
25             browseName: "Sensor1"
26         });
27         // declare a new object
28         const sensor2 = namespace.addObject({
29             organizedBy: addressSpace.rootFolder.objects,
30             browseName: "Sensor2"
31         });
32
33         const sensor3 = namespace.addObject({
34             organizedBy: addressSpace.rootFolder.objects,
35             browseName: "Sensor3"
36         });
37         //-----
38         //Variablen für das Objekt sensor1 definieren und dem Namespace hinzufügen
39         let variable1_s1 = 0;
40         namespace.addVariable({
41
42             componentOf: sensor1,
43             browseName: "Variable_1",
44             dataType: "Int32",
45             value: {
46                 get: function () {
47                     return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable1_s1 });
48                 },
49                 set: function (variant) {
50                     variable1_s1 = parseFloat(variant.value);
51                     return opcua.StatusCodes.Good;
52                 }
53             }
54         });
55         // zweite Variable fuer Sensor1 - read and write
56         let variable2_s1 = 0;
57         namespace.addVariable({
```

```
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
      componentOf: sensor1,
      browseName: "Variable_2",
      dataType: "Int32",
      value: {
        get: function () {
          return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable2_s1 });
        },
        set: function (variant) {
          variable2_s1 = parseFloat(variant.value);
          return opcua.StatusCodes.Good;
        }
      }
    });
  // dritte Variable fuer Sensor1 - read and write
  let variable3_s1 = 0;
  namespace.addVariable({
    componentOf: sensor1,
    browseName: "Variable_3",
    dataType: "Int32",
    value: {
      get: function () {
        return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable3_s1 });
      },
      set: function (variant) {
        variable3_s1 = parseFloat(variant.value);
        return opcua.StatusCodes.Good;
      }
    }
  });
  // vierte Variable fuer Sensor1
  let variable4_s1 = 0;
  namespace.addVariable({
    componentOf: sensor1,
    browseName: "Variable_4",
    dataType: "Int32",
    value: {
      get: function () {
        return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable4_s1 });
      },
      set: function (variant) {
        variable4_s1 = parseFloat(variant.value);
        return opcua.StatusCodes.Good;
      }
    }
  });

  //-----
  //Variablen für das Objekt sensor2 definieren und dem Namespace hinzufügen
  let variable1_s2 = 0;
  namespace.addVariable({
    componentOf: sensor2,
    browseName: "Variable_1",
    dataType: "Int32",
    value: {
      get: function () {
        return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable1_s2 });
      },
      set: function (variant) {
        variable1_s2 = parseFloat(variant.value);
        return opcua.StatusCodes.Good;
      }
    }
  });

```

```
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
    }
  });
  //Zweite Variable für sensor2 - read and write
let variable2_s2 = 0;
namespace.addVariable({
  componentOf: sensor2,
  browseName: "Variable_2",
  dataType: "Int32",
  value: {
    get: function () {
      return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable2_s2 });
    },
    set: function (variant) {
      variable2_s2 = parseFloat(variant.value);
      return opcua.StatusCodes.Good;
    }
  }
});
//Dritte Variable für sensor2 - read and write
let variable3_s2 = 0;
namespace.addVariable({
  componentOf: sensor2,
  browseName: "Variable_3",
  dataType: "Int32",
  value: {
    get: function () {
      return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable3_s2 });
    },
    set: function (variant) {
      variable3_s2 = parseFloat(variant.value);
      return opcua.StatusCodes.Good;
    }
  }
});
//Vierte Variable für sensor2 - read and write
let variable4_s2 = 0;
namespace.addVariable({
  componentOf: sensor2,
  browseName: "Variable_4",
  dataType: "Int32",
  value: {
    get: function () {
      return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable4_s2 });
    },
    set: function (variant) {
      variable4_s2 = parseFloat(variant.value);
      return opcua.StatusCodes.Good;
    }
  }
});
//-----
//Variablen für das Objekt sensor3 definieren und dem Namespace hinzufügen
let variable1_s3 = 0;
namespace.addVariable({
  componentOf: sensor3,
  browseName: "Variable_1",
  dataType: "Int32",
  value: {
    get: function () {
      return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable1_s3 });
    },
  }
},
```

