Hes·SO VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

Hes·SO
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Studiengang Systemtechnik

Vertiefungsrichtung Power & Control

Bachelorarbeit
Diplom 2017

Baptiste Cavin

Algorithmus zum Laden von Elektroautos

Dozent
Gilbert Morand

Experte
Johann Eberhardt

Datum der Abgabe des Schlussberichts
18.08.2017

HES-SO Valais-Wallis • rte du Rawyl 47 • C.P. • 1950 Sion 2
+41 27 606 85 23 • hei@hevs.ch • www.hevs.ch

HES-SO Valais

SYND	ETE	TEVI
X	X	X

Données du travail de diplôme
Aufgabenstellung der Bachelorarbeit

FO 1.2.02.07.CB
haj/11/01/2016

Filière / Studiengang **SYND**	Année académique / *Studienjahr* **2016/2017**	No TD / *Nr. DA* **pc/2017/57**
Mandant / *Auftraggeber* ☐ HES—SO Valais ☒ Industrie: winsun AG ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Baptiste Cavin** Professeur / *Dozent* **Gilbert Morand**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Johann Eberhardt** Winsun AG / Besche Mattenstrasse 2, 3940 Steg	

Titre / *Titel*

Algorithmus zum Laden von Elektroautos
Algorithme de recharge pour voitures électriques

Description / *Beschreibung*

Sobald die Elektromobilität an Grösse gewinnt, wird es wesentlich sein, Systeme zu entwickeln, die das Laden der Fahrzeuge intelligent steuern.

Das Ziel dieses Projektes ist ein Optimierungs-Algorithmus zu entwickeln, welcher unter anderem folgende Punkte berücksichtigt:

- die Vorgaben des Benutzers
- die erneuerbaren Ressourcen, die zur Verfügung stehen
- der energetische Bedarf des Haushaltes
- der Bedarf der Ladung der elektrischen Fahrzeuge
- die Einschränkungen des elektrischen Netz-Anschlusses
- die Wettervorhersage

In einem ersten Schritt soll der Fall einer Familie studiert werden, die in einem Einfamilienhaus mit Photovoltaik-Anlage wohnt und sich mit einem Elektroauto fortbewegt.

Der Algorithmus soll genug flexibel sein, um einfach an ein Mehrfamilienhaus mit mehreren Elektrofahrzeugen angepasst zu werden.

Das zu entwickelnde System soll die Ladesäule steuern, um unter Einschränkungen des Benutzers z.B. den Eigenverbrauch zu maximieren.

Objectifs / *Ziele*

– Entwicklung eines Algorithmus zum Laden von Elektroautos

Signature ou visa / *Unterschrift oder Visum*	Délais / *Termine*
Responsable de l'orientation / filière *Leiter der Vertiefungsrichtung / Studiengang:* ..... [1] Etudiant / Student :	Attribution du thème / *Ausgabe des Auftrags:* **15.05.2017** Présentation intermédiaire / *Zwischenpräsentation* **08 – 09.06.2017** Remise du rapport / *Abgabe des Schlussberichts:* **18.08.2017 / 12h00** Expositions / *Ausstellungen der Diplomarbeiten:* **30.08 – 31.08 – 01.09.2017** Défense orale / *Mündliche Verfechtung:* **Semaine 36 / *Woche 36***

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive DI.1.2.02.07 liée au travail de diplôme.*
Durch seine Unterschrift verpflichtet sich der/die Student/in, sich an die Richtlinie DI.1.2.02.07 der Diplomarbeit zu halten.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Algorithmus zum Laden von Elektroautos

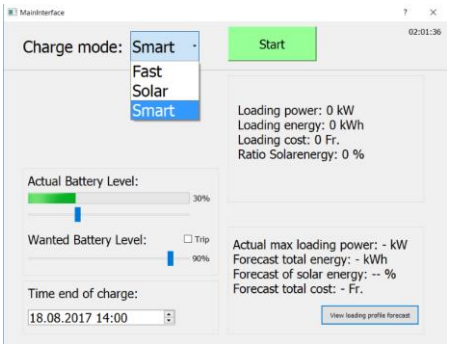
Diplomand Baptiste Cavin

Ziel des Projekts

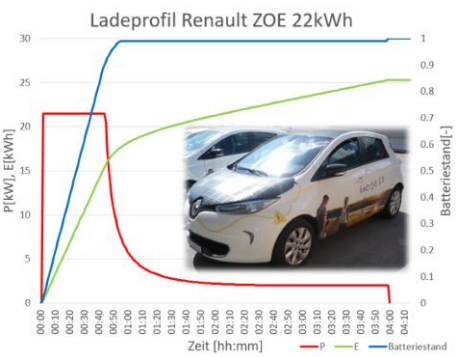
Ziel dieser Arbeit ist es, ein System zu entwickeln, das in Abhängigkeit der Photovoltaikanlage und des Nachtstroms das Ladeprofil von Elektroautos in einem Einfamilienhaus optimiert.

Methoden | Experimente | Resultate

Um den Ladevorgang eines Elektroautos optimieren zu können, wurde eine steuerbare Ladesäule (KEBA P30) verwendet, deren Maximalleistung vorgegeben wird. Das System steuert diese via eine schon vorhandene Plattform. Um mit der Plattform über eine REST-Schnittstelle zu kommunizieren, wurde in einem ersten Teil der Arbeit ein Testprogramm mit Qt Creator® entwickelt. Um den Ladevorgang der Elektroautos (Renault ZOE) zu modellieren, wurde das Testprogramm verwendet und Messungen durchgeführt. Unter anderem wurde festgestellt, dass die vier getesteten ZOEs Blindleistung produzieren und unter 9 kW unregelmässig laden. Zudem unterstützt die Ladesäule keine digitale Kommunikation mit dem ZOE. Das heisst, dass sich das System ausschliesslich auf die manuell eingetragenen Angaben und auf das Ladesimulationsmodell des Elektroautos verlassen muss.
In einem zweiten Teil wurde das *Smart-Charge System* entwickelt, das die Ladung anhand 3 verschiedener Lademodi optimiert: Fast (lädt so schnell wie möglich), Solar (lädt nur mit dem Überschuss der PV-Analge) und Smart (findet dank einem genetischen Algorithmus ein vorteilhaftes Ladeprofil, um das Fahrzeug bis zu einer gewissen Zeit zu laden).



Die Benutzeroberfläche des *Smart-Charge Systems* mit seinen 3 Lademodi. Der Stand der Batterie wird am Anfang der Ladung manuell eingegeben.



Das modellierte Ladeprofil eines Renault ZOE mit einer 22kWh Batterie. In rot die Leistung, in grün die Energie und in blau der Batteriestand.

Diplomarbeit
| 2017 |

Studiengang
Systemtechnik

Anwendungsbereich
Power & Control

Verantwortlicher Dozent
Dr. Gilbert Morand
gilbert.morand@hevs.ch

Partner
winsun AG

Algorithme de recharge pour voitures électriques

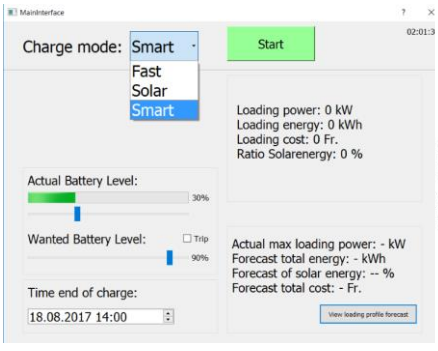
Diplômant Baptiste Cavin

Objectif du projet

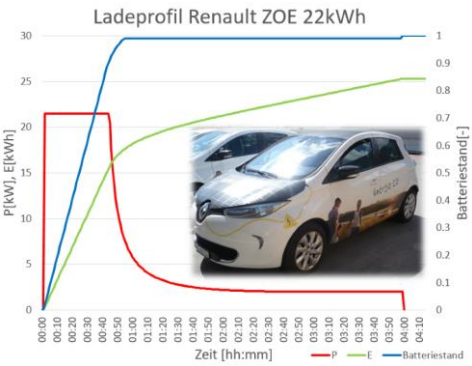
Le but de ce travail est de créer un système qui soit capable d'optimiser le profil de charge d'une voiture électrique dans une maison individuelle en fonction de l'installation photovoltaïque et des heures creuses.

Méthodes | Expériences | Résultats

Afin d'optimiser le profil de charge d'une voiture électrique, une borne de recharge pilotable (KEBA P30), à laquelle on impose une puissance maximale, a été employée. Le système la pilote via une plateforme déjà existante. Pour communiquer par une interface Rest avec la plateforme, un programme test a été réalisé avec Qt Creator®. Ce programme a été utilisé pour effectuer des mesures afin de modéliser le chargement d'une voiture électrique (Renault ZOE). Entre autres, il a été constaté que les quatre ZOEs mesurées produisaient de la puissance réactive et chargeaient de manière irrégulière à des puissances inférieures à 9 kW. La borne ne peut pas communiquer de manière digitale avec la ZOE. De ce fait, le système doit se fier entièrement aux données saisies par l'utilisateur et au modèle de simulation de charge.
Dans un second temps, le système Smart-Charge a été développé. Il permet de choisir entre 3 modes de charge : Fast (charge la voiture le plus rapidement possible), Solar (charge la voiture uniquement avec le surplus photovoltaïque) et Smart (trouve grâce à un algorithme génétique un profil de charge avantageux pour charger la voiture jusqu'à un certain délai).



L'interface utilisateur du système Smart-Charge avec ses 3 modes de charge. Le niveau de la batterie au début doit être rentré manuellement.



Le profil de charge modélisé d'une Renault ZOE avec batterie de 22 kWh. En rouge la puissance, en vert l'énergie et en bleu le niveau de la batterie.

Travail de diplôme
| édition 2017 |

Filière
Systèmes industriels

Domaine d'application
Power & Control

Professeur responsable
Dr. Gilbert Morand
gilbert.morand@hevs.ch

Partenaire
winsun AG

Inhaltsverzeichnis

1 Einführung

Die Elektromobilität entwickelt sich stark seit den letzten Jahren. 2016 waren in der Schweiz 0,24 % der Personenwagen rein elektrisch, das sind 14-mal mehr Elektrowagen als anfangs des Jahrhunderts. Dieser Entwicklung folgen auch die Ladesäulen an öffentlichen Ladestellen oder im Privatbereich. In 90 % der Fälle wird ein Elektrofahrzeug zuhause geladen, was den Stromverbrauch des Haushaltes steigert. Mit zunehmender Anzahl Elektrofahrzeuge kann auch die Netzbelastung bedeutend beeinflusst werden. Darum wird es in Zukunft unerlässlich sein, Lösungen zu finden, um das eigene Elektroauto intelligent zu laden, z.B. mit der Energie der eigenen Photovoltaikanlage. Im Rahmen dieser Bachelorarbeit wird ein System entwickelt, das eine solche Lösung zur Ladeoptimierung von Elektroautos im Privatbereich bietet. So wird Elektromobilität in Zukunft noch nachhaltiger.

2 Problematik

Die Ladung eines Elektroautos, kurz EV (Electric Vehicle), wirkt sich stark auf den elektrischen Verbrauch eines Haushaltes aus, wie es in den nächsten fiktiven Beispielen ersichtlich ist. In der Abbildung 1 sieht man die produzierte Leistung einer Photovoltaikanlage, kurz PV-Anlage, sowie die verbrauchte elektrische Leistung eines Hauses während 24 Stunden. Die PV-Anlage produziert 19 kWh und das Haus verbraucht 20 kWh. Die Eigenverbrauchsrate beträgt nur 51 %. Diese sagt aus, welcher Anteil der eigenproduzierten PV-Energie selber gebraucht wird. Die überschüssige Energie wird ins Netz eingespeist. Wenn man für den Tagesstrom 20 Rp., für den Nachtstrom (von 22:00 bis 06:00) 15 Rp. und für den Einspeisepreis 5 Rp. pro kWh annimmt, ergibt dies ein Total von 1.50 Fr. pro Tag.

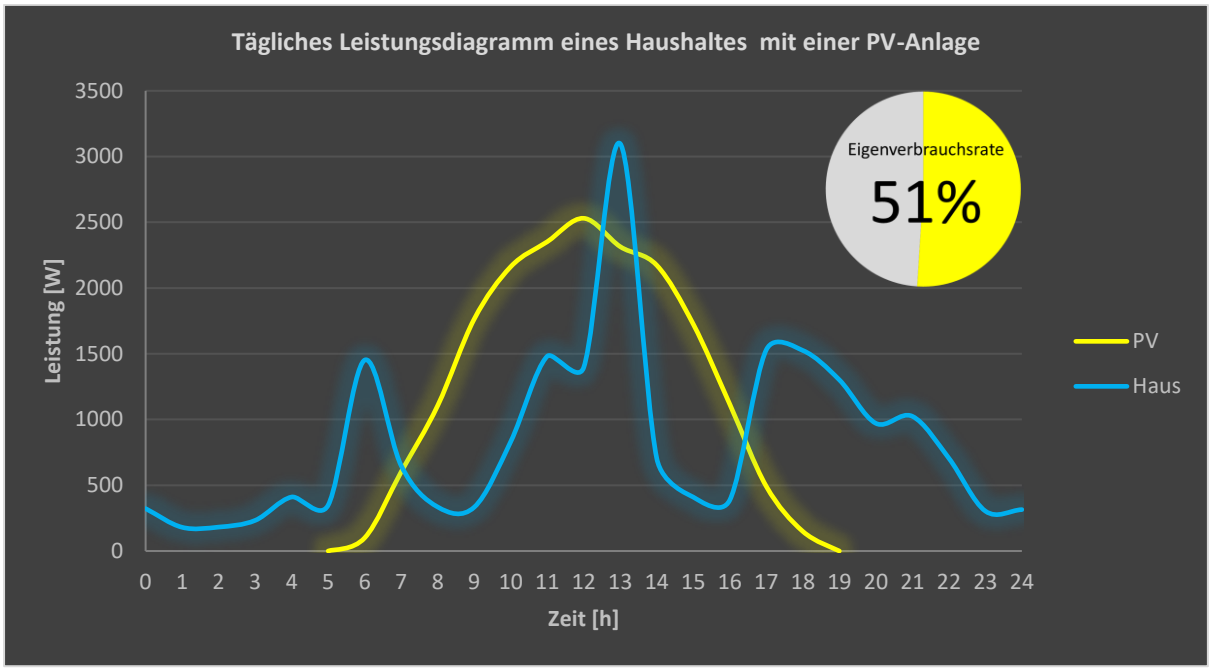


Abbildung 1: Tägliches Leistungsdiagramm eines Hauses ohne EV

In unserem Beispiel kommt der Benutzer mit seinem EV um Mitternacht nach Hause und möchte, dass es bis um 19:00 geladen ist. Mit einer herkömmlichen Ladesäule würde das EV mit voller Leistung laden (siehe Abbildung 2) bis es voll ist. Die nötige Ladeenergie von 13 kWh würde dann 2 Fr. kosten und die Eigenverbrauchsrate wäre nur 56 %.

