HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Ingenieurwissenschaften

6010 - Praxisprojekt

Implementierung einer OCPP-Backend-Testumgebung für automatische Integrationstests einer DC High Power Ladestationen der Plattform Sicharge D

Author Ivan Agibalov
Betreuer Andre Vieweg

5. Juli 2022

Inhaltsverzeichnis

1 Aufgabenstellung **1**

1.1 Anforderungen an den Standalone Server 1

1.2 Anforderungen an das Testframework 2

1.3 Anforderungen an ERK Automatisierungstool 2

2 Umsetzung **3**

2.1 OCPP Standalone Server und Testframework 3

2.2 ERK Automatisierungstool 5

1 Aufgabenstellung

Zu Begin der Praxisphase war die Entwicklung und Testen an vielen Stellen an eine externe Schnittstelle gebunden. Um diese Abhängigkeit zu reduzieren und die Möglichkeit zu bekommen eigene Funktionalitäten hinzuzufügen, bekammm ich die Aufgabe einen OCPP 1.6. Server zu implementieren.

Zu dem Zeitpunkt gab es bereits drei Stellen, bei denen der OCPP1.6 Server Funktionalitäten in unterschiedlichem Umfang gebraucht wurden.

Das sind:

- Testframework für die automatisierten Systemtests von der Ladesäule
- Eigenständiger OCPP1.6 und später OCPP2.0 Server für die manuellen Tests
- Automatisierungstool für den ERK (Eichrechtkonformität) Prozess

Die dritte Aufgabe (Automatisierungstool für den ERK Prozess) hatte größere Priorität und somit wurde diese als erstes erledigt. Die Aufgabe hat dabei den kleinsten Anteil an OCPP Server Funktionalitäten gebraucht.

Anschließend wurden die Teilaufgaben erledigt, aufgrund der großen Überschneidung der programmiertechnischen Anforderungen, konnten Teile der vorhergehenden Aufgabe übernommen werden.

1.1 Anforderungen an den Standalone Server

Der Server, der im lokalen Netz auf einem Raspberry Pi läuft, soll für die manuellen Tests der Ladesäule benutzt werden. Dieser Server wird benutzt um die komplexeren Testfälle nachzubilden oder neue Funktionalitäten, die mit Hardware interagieren, zu testen. Der Server kann bei der Präsentation der Funktionalitäten genutzt werden.

Die Anforderungen an den Server sind:

- Der Server soll eine OCPP1.6 Schnittstelle besitzen.
- Der Nutzer soll in der Lage sein den Server zu parametrieren (z.B. einen neuen Benutzer hinterlegen)
- Der Nutzer soll in der Lage sein die Nachrichten an die Ladesäule manuell verschicken zu können

1.2 Anforderungen an das Testframework

Das Testframework soll für die automatisierten Systemtests der Software der Ladesäule (sowohl mit als auch ohne Hardware) benutzt werden.

Die Anforderungen an das Testframework sind:

- Der OCPP Server soll den Port selber auswählen können, um mehrere Tests parallel starten zu können.
- Das Verhalten von dem Testserver soll geändert werden können (auch während der Tests)
- Das Defaultverhalten von dem Testserver soll parametrierbar sein (z.b. einen Benutzer hinzufügen)
- Alle Events, die den Zustand der getesteten Ladesäule aufdecken, sollen beobachtbar sein (z.B. OCPP Nachrichten, Netzwerkevents usw.)

1.3 Anforderungen an ERK Automatisierungstool

Der Zertifizierung nach dem deutschen Eichrecht entsprechend, muss jede Ladesäule auf Eichrechtskonformität (ERK) überprüft werden. Dieser Prozess wird immer wieder auf die gleiche Art und Weise im gleichen Umfang durchgeführt. Er beinhaltet somit ein entsprechendes Automatisierungspotential.

Dafür muss für jede Ladesäule ein Ladevorgang gestartet werden (Transaction), währenddessen Strom fließt und gemessen wird. Um die Datenintegrität der Messwerte und deren Transport sicherzustellen wird der OCPP Server verwendet. Mit dessen Hilfe kann nachgewiesen werden, dass die Daten nirgendwo in der Software geändert und ebenso unverändert an den Server übertragen und dort abgerechnet wurden. Nach Beenden der Transaction werden mittels einer Drittsoftware die gemessenen Daten mit den Transactiondaten verglichen.