```
188     set: function (variant) {
189         variable1_s3 = parseFloat(variant.value);
190         return opcua.StatusCodes.Good;
191     }
192 });
193 //Zweite Variable für sensor2 - read and write
194 let variable2_s3 = 0;
195 namespace.addVariable({
196
197     componentOf: sensor3,
198     browseName: "Variable_2",
199     dataType: "Int32",
200     value: {
201         get: function () {
202             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable2_s3 });
203         },
204         set: function (variant) {
205             variable2_s3 = parseFloat(variant.value);
206             return opcua.StatusCodes.Good;
207         }
208     }
209 });
210 //Dritte Variable für sensor2 - read and write
211 let variable3_s3 = 0;
212 namespace.addVariable({
213     componentOf: sensor3,
214     browseName: "Variable_3",
215     dataType: "Int32",
216     value: {
217         get: function () {
218             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable3_s3 });
219         },
220         set: function (variant) {
221             variable3_s3 = parseFloat(variant.value);
222             return opcua.StatusCodes.Good;
223         }
224     }
225 });
226 //Vierte Variable für sensor2 - read and write
227 let variable4_s3 = 0;
228 namespace.addVariable({
229     componentOf: sensor3,
230     browseName: "Variable_4",
231     dataType: "Int32",
232     value: {
233         get: function () {
234             return new opcua.Variant({dataType: opcua.DataType.Int32, value: variable4_s3 });
235         },
236         set: function (variant) {
237             variable4_s3 = parseFloat(variant.value);
238             return opcua.StatusCodes.Good;
239         }
240     }
241 });
242 }
243
244 construct_my_address_space(server);
245 server.start(function() {
246     console.log("Server is now listening ... ( press CTRL+C to stop)");
247     console.log("port ", server.endpoints[0].port);
248     const endpointUrl = server.endpoints[0].endpointDescriptions()[0].endpointUrl;
249     console.log(" the primary server endpoint url is ", endpointUrl );
250 });
251
252 server.initialize(post_initialize);
```

## Quellcode für das MessageHelper-AddOn

```
1  from vcCommand import *
2
3  cmd = getCommand()
4  app = getApplication()
5
6  def clearMessages(prop):
7      app.clearMessages()
8
9  def saveMessages(prop):
10     msg = app.getMessages()
11     uri = r"C:\Users\Michael\Documents\Visual Components\4.1\output.txt"
12     with open(uri, "a") as f:
13         f.write(msg)
14
15 def first_state():
16     executeInActionPanel()
17
18 addState(first_state)
19
20 btnClear = cmd.createProperty(VC_BUTTON, "Clear Messages")
21 btnClear.OnChanged = clearMessages
22
23 btnSave = cmd.createProperty(VC_BUTTON, "Save Messages")
24 btnSave.OnChanged = saveMessages
```

```
1  from vcApplication import *
2
3  def OnStart():
4      cmduri = getApplicationPath() + "messageHelper.py"
5      cmd = loadCommand("messageHelper", cmduri)
6      addMenuItem("VcTabHome/Test", "messageHelper", -1, "messageHelper")
7
```

## Quellcode für den Volumensensor in Visual Components

```
1  from vcScript import *
2
3  comp = getComponent()
4  sensor = comp.findBehaviour("VolumeSensor")
5  #Deklaration und Initialisierung einer Zählvariable
6  bottle_counter = 0
7
8  def OnSignal( signal ):
9      #print signal.Value
10     global bottle_counter
11     bottle_counter += 1
12     print "Filled bottle no. ", bottle_counter
13
14 def OnRun():
15     pass
```

## Quellcode für die relationale Datenbank