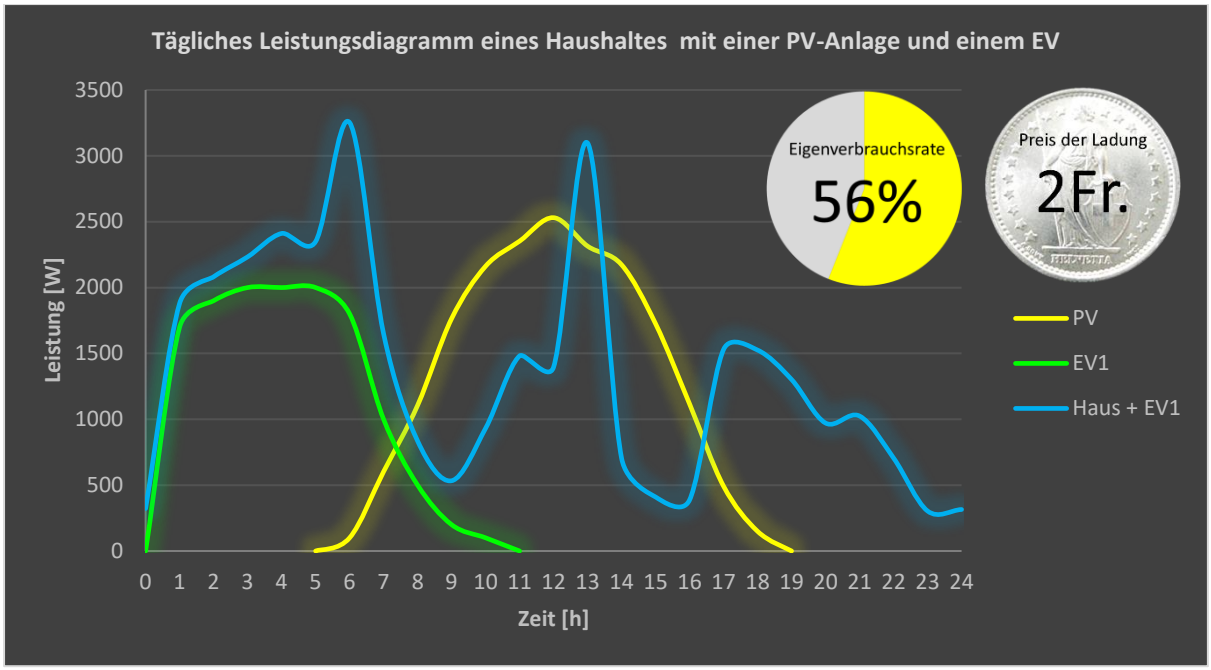


Abbildung 2:Tägliches Leistungsdiagramm eines Hauses mit EV

Man kann jedoch den Preis der Ladung senken, indem man die Leistung der Ladesäule reguliert. Da das EV erst um 19:00 bereit sein muss, sollte es nicht nur mit Nachtstrom, sondern hauptsächlich mit der überschüssigen Leistung der PV-Anlage geladen werden. Genau so soll das in dieser Arbeit zu entwickelnde System (Smart-Charge System) den Preis der Ladung optimieren. Dieselbe Ladeenergie (13 kWh) würde in diesem Fall nur noch 1.20 Fr. kosten, was eine Ersparnis von 40 % darstellt. Der Eigenverbrauch beträgt gute 93 %. Ideal wären es 100 %; diese können jedoch wegen dem Ladeprofil der Batterie nicht komplett erreicht werden. In der Abbildung 3 sieht man, wie das System die Ladesäule steuern würde.

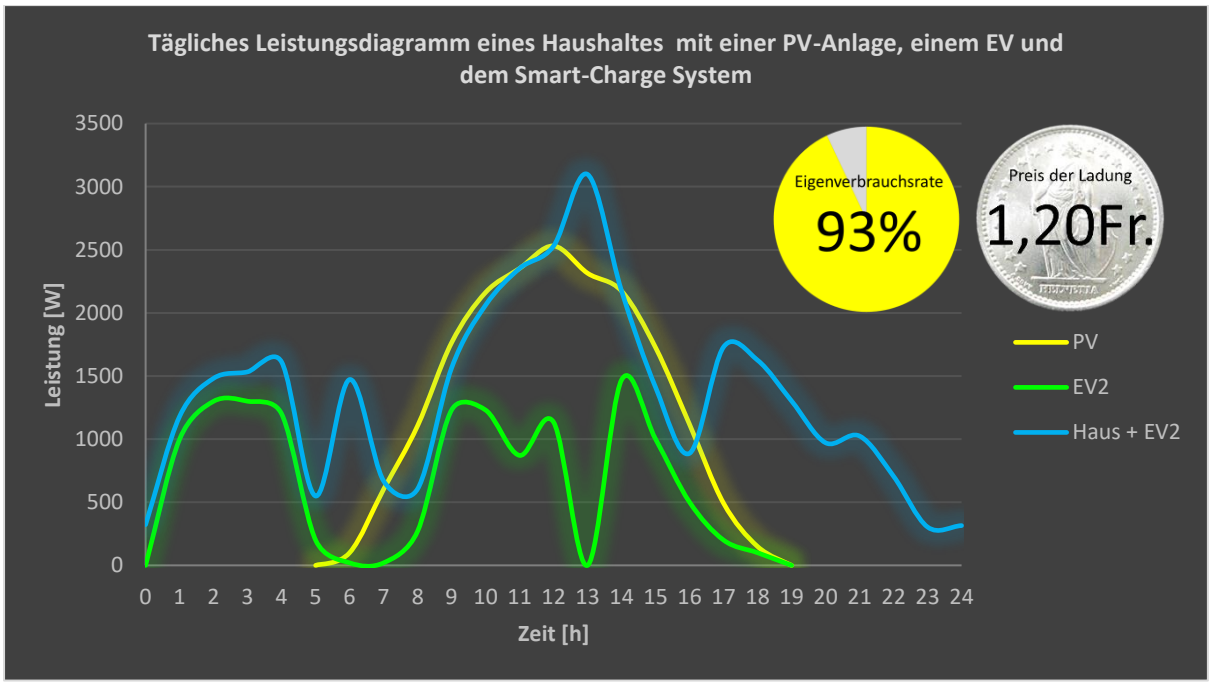


Abbildung 3: Tägliches Leistungsdiagramm eines Hauses mit EV und dem Smart-Charge System

Die Ladeleistung des EVs zu steuern ist also eine gute Lösung, um die Eigenverbrauchsrate zu vergrössern und somit den Preis der Stromrechnung zu senken. Dies ist möglich, da ein EV ein flexibler Verbraucher ist. Das heisst, dass man den Zeitpunkt seiner Ladung verschieben kann. Dafür gibt es aber zwei Voraussetzungen: die Ladesäule muss vernetzt sein und gesteuert werden können und das Auto muss sich zuhause am Stecker befinden. Das System kann die Eigenverbrauchsrate nur dann optimieren, wenn das Auto an den Zeiten mit PV-Überschuss, also tagsüber, angesteckt ist. Der Preis der Ladung hingegen kann auch mit der Bevorzugung des Nachtstroms gesenkt werden.

Unser Beispiel veranschaulicht gut die Vorteile einer optimierten Ladung des EVs. Es ist aber nicht sehr typisch und wird eher am Wochenende vorkommen, wenn das Auto tagsüber zuhause ist. Dies ist die beste Voraussetzung für die Optimierung. An Werktagen wird der Einwohner wahrscheinlich früh am Morgen mit seinem EV zur Arbeit fahren und kommt erst am Abend zurück. Aber auch hier soll das *Smart-Charge System* den Preis anhand des Nachtstroms senken. Der Besitzer wird z.B. sein EV um 20:00 anstecken, aber das System wird mit der Ladung abwarten, um erst ab 22:00 mit billigerem Nachtstrom zu laden.

Das *Smart-Charge System* wird also den Ladevorgang des EVs in jedem Fall optimieren können.

3 Schon existierende Systeme

Auf dem Markt gibt es schon einige Lösungen zur Optimierung des Ladevorgangs von EVs. Die Funktionalitäten dieser Systeme sind noch beschränkt. Hier werden davon zwei vorgestellt. Beide basieren auf derselben vernetzten Ladesäule (KEBA P30), die im nächsten Kapitel beschrieben wird.

3.1 BKW Home Energy

Das erste System ist das BKW Home Energy. Sein Herzstück ist der Solar-Log™, der die PV-Produktion sowie den Haushaltsverbrauch misst und die flexiblen Lasten steuert, unter anderem die KEBA-Ladesäule. Es gibt 3 Lademodi zur Auswahl: die Solarladung (der Ladestrom passt sich kontinuierlich der verfügbaren PV-Produktion an), die Kombiladung (das EV wird mit vorgegebenem Strom konstant von der PV-Anlage und vom Netz geladen) und die Schnellladung (lädt mit dem maximalen Ladestrom, unabhängig von der PV-Produktion). Bei der Installation des Systems können jedoch nur 2 der 3 Lademodi installiert werden (Solarladung oder Kombiladung, die Schnellladung ist standardmässig installiert). Um den Modus zu wählen, wird ein mechanischer Schalter neben der Ladesäule montiert. Über die mitgelieferte Home Energy App können der aktuelle Ladestrom sowie die Ladeenergie überwacht werden.

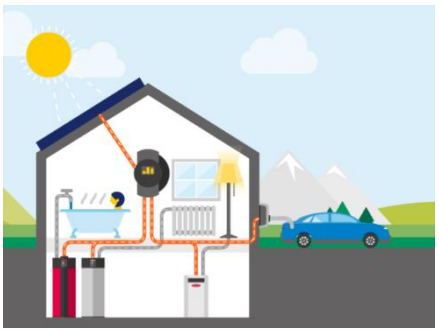


Abbildung 4: BKW Home Energy (https://www.bkw.ch/en/home/waerme-gebaeudetechnik/home-energy)

Der Preis des schlüsselfertigen Systems startet bei 3'200 Fr. für eine 11 kW Installation und geht bis zu 4'300 Fr. für eine 22 kW Installation. Falls noch nicht vorhanden, muss der Solar-Log™ für 1'700 Fr. zusätzlich installiert werden. (Richtofferte im Anhang 1)

3.2 SMARTFOX Car Charger

Das zweite System ist der SMARTFOX Car Charger, der eine Anpassung der KEBA-Ladesäule ist. Dieser wird vom SMARTFOX Reg gesteuert. Es stehen zwei Ladearten zur Auswahl: die Überschussladung (lädt nur mit der überschüssigen Leistung der PV-Anlage) und der Schnellladem odus (lädt mit voller Leistung). Die Ladeart kann über ein Online-Portal, eine App oder einen Schalter gewählt werden.

Für den SMARTFOX Car Charger muss mit einem Preis von 2'600 Fr. gerechnet werden. Dazu kommt der SMARTFOX Reg für ca. 800 Fr. Die Montage ist nicht inbegriffen. Rechnet man für diese 2'000 Fr., so sind die Kosten für diese Lösung in derselben Grössenordnung wie die vom BKW Home Energy.

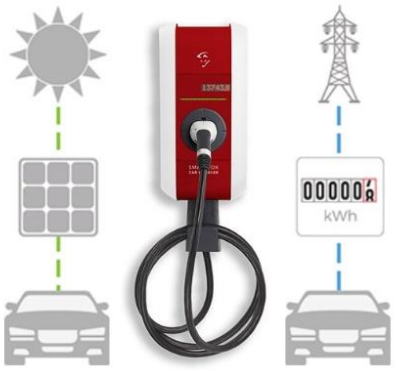


Abbildung 5: SMARTFOX Car Charger mit den zwei Ladearten (http://www.smartfox.at/car-charger)

4 EV-Ladesäulen

4.1 Allgemeines

Um ein EV zu laden, gibt es verschiedene Ladearten und Steckertypen, die in der Norm IEC 62196 beschrieben sind. Man unterscheidet hauptsächlich die Gleichstrom-Ladung (DC-Ladung) von der Wechselstrom-Ladung (AC-Ladung). Die Abbildung 6 zeigt eine Übersicht dieser zwei Ladearten.

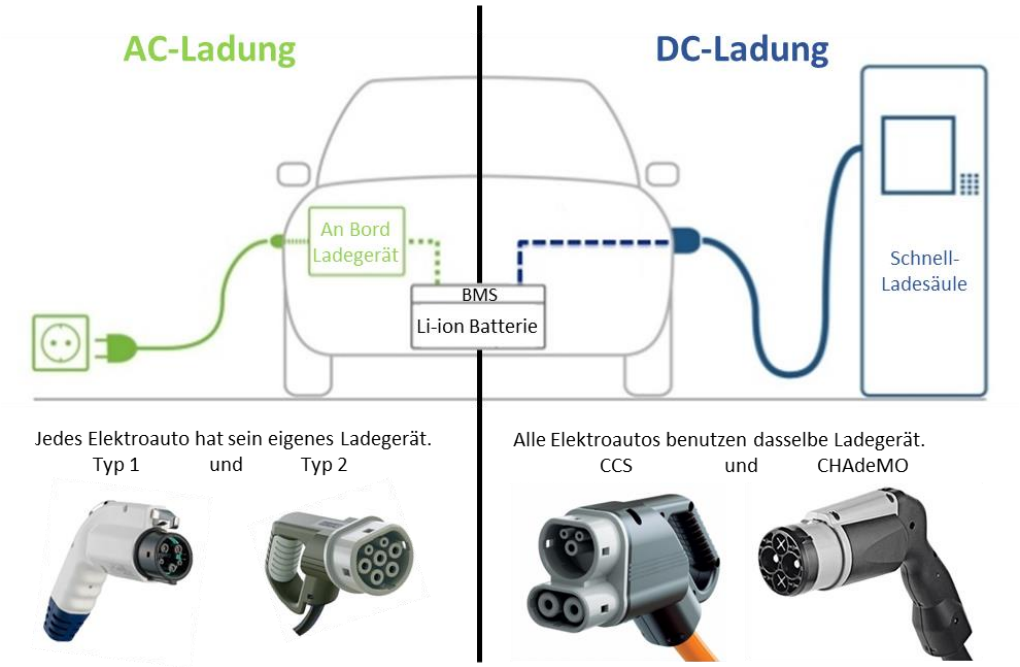


Abbildung 6: AC- und DC-Ladung in einem EV (https://www.slideshare.net/WiCleanCities/abb-presentation)

Die DC-Ladung wird für Schnellladungen bis zu 200 kW verwendet. Der AC-DC Konverter befindet sich in der Ladesäule, welche direkt mit der Batterie des EVs (300 V-500 V) verbunden ist. Eine High Level Kommunikation zwischen EV und Ladesäule ist zwingend, da die Ladesäule die Ladeleistung festlegt. Standardisierte Steckertypen sind das Combined Charging System (CCS) in Europa und der CHAdeMO in Japan.

Die AC-Ladung eignet sich für langsamere Ladungen mit weniger als 43 kW. Der AC-DC Konverter ist im EV integriert (an Bord Ladegerät). In diesem Fall legt das Auto selber die Ladeleistung fest. Hier reicht eine analoge Kommunikation zwischen Fahrzeug und Ladesäule aus. Standardisierte Steckertypen sind der Typ 1 Stecker in Japan und der Typ 2 Stecker in Europa. Ein Vorteil des Typ 2 Steckers ist, dass er 3-phasig laden kann.

Im Rahmen dieser Arbeit wird ausschliesslich die AC-Ladung des EVs gesteuert.

4.2 KEBA-Ladesäule

Wie bei den schon existierenden Ladeoptimierungs-Systemen wird auch für die Entwicklung des *Smart-Charge Systems* eine KEBA-Ladesäule verwendet. In unserem Fall ist es das Modell P30 c-serie mit fixem Kabel und einem Typ 2 Stecker. Die Ladesäule lädt das EV mit dreiphasigem Wechselstrom (AC) auf. Eine Stromüberwachung und eine DC-Fehlerstromerkennung sind integriert und können bei Problemen den Ladevorgang unterbrechen. Über eine Ethernet-Schnittstelle (RJ45) können via UDP Daten ausgetauscht werden. Die Ladesäule misst die Spannungen, die Ströme, die Leistungen und die Ladeenergie sowie die Präsenz eines EVs. Via UDP (User Datagram Protocol) gibt man der Ladesäule den maximal zulässigen Ladestrom und die Freigabe vor. Der Ladestrom kann zwischen 6 und 32 A reguliert werden, was bei 3-phasiger Ladung eine Leistung zwischen 4.14 kW und 22.08 kW ergibt. In der Ladesäule selber gibt es keine Energiewandlung. Sie schliesst und öffnet den Stromkreis und generiert das Signal der analogen Kommunikation mit dem EV.

Abbildung 7: KEBA-Ladesäule mit Typ 2 Stecker

4.3 Der Typ 2 Stecker

Der Typ 2 Stecker ist mit 7 Kontakten versehen: die drei Phasen L1, L2 und L3, der Neutralleiter N, der Erdschutzleiter PE und die zwei Kontakte für die Kommunikation: der PP (Proximity Pilot) und der CP (Control Pilot). Die KEBA-Ladesäule unterstützt jedoch keine digitale High Level Kommunikation mit dem EV. Das heisst, dass man keinen Zugriff zu den Informationen des BMS (Battery Management System) hat wie z.B. den aktuellen Ladestand der Batterie. Dies erschwert die Optimierung des Ladevorgangs, da sich das *Smart-Charge System* nur auf das eigene Modell des EVs verlassen kann. Die analoge Kommunikation wird von der KEBA-Ladesäule unterstützt. Diese funktioniert wie folgt:

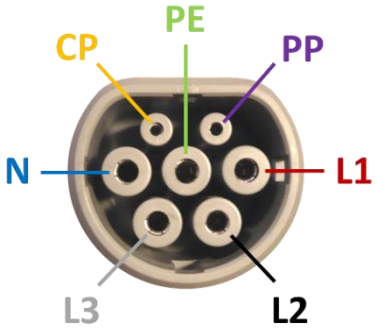


Abbildung 8: Kontakte eines Typ 2 Steckers von Phoenix Contact

Der maximale Ladestrom, bedingt durch den Querschnitt des Ladekabels, wird durch einen Widerstand zwischen PP und PE codiert. So entsprechen z.B. 220 Ω einem Maximalstrom von 32 A. Die Ladesäule misst diesen Widerstand und überwacht, dass der Maximalstrom des Kabels während der Ladung nicht überschritten wird.