Die gewünschte Software soll demnächst von den Mitarbeitern im End-Of-Line benutzt werden, somit muss die Bedienbarkeit der Software sehr hoch sein, um die Fehlermöglichkeiten stark einzugrenzen und die Einarbeitungszeit zu reduzieren.

Die Anforderungen an das ERK Automatisierungstool sind:

- Leichte Bedienbarkeit der Software
- Leicht Integrierbar in das andere Automatisierungstool

2 Umsetzung

Alle gestellten Aufgaben benötigen einen OCPP Server um die OCPP Kommunikation mit der Ladesäule auswerten zu können.

Der gemeinsame Teil des Servers für alle Anwendungen ist die serverseitige OCPP Kommunikation. Das Verhalten auf verschiedene Nachrichten soll dabei abhörbar und leicht änderbar sein, um es für die jeweilige Anwendung anpassen zu können.

Um den Server so flexibel wie möglich zu planen, wurde von meiner Seite entschieden, ihn nach **Clean Architecture** zu implementieren.

2.1 OCPP Standalone Server und Testframework

Der OCPP Standalone Server benutzt mehrere Schnittstellen nach Außen:

- Persistenz (Datenbank und Logging)
- WS Server (OCPP)
- HTTP Server (zum Parametrieren des Verhaltens)

Es wird nur eine WS Server Schnittstelle vom Testframework gebraucht. Damit man dies mit geringsten Kosten wie möglich umsetzt, werden die Schnittstellen nach Außen mittels **Dependency Injection** benutzt.

Somit lassen viele Schnittstelle vom Server bei der Implementierung des Testframeworks gemockt bzw. gefälscht werden, ohne dass man jegliches Verhalten des Programms dadurch ändern muss.

Das Framework braucht neben dem Defaultverhalten des Programms als OCPP Server noch weitere Funktionalitäten des Testframeworks das sind das Abwarten der bestimmten Events im Server um deren Inhalt auszuwerten und das Ändern und das Parametrieren des Defaultverhaltens um bestimmte Situation nachbilden zu können.

Das Beobachten der Events wird mittels OOP Design Pattern **Observer** implementiert, indem man gewisse Ereignisse abonnieren kann. Das Ändern und das Parametrieren des Verhalten des Servers wird mittels OOP Design Pattern **Facade** und **Strategy** möglich gemacht.

Die **Facade** ermöglicht die komplexe Architecture des Programms für den Nutzer zu verdecken um nur die wichtigen Methoden sichtbar zu machen. Die **Strategy** ermöglicht während das Programm läuft, das Verhalten dynamisch umzuschreiben.

Beispielablauf des Frameworks:

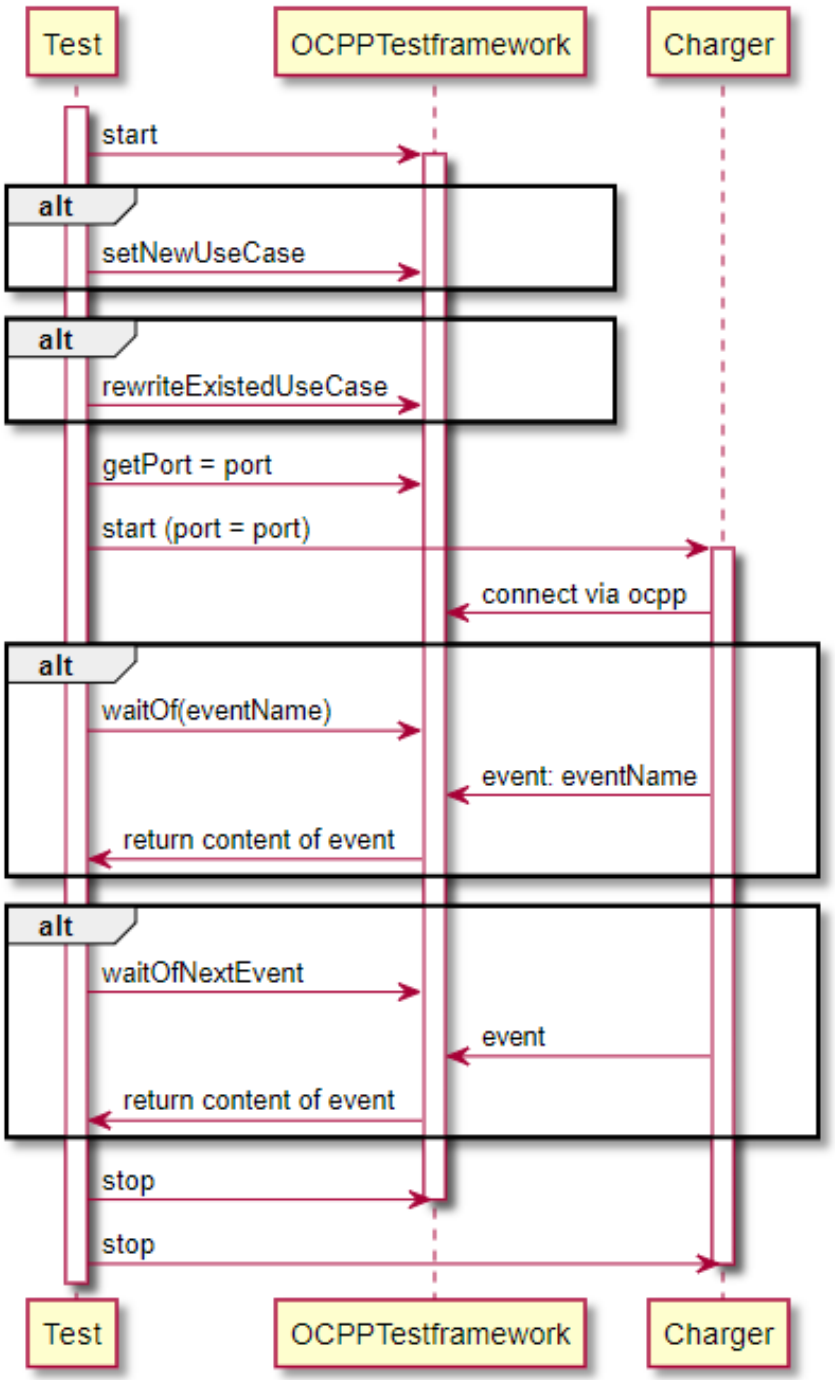

Abbildung 1: Sequenzdiagramm Testframework

Source: Eigene Quelle

2.2 ERK Automatisierungstool

Das Automatisierungstool sollte laut der Aufgabenstellung in einer anderen bestehenden Tool integriert werden. Das bereits existierte Tool ist mit Electron geschrieben, daher um das Integrationsprozess so einfach wie möglich zu machen, wurde von meiner Seite entschieden auch mit Electron zu arbeiten. Das ERK Tool soll 2 vorgegebenen Schnittstellen benutzen:

- Die Kommunikationsschnittstelle zu dem Autosimulator
- Die Kommunikationsschnittstelle zu dem Messgerät

In dem bestehenden Tool wurde zur Begin der Umsetzung nur die Kommunikationsschnittstelle zum Autosimulator implementiert, die Kommunikationsschnittstelle zum Messgerät wurde von dem Messgerätehersteller vorgegeben. Die Schnittstelle wurde mit einem Pythonscript implementiert. Damit man die Schnittstelle nicht anpassen soll um verschiedene Fehler zu vermeiden, wird das Pythonscript vom Electronprogramm mit entsprechenden Parametern gestartet, und danach wird nur die Konsoleausgabe empfängt und entsprechend ausgewertet. Am Ende des Ablaufs soll ein PDF Report entstehen, das alle wichitge Informationen über den Testablauf beinhaltet. Neben dem Report entstehen auch 2 weitere Ausgaben:

- Logs von dem Pythonscript
- Alle Messdaten

Die Logs können später benutzt werden um die Fehler im Pythonscript zu finden und um den Testablauf später nachmachen zu können.

Die Messdaten werden aufbewahrt um später die Richtigkeit des Reports nachweisen und mit anderen Werkzeugen überprüfen zu können.

Benutzten Werkzeuge:

- Typescript als Programmiersprache
- Svelte als Frontendframework
- Bulma als CSS Framework

Der komplette Testablauf:

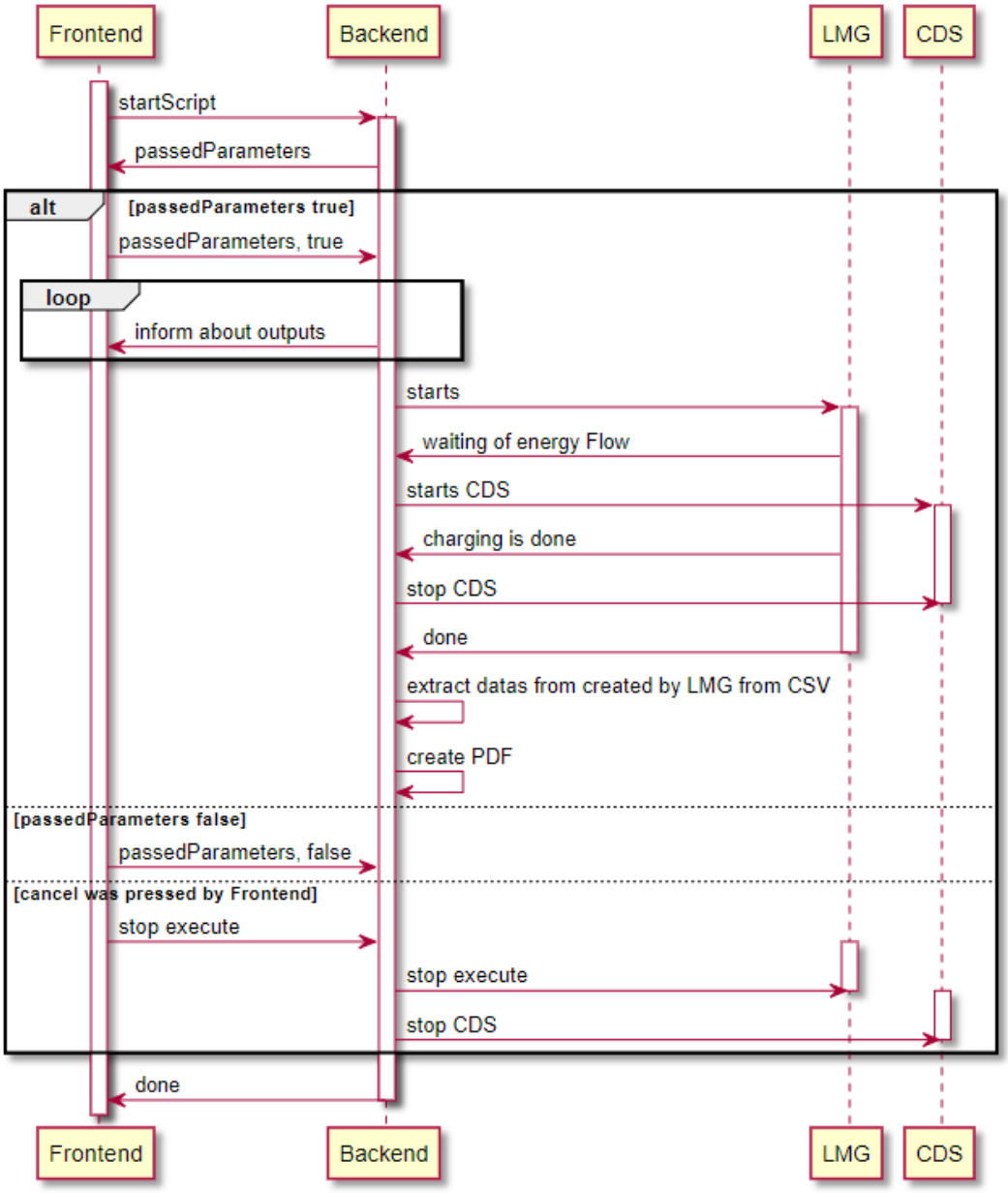

Abbildung 2: Sequenzdiagramm ERK Tool

Source: Eigene Quelle

Beispiel des Reports:

SIEMENS

Accuracy test according to MessEG

DC left

Tested Specifications

Date and Time	21.06.2022 09:52
Name of tester	1
SN of tested Meter	1
FA Charger	1
tested operating point	4A, 1000V

Measurement setup

ZES ZIMMER Electronic Systems GmbH LMG671 ATE 00022104 3.071-R67112

Channel	Group	Type	In Use	I Range	U Range	Sensor
1	1	L60-CH-A2	no	32.0	1000.0	no
2	1	L60-CH-A2	no	32.0	1000.0	no
3	1	L60-CH-A2	no	32.0	1000.0	no
4	2	L60-CH-A2	yes	0.005 (7.5 scaled)	1000.0	PCT600 (SN: 7100060045)
5	3	L60-CH-A2	no	0.6 (200.0 scaled)	1000.0	PCT200 (SN: 0480070039)
6	4	L60-CH-A2	no	32.0	1000.0	no

Measurement results

The 2.00% allowed deviation is used by 98%

Total energy	Total energy: 113.87 Wh
Total deviation	Total deviation: 2.23 Wh (1.96%)
Result	passed

Source: Eigene Quelle