```
1  #Anlegen der Datenbank für Sensordaten-----#
2  ● CREATE DATABASE IF NOT EXISTS `BottleFillingLayout`  
3      CHARACTER SET utf8 COLLATE utf8_unicode_ci;  
4  
5  ● USE `BottleFillingLayout`;  
6  
7  #Tabelle wurde mittels DROP TABLE wieder gelöscht-----#
8  #-----#
9  #Erstellung der endgültigen Tabellenaufteilung-----#  
10 ●  
11  ● CREATE TABLE IF NOT EXISTS `Limonaden_Produziert`  
12      (`FlaschenID` INT NOT NULL,  
13       `FlaschenID_Palette` INT NOT NULL,  
14       `PalettenID` INT NOT NULL,  
15       `Abfuell-Datum` VARCHAR(30) NOT NULL);  
16  
17  ● CREATE TABLE IF NOT EXISTS `Limonaden_Defekt`  
18      (`FlaschenID` INT NOT NULL,  
19       `Abfuell-Datum` VARCHAR(30) NOT NULL);  
20  
21  ● CREATE TABLE IF NOT EXISTS `Bestellungen`  
22      (`Name` VARCHAR(50) NOT NULL,  
23       `Palettenanzahl` INT NOT NULL);  
24  
25  ● CREATE TABLE IF NOT EXISTS `Vorratslager`(  
26      `Leere Flaschen` INT NOT NULL,  
27      `Flaschendeckel` INT NOT NULL);  
28  
29  #Befüllen des Lagers mit Komponenten  
30  ● INSERT INTO `Vorratslager`(`Leere Flaschen`, `Flaschendeckel`)VALUES  
31  (50000,50000);  
32  
33  #Erstellung der Trigger zur Wertänderung der Tabelle Vorratslager  
34  DELIMITER |  
35  ● CREATE TRIGGER lager_flaschen_abzug  
36      BEFORE INSERT ON `Limonaden_Produziert`  
37      FOR EACH ROW  
38      BEGIN  
39          INSERT INTO `Vorratslager`(`Leere Flaschen`, `Flaschendeckel`)  
40          SELECT MIN(`Leere Flaschen`)-1, MIN(`Flaschendeckel`)-1  
41          FROM `Vorratslager`;  
42      END |  
43  DELIMITER ;  
44  
45  DELIMITER |  
46  ● CREATE TRIGGER lager_flaschen_abzug2  
47      BEFORE INSERT ON `Limonaden_Defekt`  
48      FOR EACH ROW  
49      BEGIN  
50          INSERT INTO `Vorratslager`(`Leere Flaschen`, `Flaschendeckel`)  
51          SELECT MIN(`Leere Flaschen`)-1, MIN(`Flaschendeckel`)-1  
52          FROM `Vorratslager`;  
53      END |  
54  DELIMITER ;  
--
```

```

55
56 #Anlegen einer MySQL-Variable
57
58
59 #Erstellen eines Triggers zur Wertveränderung der Tabelle Limonaden_Produziert
60 DELIMITER |
61 • CREATE TRIGGER bestellungen_abzug
62 BEFORE INSERT ON `Bestellungen`
63 FOR EACH ROW
64 BEGIN
65     SET @FLASCHENANZAHL := 1;
66     SET @FLASCHENANZAHL = NEW.`Palettenanzahl` * 800;
67     #Löschen der Flaschen und Paletten
68     WHILE @FLASCHENANZAHL > 0
69     DO
70         DELETE FROM `Limonaden_produziert` WHERE `FlaschenID` IN (
71             SELECT MIN(`FlaschenID`));
72         SET @FLASCHENANZAHL = @FLASCHENANZAHL - 1;
73     END WHILE;
74 END |
75
76 • SHOW TRIGGERS;
77
78 #Anlegen der Unique-Schlüssel zur Sicherung der Datenintegrität
79
80 ALTER TABLE `bottlefillinglayout`.`limonaden_produziert`
81 ADD UNIQUE `Unique_Limonaden_Produziert` (`FlaschenID`, `FlaschenID_Palette`, `PalettenID`, `Abfuell-Datum`);
82
83 ALTER TABLE `bottlefillinglayout`.`limonaden_defekt`
84 ADD UNIQUE `FlaschenID` (`FlaschenID`, `Abfuell-Datum`);

```