Der CP hat eine doppelte Funktion. Einerseits teilt die Ladesäule den vorgegebenen Maximal-Ladestrom dem EV mit. Dazu wird ein PWM-Signal von 1 kHz und +/- 12 V über einen in Serie montierten 1 kΩ Widerstand zwischen CP und PE generiert. Die Pulsbreite entspricht dem vorgegebenen Maximal-Ladestrom. Eine Pulsbreite von z.B. 50 % gibt einen Maximal-Ladestrom von 30 A an. Andererseits gibt das EV seinen Zustand über den CP an, indem es einen variablen Widerstand in Serie mit einer Diode zwischen CP und PE setzt und somit einen Spannungsabfall der positiven PWM-Spannung bewirkt. So weiss die Ladesäule, ob ein EV angeschlossen ist und ob es zum Laden bereit ist.

5 Entwicklung Smart-Charge System

5.1 Funktionsprinzip

Das *Smart-Charge System* wird in einem Gebäude in Steg installiert, das schon eine PV-Anlage, eine steuerbare Ladesäule (von KEBA) und ein Gateway (von Misurio) besitzt. Das Gateway sammelt Daten von verschiedenen Analgen im Haus, wie z.B. den aktuellen Haushaltsverbrauch, die PV-Produktion, den Netzbezug usw. und schickt diese in die Cloud. Über eine Plattform kann das *Smart-Charge System* zu den relevanten Daten zugreifen, sowie die Ladesäule steuern.

Das zu entwickelnde System besteht aus einem Algorithmus, der den Ladevorgang optimiert, und einem Interface, das die Vorgaben des Nutzers erfasst.

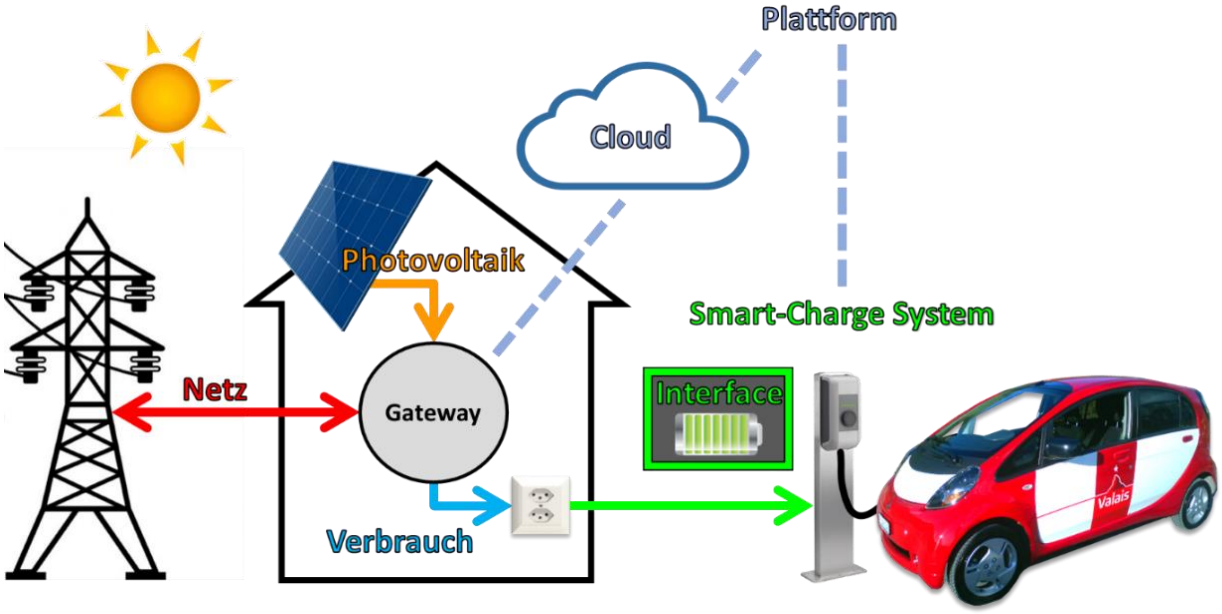


Abbildung 9: Funktionsprinzip Smart-Charge System

Übers Interface stehen 3 Lademodi zur Auswahl, die in der folgenden Tabelle erklärt werden.

Modus	Beschreibung	Manuelle Eingabe am Interface	Anzeige auf dem Interface
FastCharge	Das Auto wird so schnell wie möglich geladen, ohne die Maximal-leistung des Hausanschlusses zu überschreiten.	Maximale Ladeleistung [kW]	Ladeleistung [kW] Ladeenergie [kWh] Preis der Ladung [Fr.]
SolarCharge	Das Auto wird nur mit der verbleibenden Leistung der PV-Anlage geladen.	Maximale Ladeleistung [kW]	Ladeleistung [kW] Ladeenergie [kWh]
SmartCharge	Das Auto wird bis zu einer gewissen Zeit auf einen vorgegebenen Batteriestand geladen. Die Leistung der Säule wird nach einem Profil gesteuert, das den Preis der Ladung und somit den Eigenverbrauch optimiert.	Endzeit [hh:mm] Batteriestand verlangt [%] Batteriestand aktuell [%]	Ladeleistung [kW] Ladeenergie [kWh] Verhältnis Solar-/Ladeenergie [%] Batteriestand [%] Preis der Ladung [Fr.]

Tabelle 1: Lademodi des Smart-Charge Systems

Im Vergleich zum BKW Home Energy und zum SMARTFOX Car Charger bietet das *Smart-Charge System* einen zusätzlichen "SmartCharge" Modus. Dieser berechnet ein optimales Ladeprofil fürs EV und benötigt dafür die Wettervorhersagen von einem Meteoserver, eine Schätzung des voraussichtlichen Haushaltsverbrauchs und ein Modell des Ladeprofils des EVs. Das folgende Blockschema zeigt die Signale, die zwischen den einzelnen Komponenten ausgetauscht werden:

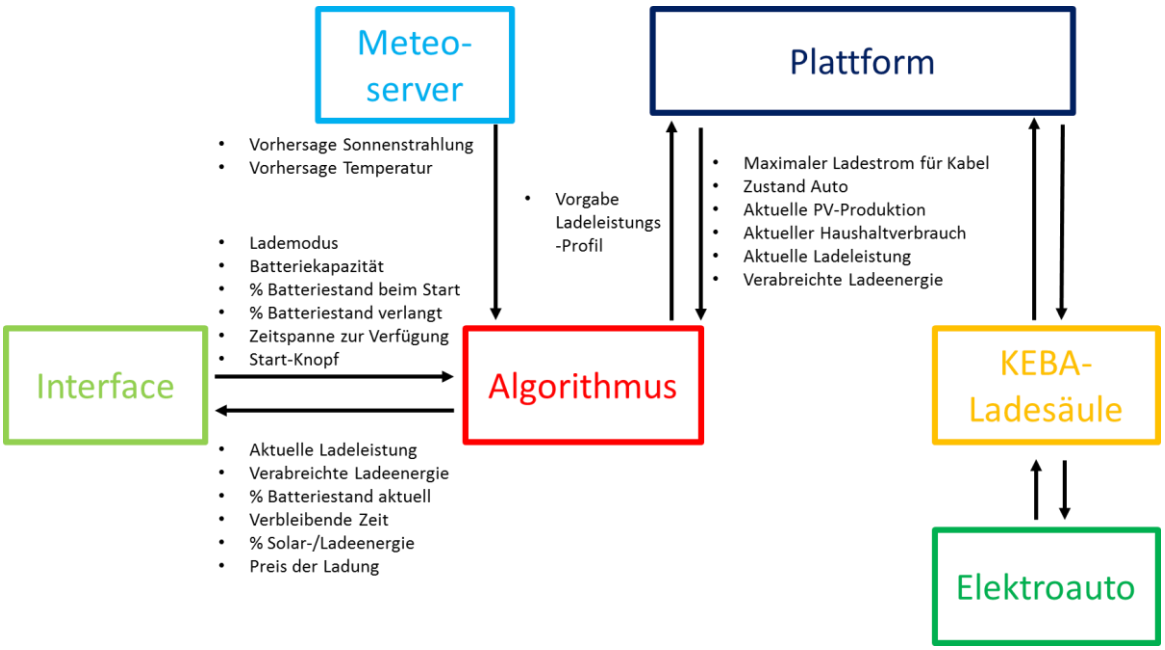


Abbildung 10: Blockschema Smart-Charge System

Der Integralteil des Pflichtenhefts des *Smart-Charge Systems* kann im Anhang 2 gelesen werden.

5.2 Test-Programm

Im ersten Teil der Arbeit wurde ein Testprogramm mit QT Creator entwickelt. Dieses kommuniziert mit der Plattform via HTTP. Die Plattform besitzt ein REST-Schnittstelle (Representational State Transfer), die mit GET oder PUT Methoden Daten austauscht. Die Benutzeroberfläche des Testprogramms ist in 6 Blöcken gegliedert, die in der Abbildung 11 dargestellt sind.

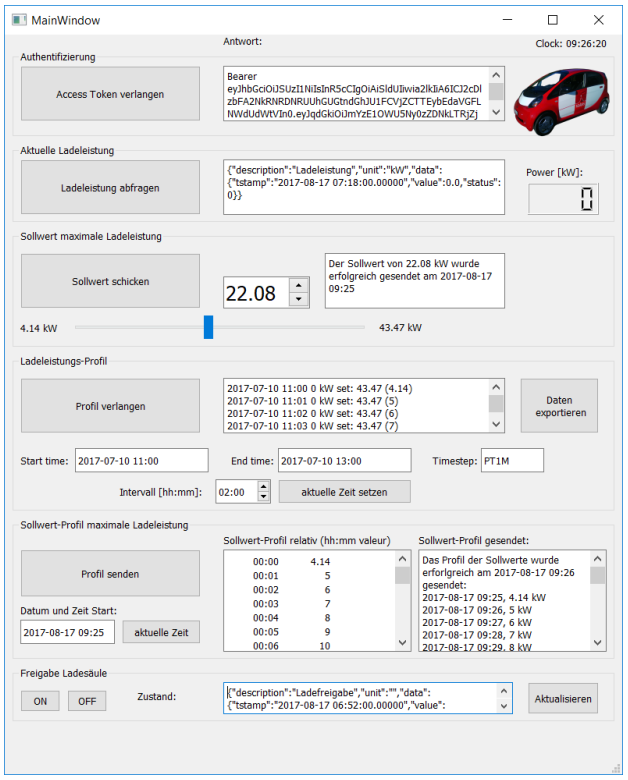


Abbildung 11: Testprogramm Benutzeroberfläche

Mit dem Block "Authentifizierung" identifiziert sich das Testprogramm als Client. Dazu wird eine GET Methode mit dem Benutzernamen und dem Passwort geschickt. Als Antwort schickt die Plattform einen Zugangsschlüssel (Access Token), der während 5 Minuten gültig ist. Dieser muss zusammen mit jeder Anfrage als Authentifizierungs-Header mitgeschickt werden.

Der Block "Aktuelle Ladeleistung" fragt die momentane Ladeleistung ab. Diese wird anfangs jeder Minute von der Plattform aktualisiert, indem sie die Momentanleistung der KEBA-Ladesäule ausliest.

Mit dem Block "Sollwert maximale Ladeleistung" kann die Vorgabe der maximalen Ladeleistung an die Ladesäule geschickt werden. 15 s vergehen ungefähr, bis die Ladesäule sich dieser Vorgabe anpasst. Die Leistung kann theoretisch zwischen 4.14 kW und 43.47 kW vorgegeben werden, jedoch kann die Hardware der Ladesäule nicht mehr als 22.08 kW (32 A) liefern.

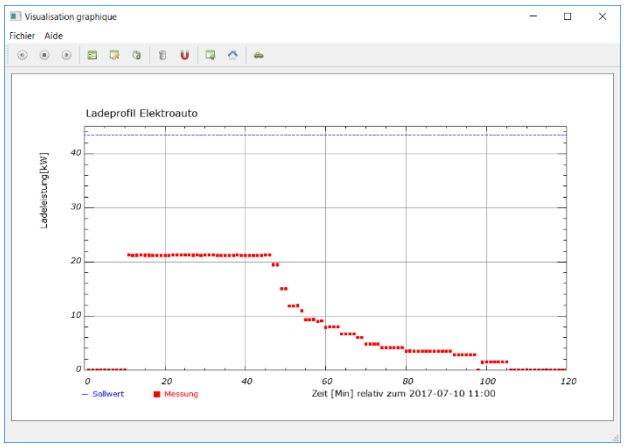


Abbildung 12: Visualisierung des Ladeprofils

Der Block "Ladeleistungs-Profil" ermöglicht, eine Serie von Messungen der Ladeleistung und der Ladevorgabe abzufragen. Die Startzeit und Endzeit sowie der Zeitschritt kann angegeben werden, oder mit dem Knopf "aktuelle Zeit setzen" und einem Intervall automatisch gesetzt werden. Mit dem Knopf "Daten exportieren" werden die Daten in einem Teil des Speichers geschrieben und von einem schon existierenden Programm als Grafik dargestellt (Abbildung 12).

Mit dem Block "Sollwert-Profil maximale Ladeleistung" kann ein beliebiges Profil für die maximale Ladeleistung gesendet werden. Das Profil kann z.B. in einer Excel-Tabelle geschrieben werden. In der ersten Kolonne muss die Relativzeit stehen und in der zweiten der Sollwert in kW. Dann können die zwei Kolonnen kopiert und im Feld "Sollwert-Profil relativ" eingefügt werden. Das Startdatum muss noch angegeben werden, bevor das Profil gesendet wird.

Im Block "Freigabe Ladesäule" können die Ladesäule ein- oder ausgeschaltet und der aktuelle Zustand visualisiert werden.

5.3 Messungen in Steg

Mit dem Testprogramm wurden bei der winsun AG in Steg, welche mit der KEBA-Ladesäule und dem Gateway ausgerüstet sind, Ladeprofile von verschiedenen EVs gemessen. Es wurden 3 Messpunkte aufgenommen:

- über die Plattform wurden die Angaben der KEBA-Ladesäule (Leistung und Energie) erfasst
- mit einem Power & Energy Logger der HEI Sitten (PEL 103, CHAUVIN ARNOUX, Serial: 154389, CR01/9529.05) wurden die Leistungen P, Q und S sowie der Power factor und die Energie gemessen. Der PEL wurde im Elektroschrank beim Start der Leitung zur Ladesäule montiert.
- vom Display des EVs wurde im Minutentakt der Batteriestand in % abgelesen.

Abbildung 13: Messungen mit dem PEL im Elektroschrank in Steg

Alle erhaltenen Daten wurden in einer Excel Datei gespeichert, um die Grafiken zu generieren.

Es wurden 5 Firmenwagen der winsun AG gemessen: 1 Tesla Model S 70D, 3 Renault ZOEs mit 22 kWh Batterie (Zoe Nr. 1, 2 und 4) und ein neuer Renault ZOE mit 41 kWh Batterie (Zoe Nr. 3).


Abbildung 14: Zwei Renault ZOEs in Steg. Einer davon ist am Laden

Als erstes wurde getestet, ob das EV dem Sollwert der maximalen Ladeleistung folgt (Sollwert KEBA), indem ein ansteigendes Profil geschickt wurde. Es wurde bemerkt, dass der Tesla den Sollwert bis auf eine Marge von in etwa 0.8 kW treu verfolgt, und das bis zu seiner Maximal-Ladeleistung von 11 kW (16 A). Die Messwerte P und E der KEBA-Ladesäule entsprechen auch denen des PELs. Es gibt jedoch ab und zu kleine Verzögerungen, da die Plattform die Daten aktualisieren muss. Im Fall des Teslas stellt man fest, dass er einen sehr guten Power factor hat, da er sehr wenig Blindleistung produziert (kapazitives Verhalten). In den Grafiken wird einfachheitshalber immer der Absolutwert der Blindleistung dargestellt.

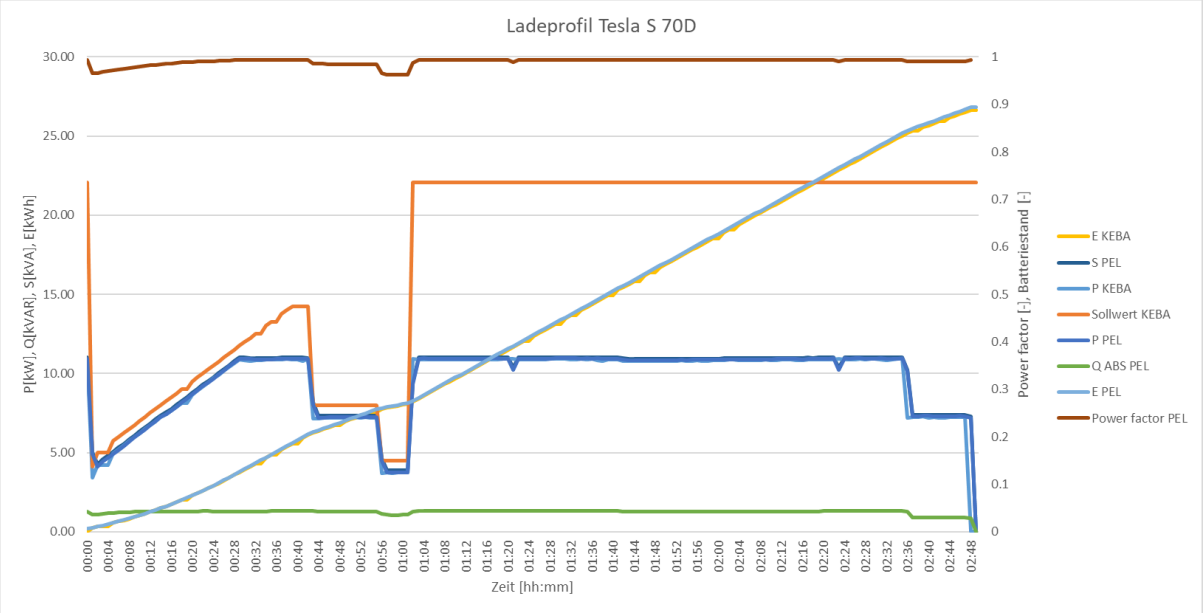


Abbildung 15: Ladeprofil Tesla S 70D. Der Power factor ist sehr gut

Als zweites EV wurde der Zoe 1 (22kWh Batterie) gemessen. Es wurde bemerkt, dass der Renault ZOE bei kleinen Leistungen eine viel grössere Marge als bei grossen Leistungen gegenüber dem Sollwert aufweist. Dies ist besonders beim Rampenprofil am Anfang der Messung ersichtlich. Das liegt daran, dass der ZOE viel mehr Blindleistung produziert als der Tesla. Der Power factor ist daher kleiner und die Scheinleistung grösser. Die KEBA-Ladesäule begrenzt nicht die Wirkleistung P, sondern die Scheinleistung S. Deswegen gibt es beim ZOE so eine grosse Marge zwischen dem Sollwert und der Ladeleistung P.

Nach 57 Minuten folgt der ZOE nicht mehr dem Sollwert, da seine Batterie fast voll ist und er nicht mehr mit voller Leistung laden kann.

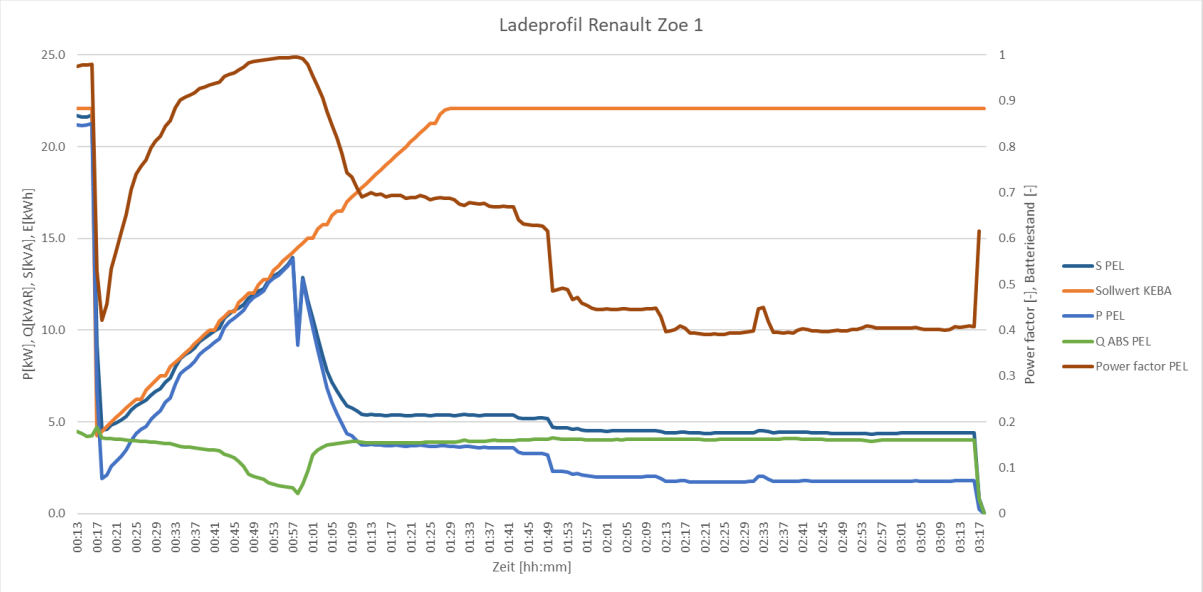


Abbildung 16: Ladeprofil Renault Zoe 1. Der ZOE produziert mehr Blindleistung als der Tesla

Beim Renault Zoe 2 (auch 22 kWh Batterie) ist ein Problem aufgetreten. Wenn der Sollwert der maximalen Ladeleistung unter 8 kW gesetzt wird, lädt der Zoe 2 überhaupt nicht. Zwischen 8 kW und 9 kW lädt er während 7 s, danach bricht er die Ladung ab und wartet 16 s bevor er einen neuen Ladeversuch startet. Um richtig zu laden, muss der Sollwert über 9 kW liegen. Dieses Problem wurde auch beim Zoe 4 festgestellt. Die Abbildung 17 zeigt eine Messung dieses Problems mit Sekundenauflösung.

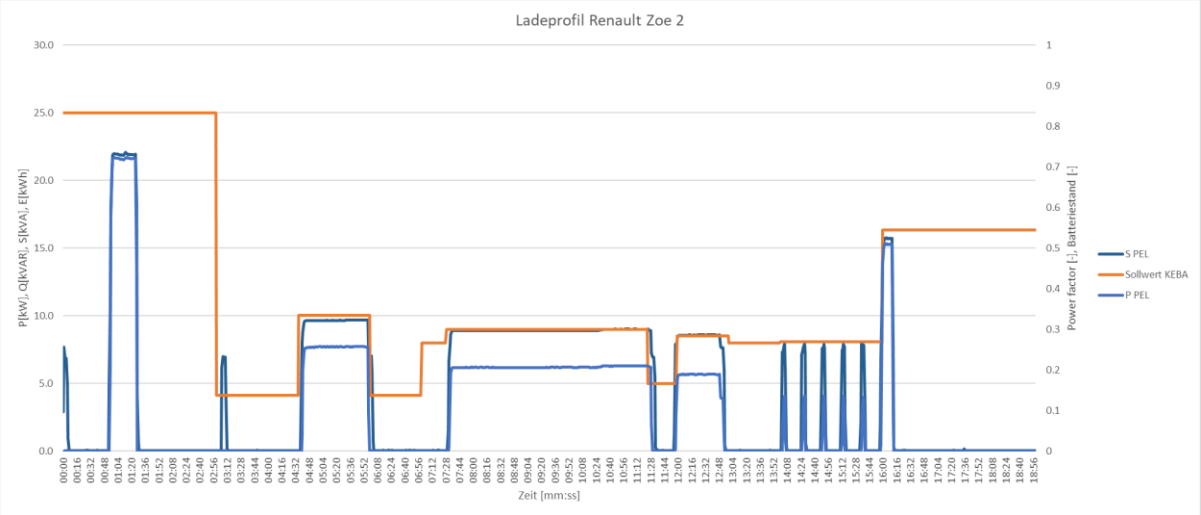


Abbildung 17: Ladeprofil Renault Zoe 2. Bei kleinen Leistungen lädt der Zoe 2 nicht richtig

Komischerweise weist der Zoe 1, der auch ein älteres Modell ist, dieses Problem nicht auf. Der Zoe 3, welcher das neue Modell ist, weist auch keine Probleme auf. Eine Koinzidenz wurde jedoch gefunden: am Ende der Ladung produzieren die zwei problemhaften ZOEs (Zoe 2 und Zoe 4) mehr Blindleistung (6.7 kVAR) als die zwei anderen Modelle (Zoe 1 und Zoe 3). Wahrscheinlich besitzen der Zoe 1 und Zoe 3 eine neuere Version des an Bord Ladegeräts. Das Problem könnte von der Kommunikation zwischen der Ladesäule und dem Ladegerät herkommen.

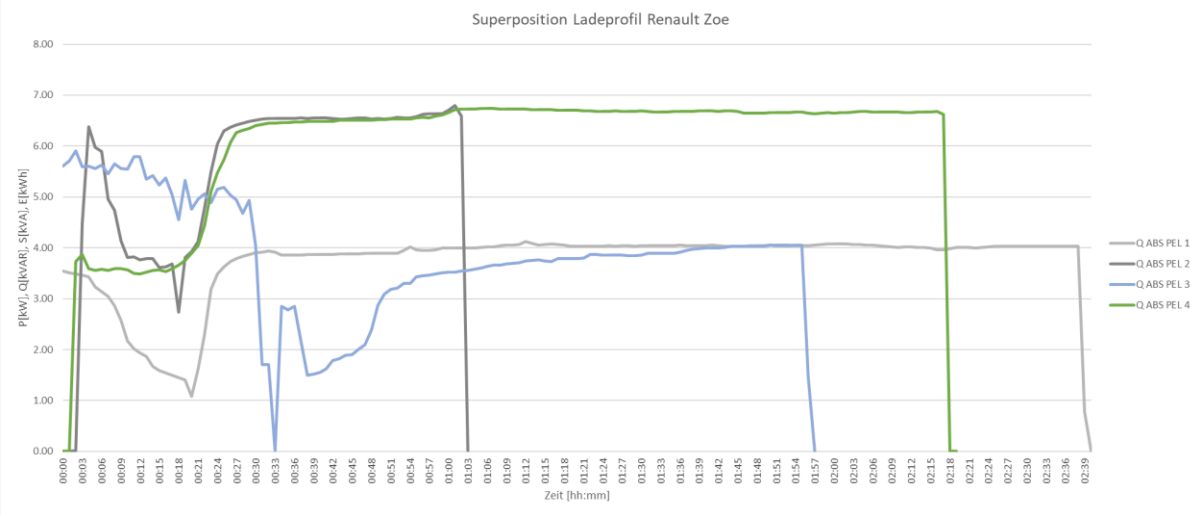


Abbildung 18: Blindleistung der Zoes 1-4 am Ende der Ladung

Bei der Messung des Zoe 4 wurde die Ladeleistung nicht begrenzt. In der Abbildung 19 kann man den Verlauf der Leistungen P, Q und S sowie den Power factor und den Batteriestand sehen. Man bemerkt, dass der Power factor während des letzten Teiles der Ladung sehr klein ist.

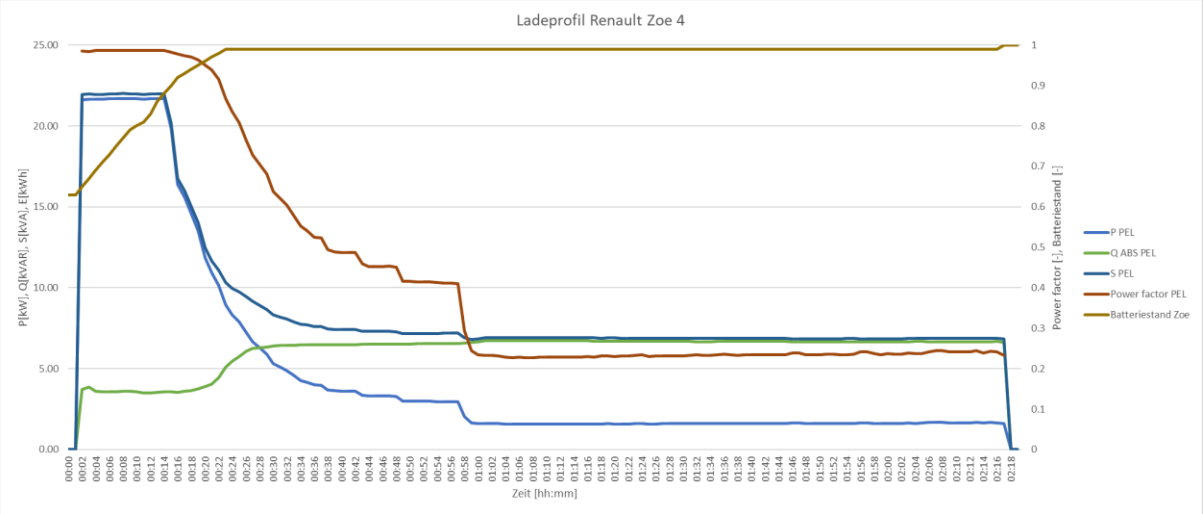


Abbildung 19: Messung einer Ladung des Zoe 4 ohne Leistungsbegrenzung

5.4 Modellierung Renault ZOE 22 kWh

Anhand der Messungen in Steg konnte gezeigt werden, dass ein EV ein maximales Ladeprofil hat, das man bei der Entwicklung des Smart-Charge Systems modellieren muss. Es wurde beschlossen, das Modell für einen Renault ZOE mit einer 22 kWh Batterie zu berechnen, da dies das meistvorhandene Firmenauto der winsun AG ist. Dazu wurde eine ganze Ladung des Zoe 4 durchgeführt (von 0 % bis 100 %), die in der Abbildung 20 dargestellt ist.

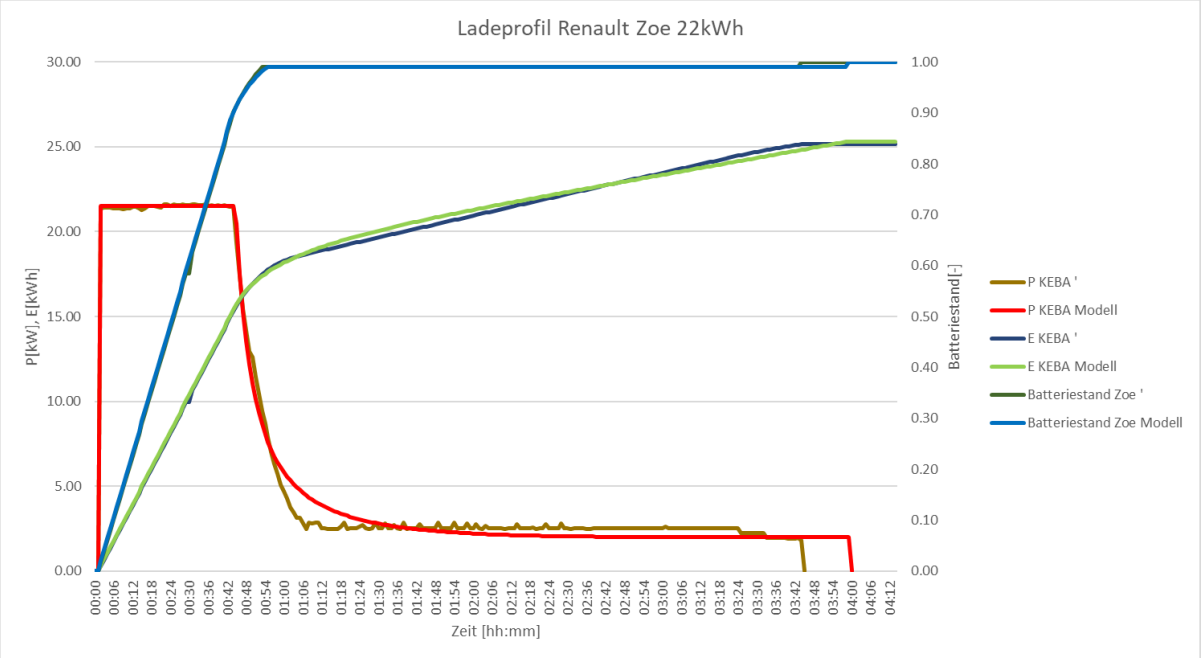


Abbildung 20: Modell und Messung einer vollen Ladung des Renault ZOEs

Die Ladeleistung wird folgendermassen modelliert: von 0 % bis 90 % des Batteriestandes, was 15.3 kWh entspricht, lädt der ZOE mit voller Leistung (21.5 kW). Danach reduziert er die Leistung, um die Batterie zu schonen. Das Modell berechnet die Ladeleistung von 90 % bis 100 % des Batteriestandes mit folgender Funktion:

$$P = 2\ kW + 19.5\ kW \times (1 - \frac{E - 15.3\ kWh}{10\ kWh})^5$$

P ist die Ladeleistung in kW und E die Ladeenergie in kWh. Die Funktion startet bei 21.5 kW und endet bei 2 kW.

Abbildung 21: Anzeige des Batteriestands im Renault ZOE

Der Batteriestand, der auf dem Renault ZOE angezeigt wird, ist nicht während der ganzen Ladung proportional zur Ladeenergie. Das Modell unterscheidet 3 Bereiche: zwischen 0 kWh und 15.3 kWh Ladeenergie wird der Batteriestand proportional zu einer 17 kWh Batterie berechnet. B ist der Batteriestand in %. Die Funktion ergibt für E zwischen 0 kWh und 15.3 kWh einen Batteriestand von 0 % bis 90 %:

$$B = \frac{E}{17\ kWh} \times 100\ \%$$

Zwischen 15.3 kWh und 17.46 kWh wird jedes zusätzliche % des Batteriestandes proportional zu einer 24 kWh Batterie berechnet. Die Funktion ergibt für E zwischen 15.3 und 17.46 kWh einen Batteriestand von 90 bis 99 %:

$$B = \frac{E - 15.3\ kWh}{24\ kWh} \times 100\ \% + 90\%$$

Das letzte % wird proportional zu einer 784 kWh Batterie berechnet, das heisst, dass man noch 7.84 kWh ins EV stecken muss, um das letzte % zu laden.

Eine volle Batterie enthält dementsprechend 25.3 kWh. Dieses Modell nähert sich circa der reellen Messung, wobei zu erwähnen ist, dass dasselbe EV nicht immer gleich lädt. Das kann von vielen Faktoren abhängen wie z.B. die Temperatur und das Alter der Batterie. Daher reicht das hier vorgestellte Modell des Renault ZOEs vollkommen aus.

Im Programm des *Smart-Charge Systems* wurde das Modell des ZOEs in einer eigenen Klasse namens ElectricVehicle in c++ geschrieben. Der Code befindet sich im Anhang 5. Die folgende Abbildung zeigt die verschiedenen Methoden, die implementiert wurden:

Abbildung 22: Methoden der ElectricVehicle Klasse (Bild: http://www.goingelectric.de/elektroautos/renault-zoe-2013)

5.5 Testoptimierung mit Excel

Der Lademodus "Smart" des *Smart-Charge Systems* verwendet einen evolutionären Algorithmus. Dieser sucht durch gezieltes Ausprobieren verschiedener Lösungen, ein vorteilhaftes Ladeprofil fürs EV. Es kann sein, dass es nicht die bestmögliche Lösung ist, aber es wird eine gute sein. Die Lösungen werden anhand einer Kostenfunktion bewertet. Je kleiner der Wert der Kostenfunktion ist, desto besser ist die Lösung.

Um diese zu testen wurde das System mit einem Excel Solver programmiert. Die Kostenfunktion setzt sich aus vier Werten zusammen, die einzeln mit einem Faktor (g1-g4) gewichtet werden:

$$Kostenfunktion = g1 \times Preis\ der\ Ladung + g2 \times bezogener\ Netzstrom + g3 \times Ladefrist + g4 \times Schnelligkeit$$

Die Parameter des Systems und die Gewichtungsfaktoren können angepasst werden. In der Abbildung 23 ist ein Beispiel einer Ladung einer 90 kWh Batterie gezeigt. Der Preis der Ladung hat eine grosse Gewichtung (g1 = 50 %). Dies bewirkt, dass der grösstmögliche Teil der Energie vom Überschuss der PV-Anlage kommt und der Rest möglichst mit Nachtstrom geladen wird. Die Ladefrist wurde bis 16 Uhr vorgegeben und auch stark gewichtet (g3 = 35 %). Damit wird die Ladung spätestens um 16 Uhr fertig sein. Die Schnelligkeit ist auch gewichtet (g4 = 15 %). Dies bewirkt, dass nicht erst im letzten Moment geladen wird, sondern sobald günstige Energie vorhanden ist. In der Nacht wird z.B. direkt um 1, 2 und 3 Uhr geladen, und nicht erst um 4, 5 oder 6 Uhr.

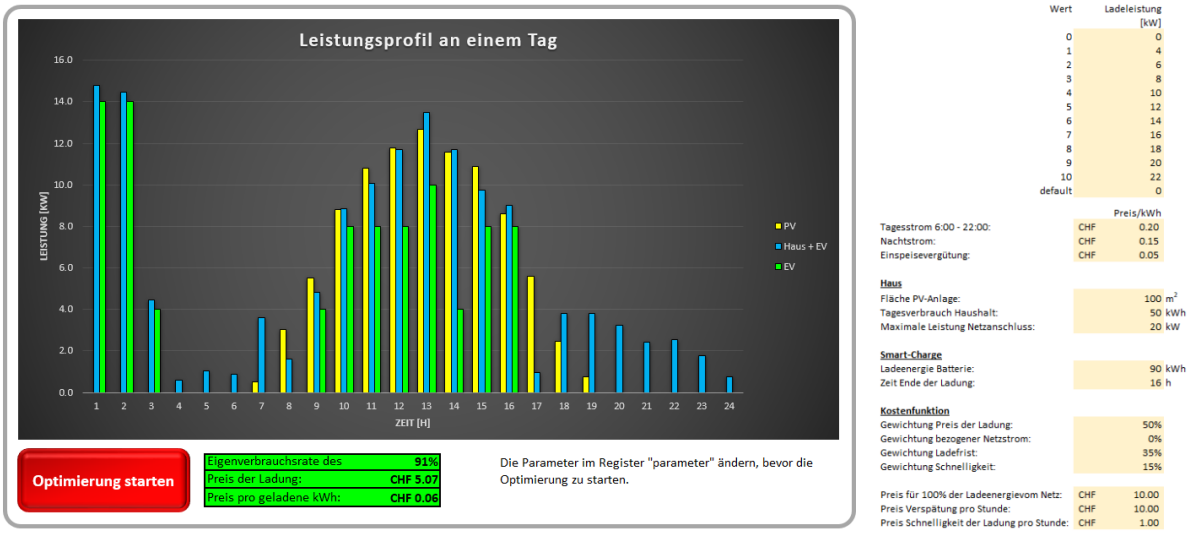


Abbildung 23: Excel Solver mit dem evolutionären Algorithmus und den Parametern

In diesem Excel Solver wurde die Ladekurve des EV Modells einfachheitshalber nicht berücksichtigt. In Wirklichkeit könnte man das EV nicht genau so laden. Es konnte jedoch mit diesem Beispiel gezeigt werden, dass sich ein evolutionärer Algorithmus für die Optimierung im *Smart-Charge System* gut eignet.

5.6 Smart-Charge Programm

Beim Start des Programmes wird standardmässig der Modus "Fast" vorgeschlagen. Falls gewünscht, kann die maximale Ladleistung mit einem Potentiometer von 22 kW bis auf 4 kW herabgesetzt werden. Durch Drücken auf den Startknopf wird der Sollwert der Ladeleistung geschickt und die Ladesäule freigegeben. Wie in allen Modi werden oben rechts in einem Kasten die relevanten Informationen (gemessene Ladeleistung und Ladeenergie, Preis der Ladung und Prozentsatz Solarenergie) angezeigt und im Minutentakt aktualisiert.

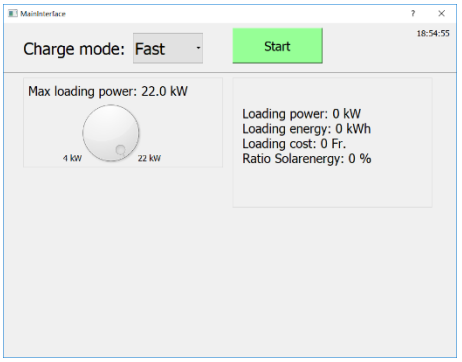


Abbildung 24: Benutzeroberfläche FastCharge

Im "Solar" Modus verschwindet das Potentiometer, da der Sollwert der Ladeleistung jede Minute durch das System berechnet wird. Dieser entspricht dem Leistungsüberschuss der PV-Anlage.

Wählt man den "Smart" Modus, so erscheinen drei neue Felder. Im ersten können mit Slidebars der aktuelle sowie der verlangte Batteriestand angegeben werden. Der zugelassene Bereich liegt zwischen 0 % und 99 %. Man kann den verlangten Batteriestand bis auf 100 % setzen, indem man die Checkbox "Trip" anklickt. Dabei erscheint folgende Infomeldung:

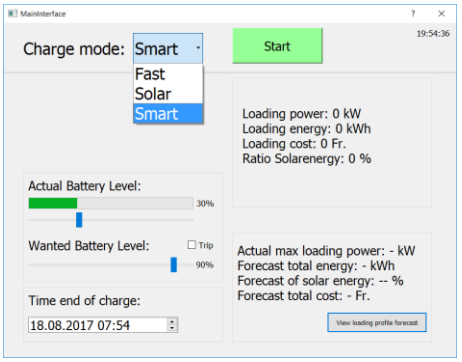


Abbildung 25: Benutzeroberfläche SmartCharge

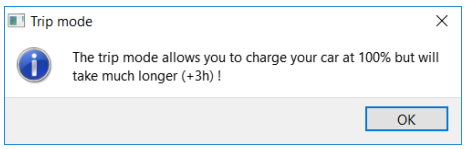


Abbildung 26: Infomeldung Trip mode

Die Ladung dauert laut dem EV Modell 3 Stunden länger, um das letzte % der Batterie zu laden. Deswegen sollte man den Trip Modus nur in Ausnahmefällen wählen.

Im untersten Kasten kann die Ladefrist eingegeben werden. Sollte diese jedoch nach den Berechnungen des EV Modells unrealistisch sein, so erscheint beim Druck auf den Startknopf folgende Warnmeldung:

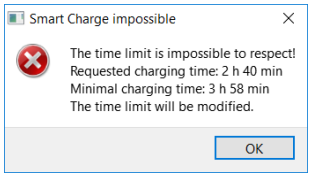


Abbildung 27: Warnmeldung unrealistische Ladefrist

Ist alles in Ordnung, werden diese Angaben im Speicher geschrieben und ein schon vorhandenes separates Programm mit einem genetischen Algorithmus (gehört zu den evolutionären Algorithmen) im Hintergrund gestartet. Dieses soll dann die Angaben vom Speicher entnehmen, die Optimierung durchführen und das bestgefundene Ladeprofil zurück in den Speicher schreiben. Danach wird das Hauptprogramm das Ladeprofil lesen und es der Plattform senden. Im letzten Kasten werden anschliessend der aktuelle Sollwert der Ladeleistung und die Vorhersagen der Ladeenergie, des Ladepreises und des Anteils an Solarstrom angezeigt. Zudem kann über einen Knopf das geplante Ladeprofil visualisiert werden. Alle 5 Minuten wird ein aktualisiertes Ladeprofil vom Algorithmus berechnet, um sich unter anderem den wechselnden Wettervorhersagen anzupassen. Die folgende Abbildung zeigt die Signale, die für den genetischen Algorithmus verwendet werden:

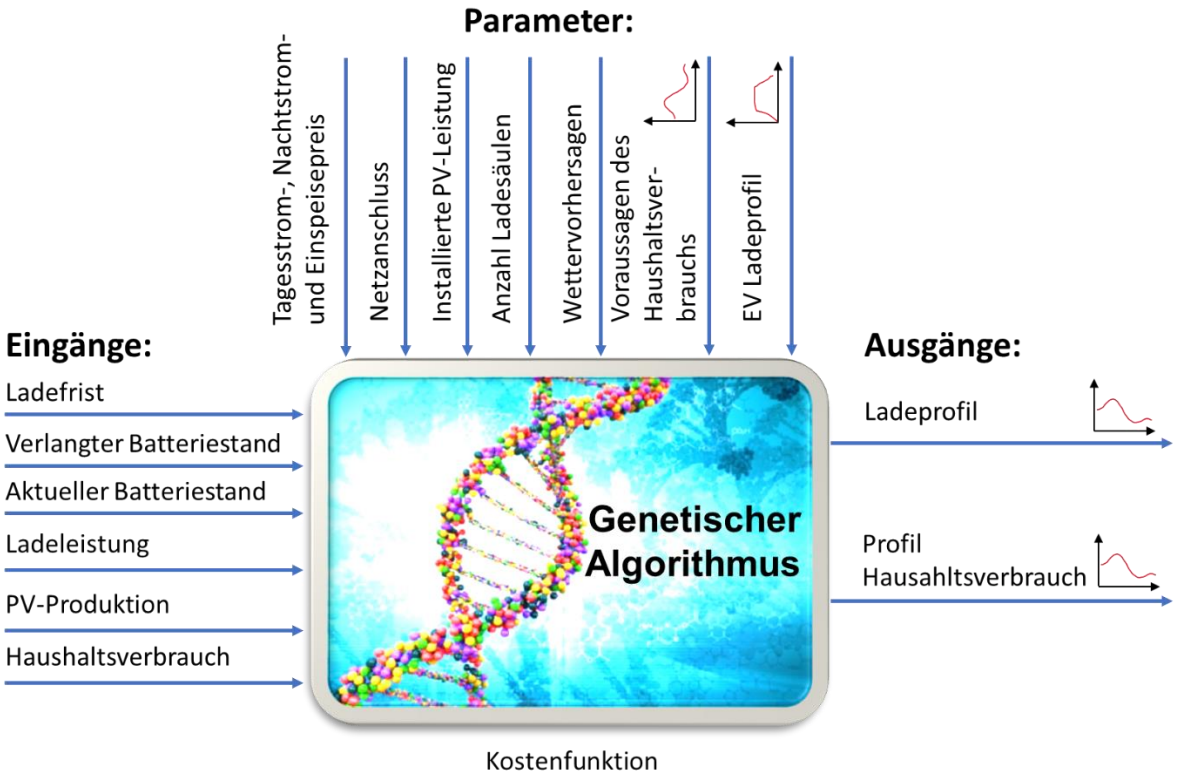


Tabelle 2: Signale für genetischer Algorithmus

Der "Smart" Modus ist an diesem Stand der Arbeit noch nicht vollständig entwickelt und wird zu einem späteren Zeitpunkt implementiert.

Das Programm des *Smart-Charge Systems* wurde mit QT Creator in c++ geschrieben. Es besteht hauptsächlich aus vier Klassen: Die Klasse MainInterface schafft die Benutzeroberfläche und führt die Berechnungen für die Lademodi durch, die Klasse RestClient kommuniziert mit der Plattform über HTTP, die Klasse ElectricVehicle modelliert das EV und die Klasse ThreadUpdateValues aktualisiert die angezeigten Werte gleichzeitig wie das Hauptprogramm. Der Code kann im Anhang 5 eingesehen werden.

6 Schlussfolgerung

In dieser Arbeit wurden die Schwierigkeiten festgestellt, die den Fortschritt von Optimierungssystemen für EV-Ladungen bremsen. Die verwendete KEBA-Ladesäule ermöglicht keinen Datenaustausch mit dem EV, was ein erster Schritt für eine solide Optimierung wäre.

Trotzdem wurde in dieser Arbeit ein Renault ZOE 22 kWh anhand von Messungen modelliert und ein Optimierungssystem entwickelt. Um daraus ein vermarktbares Produkt zu schaffen, können in einer Weiterentwicklung die Benutzeroberfläche verbessert und der "Smart" Modus vollständig implementiert werden. Das System muss noch in Realbedingungen getestet werden, was in dieser Arbeit nicht durchgeführt werden konnte. Die Modellierung des EVs müsste an jedes Automodell anpassbar sein, damit man mit diesem Produkt einen breiten Markt erreichen kann.

Mit der Entwicklung der Elektromobilität werden solche Optimierungssysteme für Elektroautos in Zukunft sicherlich eine grosse Herausforderung für die Ingenieurwelt darstellen.

7 Quellenverzeichnis

1) BUNDESAMT FÜR STATISTIK, Strassenfahrzeuge – Bestand, Motorisierungsgrad [online]. 2016. https://www.bfs.admin.ch/bfs/de/home/statistiken/mobilitaet-verkehr/verkehrsinfrastruktur-fahrzeuge/fahrzeuge/strassenfahrzeuge-bestand-motorisierungsgrad.html (besucht am 15.08.2017)

2) SMARTFOX, Car Charger [online]. 2017. http://www.smartfox.at/car-charger.html (besucht am 15.08.2017)

3) BKW, Home Energy Ladestation [online]. 2017. https://services.bkw.ch/shop/products/17.007 (besucht am 15.08.2017)

4) CHARIN, Design Guide for Combined Charging System. 2015

5) CHARIN, Combined Charging System 1.0 Specification - CCS 1.0. 2017

6) KEBA, Electric Mobility [online]. 2017 http://www.keba.com/en/emobility/emobility (besucht am 15.08.2015)

8 Datum und Unterschrift

Sitten, den 18.08.2017

Baptiste Cavin

9 Anhang

Anhang 1: Richtofferte BKW Home Energy

Anhang 2: Pflichtenheft Bachelorarbeit

Anhang 3: KeContact P30 Datenblatt

Anhang 4: Testprogramm RestCommunication QT c++ Code

Anhang 5: Smart-Charge Programm SmartChargeSystem QT c++ Code

Anhang 1:

Richtofferte BKW

Home Energy

BKW Energie AG
Viktoriaplatz 2
3013 Bern

Haben Sie Fragen?
0800 121 128
homeenergy@bkw.ch

Datum: 21.07.2017
Referenz: 14068

Unverbindliche Richtofferte für Ihre persönliche BKW Home Energy Lösung

Herzlichen Dank Herr CAvin!

Mit Ihrem Interesse an einer persönlichen, modularen Energielösung zeigen Sie, dass Ihnen eine umweltschonende Stromproduktion am Herzen liegt. Mit Home Energy produzieren Sie nicht nur Ihren eigenen Solarstrom, sondern optimieren auch Ihren Eigenverbrauch mit unserer intelligenten Steuerung: Sie speichern die Sonnenenergie und nutzen sie dann, wenn Sie sie brauchen. Mit Home Energy der Zukunft entgegen!

Unsere Richtofferte für das Rundum-sorglos-Paket * CHF 4'900.-

☑	Kostenlos nehmen wir in einem persönlichen Gespräch Ihre Energiebedürfnisse auf und erfassen die notwendigen Daten zur Erstellung einer verbindlichen Offerte. Zusätzlich zeigen wir Ihnen weitere Energieoptimierungsmöglichkeiten für Ihr Zuhause auf.
👥	Nach Vereinbarung montieren wir sämtliche Bestandteile Ihrer BKW Home Energy Lösung in Ihrem Eigenheim. Inbegriffen sind die notwendigen Bau- und Elektroinstallationen mit Gerüst und Sicherheitsvorkehrungen für Ihre Fotovoltaikanlage.
✎	Wir kümmern uns um die gesamte Administration mit den erforderlichen Bewilligungen, Anmeldungen, Fördergeldanträgen (Einmalvergütung, BKW Aktion Warmwasser-Wärmepumpen) sowie die Planung und Koordination der Beteiligten.
◔	Nach der Installation wird Ihre Anlage gründlich getestet und Sie erhalten ein vollständiges Anlagendossier von uns. Zusätzlich machen wir die Energieflüsse und den Eigenverbrauch Ihrer BKW Home Energy Lösung sichtbar, die Sie online per Home Energy App oder Web jederzeit überprüfen können.
👥	Wir sind auch nach der Montage für Sie da! Ihre Anlage ist durch die Garantien der BKW Energie AG, Ihrer Partner und der Komponentenhersteller geschützt. Jederzeit steht Ihnen der Kundenservice der BKW Energie AG zur Seite.

* Die Preisangabe versteht sich als unverbindliche Richtofferte inkl. MwSt. Eine verbindliche Offerte erstellen wir für Sie gerne nach der Vor-Ort-Bestandsaufnahme.

Hinweise

Nicht enthalten sind allfällige indirekte Kosten von Dritten oder Anschlussgebühren anderer Elektrizitätswerke, sowie u. U. notwendige Modernisierungen der Elektro-, Heizungs- und Wasserinstallation. Die Eignung zur Einbindung vorhandener Komponenten muss ggf. vor Ort geprüft werden. Voraussetzung für die Online-Visualisierung und Überwachung der Anlage ist ein Internetanschluss, der von Ihnen gestellt werden muss.

Zertifizierungen der BKW und ihrer Partner:

SWISSOLAR MEMBER

WPSYSTEMMODUL
EFFIZIENTE WÄRMEPUMPEN MIT SYSTEM

Ihre BKW Home Energy Komponenten - im Detail

BKW Home Energy bietet dank der intelligenten Steuerung viel mehr als seine Komponenten. Um den optimalen Betrieb des Systems zu garantieren, setzen wir auf Premium-Komponenten. Das rechnet sich für Sie über eine lange Lebensdauer und eine zuverlässige Leistung des Systems. Alle Preise der Komponenten verstehen sich als Richtpreise für eine schlüsselfertige Installation bis zur Inbetriebnahme.

Home Energy Komponenten

Ladestation

Keba P30 c-Series (11 kW)

Schlüsselfertige Ladestation KEBA P30 inkl. 11kW Hausinstallation (10 m Leitungslänge und 1 Wanddurchbruch), *Sicherheitsnachweis*, *Integration und Instruktion in Home Energy*

- Geprüfte Hausinstallation für 11kW Dauerladeleistung mit vorgeschriebenen Sicherheitselementen inkl. Meldewesen und SINA
- Betriebswahl: Solarladung, Kombiladung, Schnellladung inkl. Visualisierung in Home Energy App und Web
- Individuelle Beratung bei optionalen Zusatzwünschen: RFID, Ladekabel, Schnellladeschalter, etc.

CHF 3'200.-

Intelligente Steuerung

SDS Solar-Log 1200

Intelligente Steuerung inklusive Verbrauchszähler, *Installation und eigenverbrauchsoptimierter Konfiguration durch den Fachmann*

- Visualisierung der Energieflüsse und des Eigenverbrauchs via Home Energy App und Web
- Optimierte Steuerung des Eigenverbrauchs durch Ansteuerung einzelner Verbraucher
- Anschluss von Wechselrichtern unterschiedlicher Hersteller möglich

CHF 1'700.-

Zwischensumme	CHF	4'900.-
Gesamtkosten	**CHF**	**4'900.-**

Preise inkl. MwSt.

Ihre BKW Home Energy Services

Um langfristig den optimalen Betrieb Ihres Systems sicherzustellen, unterstützen wir Sie mit unseren Dienstleistungen.

Monitoring Silber

Persönliche Überwachung der Anlage per Home Energy App und Web.

- Von überall und jederzeit Zugriff auf die Visualisierung
- Jederzeit Klarheit über die aktuelle Stromproduktion
- Produktionsausfälle bleiben nicht länger unbemerkt

CHF 0.-

Support

Service Hotline, kostenlos von Montag bis Donnerstag von 8.00 bis 17.00 Uhr und Freitag von 08.00 bis 16.30 Uhr.

- Kompetente Beratung und Anlaufstelle bei Störungen und Ausbauten
- Auf Wunsch sofortige Erstanalyse durch Fernwartung über die Steuerung

CHF 0.-

Jährliche Kosten	**CHF**	**0.-**

Preise inkl. MwSt.

Einsparmöglichkeiten

Der berechnete Förderbeitrag bezieht sich auf den einmaligen Investitionsbeitrag (Einmalvergütungen) des Bundesamtes für Energie. Das Angebot ist nicht mit anderen Promotionen der BKW Energie AG oder seiner Partner kumulierbar. Diese werden mit einer allfälligen Promotion verrechnet. Zeitlich begrenzte Promotionen oder Förderbeiträge können nur dann geltend gemacht werden, wenn der Vertragsabschluss vor dem Ende des veröffentlichen Auslaufdatums stattfindet. Informieren Sie sich ausserdem über mögliche Steuervergünstigungen bei Ihrem kantonalen Steueramt. Für definitive Bewilligungen von Förderbeiträgen und Steuererleichterungen sind nur die zuständigen Stellen befugt.
Es gelten die Zahlungsbedingungen gemäss den Allgemeinen Geschäftsbedingungen Dienstleistungen der BKW Energie AG sowie den Besonderen Geschäftsbedingungen BKW Home Energy (www.bkw.ch/agb)

Vielen Dank für Ihr Interesse

Gerne beantworten wir Fragen zu Ihrer Richtofferte unter: **0800 121 128** oder per E-Mail: **homeenergy@bkw.ch**

BKW Kundenummer:
Referenz: 14068

Ihre Angaben

Die nachfolgenden Angaben dienen als Grundlage für Erstellung einer Fixofferte, um uns so bestmöglich auf eine Besichtigung bei Ihnen vorbereiten zu können.

Allgemein (Persönliche Daten)

Ansprache	Herr
Name	CAvin
Vorname	Baptiste
E-Mail	baptiste.cavin@students.hevs.ch
Interesse Finanzierungs-Lösung	Nein

Stromverbrauch

Personen im Haushalt	5
Zukünftige und bestehende Verbraucher	Elektroauto
Stromverbrauch (kWh pro Jahr)	5200

Wärmebedarf

Wohnfläche (m²)	-
Gebäudetyp	-
Gebäudestandard	-
Warmwasser durch Heizungs-Wärmepumpe	Nein
Heizleistung (kWth)	-
Wärmeenergiebedarf (kWh pro Jahr)	0

Fotovoltaikanlage

Installierte Leistung (kWp)	5

Heizung und Warmwasser

Aktuelles Heizungssystem	Gasheizung
Aktuelle Warmwasseraufbereitung	Solarkollektoren

Anhang 2:

Pflichtenheft

Pflichtenheft

Algorithmus zum Laden von Elektroautos

Bachelorarbeit von Baptiste Cavin

1 Einführung

Herkömmliche Ladesäulen für Elektrofahrzeuge in Privat- und Mehrfamilienhäusern laden mit Nennleistung, unabhängig von der verfügbaren Leistung der hauseigenen Photovoltaikanlage (PV-Anlage). Im Rahmen der Bachelorarbeit soll ein System (Smart-Charge System) entwickelt werden, das Ladesäulen intelligent steuert, um den Preis der Ladung und somit auch den Eigenverbrauch zu optimieren. Dazu wird ein Algorithmus implementiert, der Anhand der Vorgaben des Benutzers ein möglichst vorteilhaftes Ladeprofil berechnet.

2 Funktionsprinzip

Das Smart-Charge System kann in einem Gebäude installiert werden, das eine PV-Anlage, eine steuerbare Ladesäule (in unserem Fall von KEBA) und ein Gateway (in unserem Fall von Misurio) besitzt. Das Gateway sammelt Daten von verschiedenen Analgen im Haus und schickt diese in die Cloud. Über eine Plattform (in unserem Fall die EnergyON Plattform) wird das Smart-Charge System zu den relevanten Daten zugreifen, sowie die Ladesäule steuern.

Das zu entwickelnde System besteht aus einem Algorithmus, der den Ladevorgang optimiert, und einem Interface, das die Vorgaben des Nutzers erfasst.

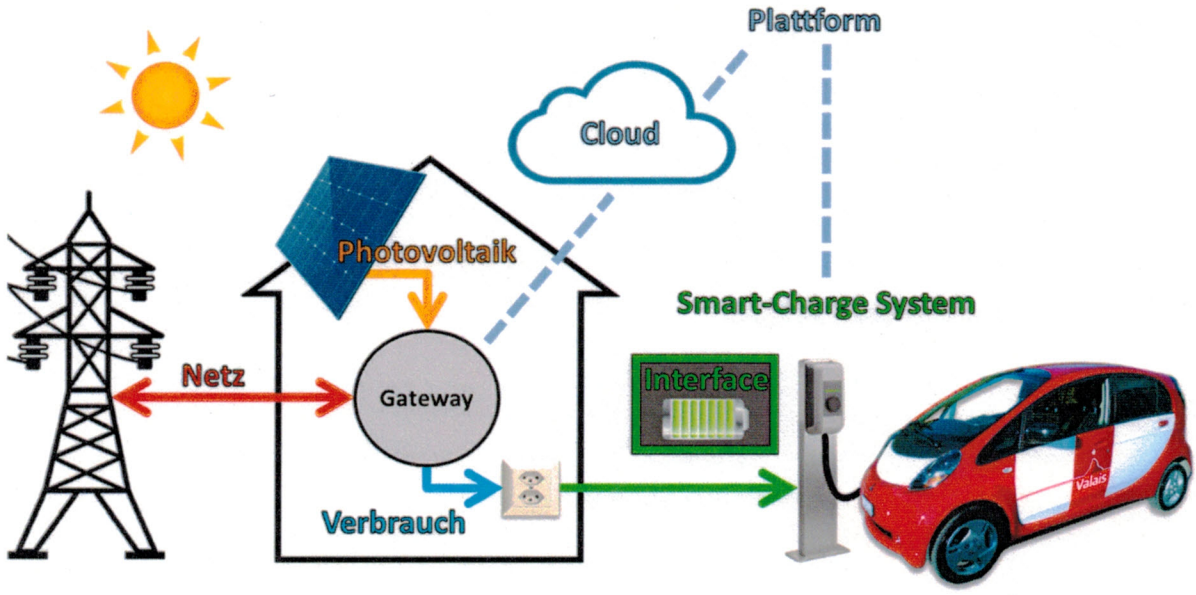


Abbildung 1: Funktionsprinzip Smart-Charge System

3 Blockschema

Im untenstehenden Schema kann man die Signale zwischen den verschiedenen Blöcken sehen. Der Algorithmus des Prototyps wird als externe Komponente ausserhalb der Plattform entwickelt. Das heisst, dass die Plattform wie eine Leitung betrachtet wird, die Informationen zwischen Algorithmus und Ladesäule weitergibt.

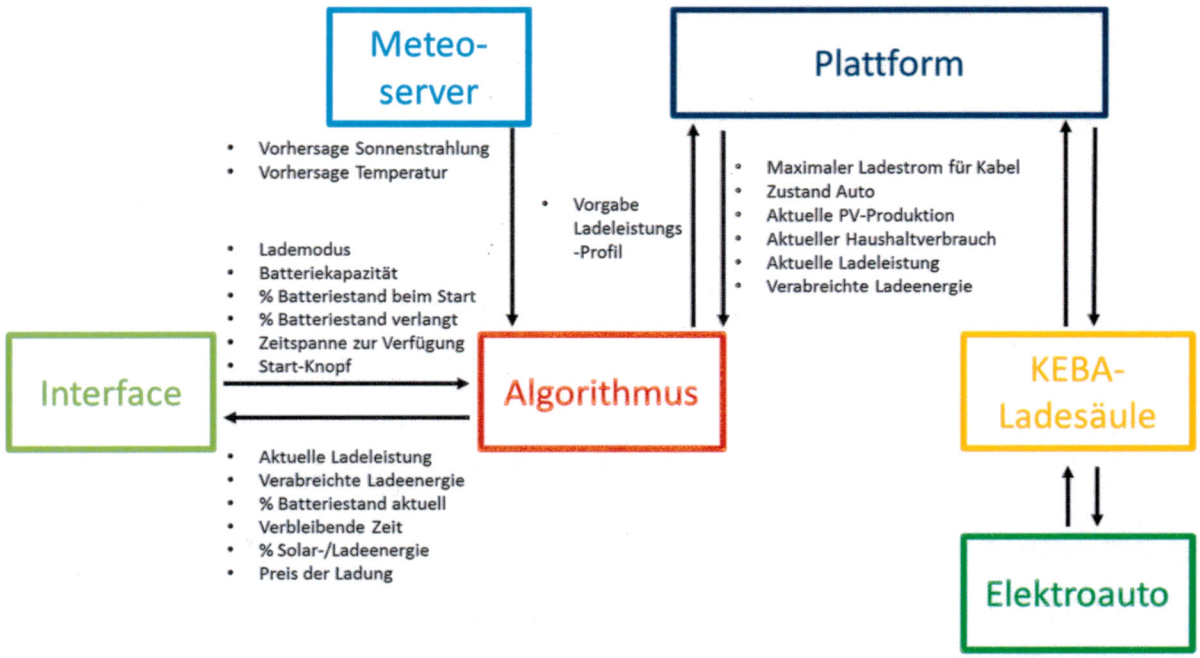


Abbildung 2: Blockschema

4 Ressourcen

Damit das Smart-Charge System funktionieren kann, braucht es:

- ein Gebäude mit einer Photovoltaikanlage
- ein Gateway, das die Leistungen der PV-Anlage und des Haushaltverbrauchs misst
- mindestens eine mit dem Gateway vernetzte steuerbare Ladesäule
- mindestens ein Elektroauto
- eine Plattform, über die man zu den Informationen des Gateways zugreifen kann und die Ladesäule ansteuern kann

Wichtig: Das System kann die Ladung eines Fahrzeuges nur dann signifikant optimieren, wenn während der Ladezeit ein Leistungsüberschuss der PV-Anlage vorliegt. Das bedingt, dass das Elektroauto mindestens einige Stunden tagsüber zuhause angesteckt sein muss.

5 Funktionalitäten

5.1 Interface

Wenn der Benutzer sein Elektroauto an die Ladesäule anschliesst, kann er über ein Interface die gewünschten Ladeparameter eingeben und die aktuellen Ladeinformationen ablesen.

5.2 Lademodi

Mit dem Smart-Charge System kann der Benutzer zwischen 4 Lademodi auswählen:

Modus	Beschreibung	Manuelle Eingabe	Anzeige
FastCharge	Das Auto wird so schnell wie möglich geladen, ohne die Maximalleistung des Hausanschlusses zu überschreiten.	Maximale Ladeleistung [kW]	Ladeleistung [kW] Ladeenergie [kWh] Preis der Ladung [Fr.]
SmartCharge	Das Auto wird bis zu einer gewissen Zeit auf einen vorgegebenen Batteriestand geladen. Die Leistung der Säule wird nach einem Profil gesteuert, das den Preis der Ladung und somit den Eigenverbrauch optimiert.	Endzeit [hh:mm] Batteriestand verlangt [%] Batteriestand aktuell [%] Batteriekapazität [kWh]	Ladeleistung [kW] Ladeenergie [kWh] Verhältnis Solar-/Ladeenergie [%] Batteriestand [%] Preis der Ladung [Fr.]
SolarCharge	Das Auto wird nur mit der verbleibenden Leistung der PV-Anlage geladen.	Maximale Ladeleistung [kW]	Ladeleistung [kW] Ladeenergie [kWh] Preis der Ladung [Fr.]
TimedCharge (optional)	Das Auto wird nur während einer gewissen Zeitspanne geladen	Startzeit [hh:mm] Endzeit [hh:mm] Maximale Ladeleistung [kW]	Ladeleistung [kW] Ladeenergie [kWh] Verhältnis Solar-/Ladeenergie [%] Preis der Ladung [Fr.]

Tabelle 1: die 4 Lademodi

5.3 Ladeprofil

Der Algorithmus generiert ein Ladeprofil für die Ladesäule, basierend auf Prognosen der PV-Leistung und des Haushaltverbrauchs, die anhand der Wettervorhersagen generiert werden.

Alle 5 Minuten wird das Ladeprofil vom Algorithmus neu berechnet und falls nötig angepasst.

5.4 Sicherheitsmarge

Um schnelle Variationen der Leistung der PV-Anlage und des Haushaltverbrauchs zu berücksichtigen, wird eine Marge zur maximalen Leistung des Hausanschlusses gelassen. Optional könnte die Periode des Algorithmus verkleinert werden.

6 Lieferbare Produkte

Am Schluss der Arbeit (04.09.2017) wird ein funktioneller Prototyp des Smart-Charge Systems, bestehend aus dem Algorithmus und einem Interface, geliefert. Optional wird dieser in einem Gebäude mit den notwendigen Ressourcen getestet. Eine Analyse der Optimierung wird anhand der vorhandenen Daten durchgeführt.

7 Datum und Unterschrift

Steg, den 06.06.2017

Baptiste Cavin	Gilbert Morand	Johann Eberhardt
Student (HEVS)	Dozent (HEVS)	Experte (Winsun)
Claudio Steiner	Kilian Jossen	
(Lauber-Iwisa)	(Misurio)	

Anhang 3:

KeContact P30 Datenblatt

KeContact P30

Die Wallbox der nächsten Generation

KeContact P30 – lädt intelligenter als jemals zuvor

Strom laden können viele. Wir von KEBA sind überzeugt, dass es für wahre grüne Mobilität mehr braucht als das. Deshalb ist unsere neueste Wallbox KeContact P30 – übrigens die dritte Generation an Wallboxen von KEBA – eine, die alles kann. Für mehr Möglichkeiten, Konnektivität und mehr Elektromobilität.

Ein Produkt für alle

Mit nur einem Produkt in verschiedenen Ausprägungen, Varianten und Ausstattungsserien können wir sämtliche Elektrofahrzeuge und Plug-in Hybride, Anwendungsfälle, Einsatzgebiete und Märkte weltweit abdecken.

Sicherheit beim Laden

KeContact P30 beinhaltet auch eine Gleichfehlerstromüberwachung, wodurch ein teurer vorgelagerter FI Typ B obsolet wird.

Neue Anwendungsmöglichkeiten

KeContact P30 bietet dank neuester Kommunikationsstandards und Features völlig neue Anwendungsmöglichkeiten:

- **Öffentlicher und halb-öffentlicher Bereich**

Via OCPP 1.5 und 2.0 lässt sich KeContact P30 einfach an ein Zentralsystem anbinden. Im öffentlichen und halb-öffentlichen Bereich ist das besonders interessant für Monitoring, Lastmanagement oder die Abrechnung von Energie. Dank des voll integrierten MID-zertifizierten Strom-, Spannungs-, Leistungs- und Energiezählers* können Kilowattstunden im öffentlichen und halb-öffentlichen Bereich einfach abgerechnet werden.

- **Privater Bereich**

Via User Data Protocol (UDP) kann im Smart Home die Wallbox einfach gesteuert bzw. Statusinformationen abgerufen werden.

So kann zum Beispiel der maximal erlaubte Strombezug des Elektrofahrzeuges in Abhängigkeit von Photovoltaikanlage, Batteriespeicher oder Wärmepumpe geregelt werden.

Externe Zählerdaten können zudem via Modbus-Protokoll* einfach ausgelesen werden, sodass die Ladevorgänge des Elektrofahrzeuges, z.B. in Abhängigkeit des Hausanschlusses, gesteuert werden können.

Personen mit Firmenfahrzeugen können mit KeContact P30 die zu Hause geladene Energie einfach gegenüber der Firma oder dem Leasingunternehmen abrechnen.

Kommunikationsschnittstellen

Dank diverser Kommunikationsschnittstellen lässt sich KeContact P30 in bestehende Zentralsysteme und Smart Homes einbinden:

- USB für Firmwareupdates oder Logfiledownloads ohne PC bzw. Notebook
- Ethernet RJ45 für Firmwareupdates oder Logfiledownloads mit PC bzw. Notebook
- Ethernet LSA+ für die Anbindung externer Zähler via Modbus und für eine permanente Kommunikationsverbindung zu OCPP Backend, Smart Home Server oder weiteren KeContact P30
- GSM für die Anbindung an OCPP Zentralsysteme
- WLAN* für die drahtlose Einbindung von Wallboxen in ein bestehendes Netzwerk bzw. die Kommunikation mit weiteren Wallboxen für lokales Lastmanagement

*** verfügbar ab 2017**

KEBA®

Automation by innovation.

KeContact P30 – Ausstattungsserien

KeContact P30 ist in vier unterschiedlichen Ausstattungsserien erhältlich. Die Serien sind aufeinander aufbauend, das bedeutet, dass Sie bei einer höherwertigen Serie die angeführten Features zusätzlich erhalten. Die **e-series** ist die Einstiegsvariante der KeContact P30 für einfaches, kostenoptimiertes Laden. KeContact P30 **b-series** bietet ein hohes Maß an individuellen Brandingmöglichkeiten, Möglichkeiten zur Identifizierung und die Schnellladefunktion mit 22kW. Mit der **c-series** wird KeContact P30 zur Kommunikationsschnittstelle für intelligent gesteuertes Laden. Um ganzheitliche E-Mobilitätslösungen wie z.B. lokales Lastmanagement zu schaffen, bedarf es der **x-series**. So finden Sie für Ihre Bedürfnisse immer die passende Stromladestation!

Vier Ausstattungsserien für Ihre Bedürfnisse:

			GSM für die drahtlose Kommunikation mit dem OCPP-Backend
		WLAN-Kommunikation als Slave*	WLAN-Kommunikation als Master*
		Lokales Lastmanagement als Slave	Lokales Lastmanagement als Master
		OCPP-Kommunikation als Slave	OCPP-Kommunikation als Master
		Slave für Master/Slave-Kommunikation	Master für Master/Slave-Kommunikation
		UDP-Schnittstelle (Smart Home Automation)	UDP-Schnittstelle (Smart Home Automation)
		Kommunikation mit einem externen Stromzähler über Modbus	Kommunikation mit einem externen Stromzähler über Modbus*
		Ethernet-Schnittstelle (LSA+)	Ethernet-Schnittstelle (LSA+)
		Energiezähler (MID-zertifiziert*)	Energiezähler (MID-zertifiziert*)
	Authentifizierung (RFID, Schlüssel)	Authentifizierung (RFID, Schlüssel)	Authentifizierung (RFID, Schlüssel)
	Freigabeeingang / Schaltausgang	Freigabeeingang / Schaltausgang	Freigabeeingang / Schaltausgang
	Stromüberwachung	Stromüberwachung	Stromüberwachung
	DC-Fehlerstromerkennung	DC-Fehlerstromerkennung	DC-Fehlerstromerkennung
	Customizing / Branding	Customizing / Branding	Customizing / Branding
Ethernet-Schnittstelle (RJ45)	Ethernet-Schnittstelle (RJ45)	Ethernet-Schnittstelle (RJ45)	Ethernet-Schnittstelle (RJ45)
USB-Schnittstelle	USB-Schnittstelle	USB-Schnittstelle	USB-Schnittstelle
e-series	**b-series**	**c-series**	**x-series**
Einphasig bis zu 20A (4,6kW)	Dreiphasig bis zu 32A (22kW)	Dreiphasig bis zu 32A (22kW)	Dreiphasig bis zu 32A (22kW)
CE-Kennzeichnung	VDE-zertifiziert	VDE-zertifiziert	VDE-zertifiziert

*** verfügbar ab 2017**

Vertriebspartner

Automation by innovation.

KEBA AG, Gewerbepark Urfahr, A-4041 Linz, Telefon: +43 732 7090-0, Fax: +43 732 730910, keba@keba.com

Anhang 4:

Testprogramm

- RestCommunication.pro
- main.cpp
- mainwindow.h
- mainwindow.cpp

```
#-------------------------------------------------
#
# Project created by QtCreator 2017-06-30T11:48:27
#
#-------------------------------------------------

QT       += core gui
QT      += network #for QtNetworkManager

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = RestCommunication
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

```cpp
#include "mainwindow.h"
#include <QApplication>
#include <QDebug>
#include <QCoreApplication>
#include <QDebug>
#include <QProcess>
#include <QDir>

int main(int argc, char *argv[])
{


    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDateTime>
#include "qsharedmemory.h"

namespace Ui {
class MainWindow;
}

struct POWER{
    float value;
    float setpoint;
    float prevision;
    QDateTime timestamp;
};

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_BP_access_token_clicked();

    void on_BP_puissance_clicked();

    void on_HS_consigne_valueChanged(int value);

    void on_BP_consigne_clicked();

    void on_BP_profil_puissance_clicked();

    void myTimer();

    void on_BP_Export_clicked();

    void on_BP_heure_actuelle_clicked();

    void on_BP_profil_consigne_clicked();

    void on_BP_heure_actuelle_2_clicked();

    void on_BP_actualiser_etat_clicked();

    void on_BP_activation_clicked();

    void on_BP_desactivation_clicked();

private:
    Ui::MainWindow *ui;
    QSharedMemory sharedMemory;
```

```
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QCoreApplication>
#include <QDebug>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QUrl>
#include <QUrlQuery>
#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
#include <QDateTime>
#include <QTimer>
#include <QTimeZone>
#include <QProcess>
#include <QBuffer>
#include <QMessageBox>
#include <QObject>


//functions
QString sendAccessRequest();
QString sendPowerRequest ();
QString sendPowerSetpoint(QString, float);
QString sendProfileRequest(QString, qint64, qint64, QString);
QString getRest(QString);
QString putRest(QString,QString);

//variables et autres
QString access_token;
uint chargeSize;
QBuffer buffer;
QDataStream out(&buffer);

MainWindow::MainWindow(QWidget *parent) ://constructor
    sharedMemory("mySharedMemory"),
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //initialisation du buffer pour stoquer les données à envoyer
dans la mémoire partagée
    buffer.open(QBuffer::ReadWrite);


    //set up timer
    QTimer *timer = new QTimer(this);
    connect(timer,SIGNAL(timeout()),this, SLOT(myTimer()));
    timer->start(1000);//start timer toutes les secondes
}

MainWindow::~MainWindow() //destructor
{
```

```
    delete ui;
}

//timer interrupt
void MainWindow::myTimer(){
    ui->LB_time->setText("Clock: " +
QTime::currentTime().toString());
}

void MainWindow::on_BP_access_token_clicked()
{
    ui->BP_access_token->setDisabled(true);
    QString stringAnswer = sendAccessRequest();
    ui->BP_access_token->setEnabled(true);

    //créer un objet Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    access_token = jsonObject.value("access_token").toString();
    ui->txt_access_token->setText("Bearer " + access_token);
}

QString sendAccessRequest(){

    QString stringAnswer = NULL;
    char body[] = "username=---&password=---&client_id=direct-
access&grant_type=password";
    QUrl url = (QUrl)"https://auth.energyon.ch/auth/realms/lauber-
iwisa/protocol/openid-connect/token";

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);

    //ajouter un header
    request.setHeader(QNetworkRequest::ContentTypeHeader,"applicat
ion/x-www-form-urlencoded");

    //requête post avec body
    QNetworkReply *reply = mgr.post(request, body);

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
```

```
        stringAnswer = reply->readAll();
        delete reply;
    }
    else{
         //échec
        qDebug() << "échec access token";
        stringAnswer = reply->readAll();
        delete reply;
    }
    return stringAnswer;
}

void MainWindow::on_BP_puissance_clicked()
{
    ui->BP_puissance->hide();
    QString stringAnswer = sendPowerRequest();
    ui->BP_puissance->show();

    ui->txt_puissance->setText(stringAnswer);

    //créer un objet Json json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    QJsonObject powerObject = jsonObject["data"].toObject();
    double power = powerObject.value("value").toDouble();

    ui->lcdPower->display(power);
}

QString sendPowerRequest(){
    QString stringAnswer = NULL;
    QUrl url = (QUrl)"https://api.energyon.ch/api/asset/00002HQ/
timeserieslast/measurement-loading-power";

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);

    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray("Bearer " + access_token.toUtf8()));

    //requête get
    QNetworkReply *reply = mgr.get(request);

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
```

```
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = reply->readAll();
        delete reply;
    }
    else{
         //échec
        qDebug() << "échec demande de la puissance actuelle";
        stringAnswer = reply->readAll();
        delete reply;
    }
    return stringAnswer;
}

void MainWindow::on_HS_consigne_valueChanged(int value)
{
    float power = value*(43.47-4.14)/100 + 4.14;
    ui->DSB_consigne->setValue(power);
}

QString sendPowerSetpoint(QString StringUrl, float maxPower){
    QString stringAnswer = NULL;
    QDateTime timestamp = QDateTime::currentDateTime();//ne joue
seulement un rôle pour la prévision
    QUrl url = (QUrl) StringUrl;
    //QString body = QString("{\"tstamp\":\"2017-01-01
00:00:00.00000\",\"value\":%1}").arg(maxPower);
    QString body = "{\"tstamp\":\"" +
timestamp.toUTC().toString("yyyy-MM-dd hh:mm:ss.000") + "\",
\"value\":" + QString::number(maxPower) + "}";

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);

    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray("Bearer " + access_token.toUtf8()));
    request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json");

    //requête put
    QNetworkReply *reply = mgr.put(request,body.toUtf8());

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé
```

```
    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = "Der Sollwert von " +
QString::number(maxPower) +" kW wurde erfolgreich gesendet am " +
timestamp.toString("yyyy-MM-dd hh:mm") + reply->readAll();
        delete reply;
    }
    else{
         //échec
        stringAnswer = "Sendefehler " + reply->readAll();
        delete reply;
    }
    return stringAnswer;
}

void MainWindow::on_BP_consigne_clicked()
{
    ui->BP_consigne->hide();
    float maxPower = ui->DSB_consigne->value();
    //envoyer consigne
    QString stringResponse = sendPowerSetpoint("https://
api.energyon.ch/api/asset/00002HQ/timeserieslast/control-max-
loading-power",maxPower);
    //envoyer prévision
    sendPowerSetpoint("https://api.energyon.ch/api/asset/00002HQ/
timeserieslast/forecast-loading-power",maxPower);
    ui->txt_statusConnection->setText(stringResponse);
    ui->BP_consigne->show();
}

void MainWindow::on_BP_profil_puissance_clicked()
{
    //saisir les prarmètres
    QString starttime = ui->LE_starttime->text();
    QString endtime = ui->LE_endtime->text();
    QString timestep = ui->LE_timestep->text();
    float totalenergyOnStart;
    bool first = true;

    QString format = "yyyy-MM-dd hh:mm";
    QDateTime startDateTime =
QDateTime::fromString(starttime,format);
    QDateTime endDateTime = QDateTime::fromString(endtime,format);

    //envoyer la requète
    ui->BP_profil_puissance->hide();
    QString stringAnswer1 = sendProfileRequest("https://
api.energyon.ch/api/asset/00002HQ/timeseries/measurement-loading-
power",startDateTime.toMSecsSinceEpoch(),endDateTime.toMSecsSinceE
poch(),timestep);
    QString stringAnswer2 = sendProfileRequest("https://
api.energyon.ch/api/asset/00002HQ/timeseries/control-max-loading-
power",startDateTime.toMSecsSinceEpoch(),endDateTime.toMSecsSinceE
poch(),timestep);
    QString stringAnswer3 = sendProfileRequest("https://
```

```
api.energyon.ch/api/asset/00002HQ/timeseries/forecast-loading-
power",startDateTime.toMSecsSinceEpoch(),endDateTime.toMSecsSinceE
poch(),timestep);
    QString stringAnswer4 = sendProfileRequest("https://
api.energyon.ch/api/asset/00002HQ/timeseries/measurement-loading-
totalenergy",startDateTime.toMSecsSinceEpoch(),endDateTime.toMSecs
SinceEpoch(),timestep);

    //pour tester offline
//    stringAnswer1 = "{\"description\":\"Ladeleistung\",\"unit\":
\"kW\",\"data\":{\"name\":\"00002HQ.08\",\"unit\":\"\",\"values\":
[{\"tstamp\":\"2017-06-16 12:00:00.00000\",\"value\":
1.3158613000000001},{\"tstamp\":\"2017-06-16 12:10:00.00000\",
\"value\":13.211924699999999},{\"tstamp\":\"2017-06-16
12:20:00.00000\",\"value\":13.060586299999999},{\"tstamp\":
\"2017-06-16 12:30:00.00000\",\"value\":6.4832499},{\"tstamp\":
\"2017-06-16 12:40:00.00000\",\"value\":2.9211921},{\"tstamp\":
\"2017-06-16 12:50:00.00000\",\"value\":2.6418951000000006},
{\"tstamp\":\"2017-06-16 13:00:00.00000\",\"value\":
2.6488330999999996},{\"tstamp\":\"2017-06-16 13:10:00.00000\",
\"value\":2.6857368},{\"tstamp\":\"2017-06-16 13:20:00.00000\",
\"value\":2.6358815000000004},{\"tstamp\":\"2017-06-16
13:30:00.00000\",\"value\":2.6114672999999997},{\"tstamp\":
\"2017-06-16 13:40:00.00000\",\"value\":0.7837584999999999},
{\"tstamp\":\"2017-06-16 13:50:00.00000\",\"value\":0.0},{\"tstamp
\":\"2017-06-16 14:00:00.00000\",\"value\":2.7593751},{\"tstamp\":
\"2017-06-16 14:10:00.00000\",\"value\":1.6336382},{\"tstamp\":
\"2017-06-16 14:20:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:30:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:40:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:50:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 15:00:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 15:10:00.00000\",\"value\":0.7261934000000001},
{\"tstamp\":\"2017-06-16 15:20:00.00000\",\"value\":
0.7192160000000001},{\"tstamp\":\"2017-06-16 15:30:00.00000\",
\"value\":1.6915023999999999},{\"tstamp\":\"2017-06-16
15:40:00.00000\",\"value\":0.9491908},{\"tstamp\":\"2017-06-16
15:50:00.00000\",\"value\":0.0}],\"type\":\"EC_INPUT\"}}";
//    stringAnswer2 = "{\"description\":\"Ladeleistung\",\"unit\":
\"kW\",\"data\":{\"name\":\"00002HQ.08\",\"unit\":\"\",\"values\":
[{\"tstamp\":\"2017-06-16 12:00:00.00000\",\"value\":
2.3158613000000001},{\"tstamp\":\"2017-06-16 12:10:00.00000\",
\"value\":13.211924699999999},{\"tstamp\":\"2017-06-16
12:20:00.00000\",\"value\":13.060586299999999},{\"tstamp\":
\"2017-06-16 12:30:00.00000\",\"value\":6.4832499},{\"tstamp\":
\"2017-06-16 12:40:00.00000\",\"value\":2.9211921},{\"tstamp\":
\"2017-06-16 12:50:00.00000\",\"value\":2.6418951000000006},
{\"tstamp\":\"2017-06-16 13:00:00.00000\",\"value\":
2.6488330999999996},{\"tstamp\":\"2017-06-16 13:10:00.00000\",
\"value\":2.6857368},{\"tstamp\":\"2017-06-16 13:20:00.00000\",
\"value\":2.6358815000000004},{\"tstamp\":\"2017-06-16
13:30:00.00000\",\"value\":2.6114672999999997},{\"tstamp\":
\"2017-06-16 13:40:00.00000\",\"value\":0.7837584999999999},
{\"tstamp\":\"2017-06-16 13:50:00.00000\",\"value\":0.0},{\"tstamp
\":\"2017-06-16 14:00:00.00000\",\"value\":2.7593751},{\"tstamp\":
\"2017-06-16 14:10:00.00000\",\"value\":1.6336382},{\"tstamp\":
```

```
\"2017-06-16 14:20:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:30:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:40:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:50:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 15:00:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 15:10:00.00000\",\"value\":0.7261934000000001},
{\"tstamp\":\"2017-06-16 15:20:00.00000\",\"value\":
0.7192160000000001},{\"tstamp\":\"2017-06-16 15:30:00.00000\",
\"value\":1.6915023999999999},{\"tstamp\":\"2017-06-16
15:40:00.00000\",\"value\":0.9491908},{\"tstamp\":\"2017-06-16
15:50:00.00000\",\"value\":0.0}],\"type\":\"EC_INPUT\"}}";
//    stringAnswer3 = "{\"description\":\"Ladeleistung\",\"unit\":
\"kW\",\"data\":{\"name\":\"00002HQ.08\",\"unit\":\"\",\"values\":
[{\"tstamp\":\"2017-06-16 12:00:00.00000\",\"value\":
3.3158613000000001},{\"tstamp\":\"2017-06-16 12:10:00.00000\",
\"value\":13.211924699999999},{\"tstamp\":\"2017-06-16
12:20:00.00000\",\"value\":13.060586299999999},{\"tstamp\":
\"2017-06-16 12:30:00.00000\",\"value\":6.4832499},{\"tstamp\":
\"2017-06-16 12:40:00.00000\",\"value\":2.9211921},{\"tstamp\":
\"2017-06-16 12:50:00.00000\",\"value\":2.6418951000000006},
{\"tstamp\":\"2017-06-16 13:00:00.00000\",\"value\":
2.6488330999999996},{\"tstamp\":\"2017-06-16 13:10:00.00000\",
\"value\":2.6857368},{\"tstamp\":\"2017-06-16 13:20:00.00000\",
\"value\":2.6358815000000004},{\"tstamp\":\"2017-06-16
13:30:00.00000\",\"value\":2.6114672999999997},{\"tstamp\":
\"2017-06-16 13:40:00.00000\",\"value\":0.7837584999999999},
{\"tstamp\":\"2017-06-16 13:50:00.00000\",\"value\":0.0},{\"tstamp
\":\"2017-06-16 14:00:00.00000\",\"value\":2.7593751},{\"tstamp\":
\"2017-06-16 14:10:00.00000\",\"value\":1.6336382},{\"tstamp\":
\"2017-06-16 14:20:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:30:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:40:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 14:50:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 15:00:00.00000\",\"value\":0.0},{\"tstamp\":
\"2017-06-16 15:10:00.00000\",\"value\":0.7261934000000001},
{\"tstamp\":\"2017-06-16 15:20:00.00000\",\"value\":
0.7192160000000001},{\"tstamp\":\"2017-06-16 15:30:00.00000\",
\"value\":1.6915023999999999},{\"tstamp\":\"2017-06-16
15:40:00.00000\",\"value\":0.9491908},{\"tstamp\":\"2017-06-16
15:50:00.00000\",\"value\":0.0}],\"type\":\"EC_INPUT\"}}";

    //créer un objet Json
    QJsonDocument jsonAnswer1 =
QJsonDocument::fromJson(stringAnswer1.toUtf8());
    QJsonObject jsonObject1 = jsonAnswer1.object();

    QJsonDocument jsonAnswer2 =
QJsonDocument::fromJson(stringAnswer2.toUtf8());
    QJsonObject jsonObject2 = jsonAnswer2.object();

    QJsonDocument jsonAnswer3 =
QJsonDocument::fromJson(stringAnswer3.toUtf8());
    QJsonObject jsonObject3 = jsonAnswer3.object();

    QJsonDocument jsonAnswer4 =
QJsonDocument::fromJson(stringAnswer4.toUtf8());
```

```
    QJsonObject jsonObject4 = jsonAnswer4.object();

    /*exemple de réponse
      "data": {
        "name": "00002HQ.0I",
        "unit": "",
        "values": [
              {
                "tstamp": "2017-07-06 09:30:00.00000",
                "value": 43.47
              },
              {
                "tstamp": "2017-07-06 09:31:00.00000",
                "value": 43.47
              },],
        "type": "EC_INPUT"
      }
    */

    //extraire les données
    QJsonObject dataObject1 = jsonObject1["data"].toObject();
    QJsonValue valuesValue1 = dataObject1.value("values");
    QJsonArray valuesArray1 = valuesValue1.toArray();

    QJsonObject dataObject2 = jsonObject2["data"].toObject();
    QJsonValue valuesValue2 = dataObject2.value("values");
    QJsonArray valuesArray2 = valuesValue2.toArray();

    QJsonObject dataObject3 = jsonObject3["data"].toObject();
    QJsonValue valuesValue3 = dataObject3.value("values");
    QJsonArray valuesArray3 = valuesValue3.toArray();

    QJsonObject dataObject4 = jsonObject4["data"].toObject();
    QJsonValue valuesValue4 = dataObject4.value("values");
    QJsonArray valuesArray4 = valuesValue4.toArray();


    chargeSize = valuesArray1.size();
    POWER charge[chargeSize];
    QString affichage = NULL;
    QString affichage2 = NULL;
    QString affichage3 = NULL;

    //envoyer le nombre d'éléments
    buffer.reset();
    out << chargeSize;

    //remplir le tableau charge
    for(uint i = 0; i < chargeSize; i++){
        QJsonObject timevalueObject1 = valuesArray1[i].toObject();
        float value =
(float)timevalueObject1.value("value").toDouble();

        QJsonObject timevalueObject2 = valuesArray2[i].toObject();
        float setpoint =
(float)timevalueObject2.value("value").toDouble();
```

```
        QJsonObject timevalueObject3 = valuesArray3[i].toObject();
        float prevision =
(float)timevalueObject3.value("value").toDouble();

        QJsonObject timevalueObject4 = valuesArray4[i].toObject();
        float totalenergy =
(float)timevalueObject4.value("value").toDouble();


        QString date =
timevalueObject1.value("tstamp").toString();
        QDateTime dateUTC = QDateTime::fromString(date, "yyyy-MM-
dd hh:mm:ss.00000");
        dateUTC.setTimeZone(QTimeZone::utc());//pour éviter le
décalage horaire vu que les données sont déjà en UTC

        charge[i].timestamp = dateUTC;
        charge[i].value = value;
        charge[i].setpoint = setpoint;
        charge[i].prevision = prevision;

        out << dateUTC << value << setpoint << prevision;

        if(first){
            totalenergyOnStart = totalenergy;
            first = false;
        }

        affichage += dateUTC.toLocalTime().toString("yyyy-MM-dd
hh:mm") + " " + QString::number(value) + " kW set: " +
QString::number(setpoint) + " (" + QString::number(prevision) + ")
\n";
        affichage2 += dateUTC.toLocalTime().toString("hh:mm:ss ")
+ QString::number(value) + " " + QString::number(setpoint) + " " +
QString::number(prevision)+ " " + QString::number(totalenergy-
totalenergyOnStart) + "\n" ;
        affichage3 += QString::number(totalenergy-
totalenergyOnStart) +"\n";
    }
    ui->txt_profil_puissance->setText(affichage);
    ui->BP_profil_puissance->show();


}

QString sendProfileRequest(QString urlPrefix, qint64 start, qint64
end, QString step){
    QString stringAnswer = NULL;
    QString stringUrl = urlPrefix + "?starttime="
                        + QString::number(start) + "&endtime=" +
QString::number(end + 60000) + "&period=" + step;
    QUrl url = (QUrl)(stringUrl);

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;
```

```
    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);

    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray("Bearer " + access_token.toUtf8()));

    //requête get
    QNetworkReply *reply = mgr.get(request);

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = reply->readAll();
        delete reply;
    }
    else{
         //échec
        qDebug() << "échec demande de la puissance actuelle";
        stringAnswer = reply->readAll();
        delete reply;
    }
    return stringAnswer;
}

void MainWindow::on_BP_Export_clicked()
{
    ui->BP_Export->hide();

    //libérer la mémoire si elle n'est pas libre
    if (sharedMemory.isAttached()){
        sharedMemory.detach();
    }
    // load into shared memory
    int size = buffer.size();

    if (!sharedMemory.create(size)) {
        QMessageBox::critical(this,"Shared memory error","Unable
to create shared memory segment");
        return;
    }
    sharedMemory.lock();
    char *to = (char*)sharedMemory.data();
    const char *from = buffer.data().data();
    memcpy(to, from, qMin(sharedMemory.size(), size));
    sharedMemory.unlock();
```

```
    //lancer l'application watchMe.exe
    QProcess *process = new QProcess(this);
    process->execute("../ASQ/dev/watchMe2017/exe/watchMe.exe");
    ui->BP_Export->show();
}

void MainWindow::on_BP_heure_actuelle_clicked()
{
    QDateTime now = QDateTime::currentDateTimeUtc();
    ui->LE_endtime->setText(now.toLocalTime().toString("yyyy-MM-dd
hh:mm"));
    QTime interval= ui->TE_interval->time();
    int intervalSec = interval.minute()*60+interval.hour()*3600;
    if(intervalSec >0){
    QDateTime previous = now.addSecs(-intervalSec);
    ui->LE_starttime-
>setText(previous.toLocalTime().toString("yyyy-MM-dd hh:mm"));
    }
}

void MainWindow::on_BP_profil_consigne_clicked()
{
    ui->BP_profil_consigne->hide();
    QString body = "[";
    QString display = NULL;
    QString texte = ui->TE_profil_consigne->toPlainText();
    QDateTime startTime = QDateTime::fromString(ui-
>LE_starttime_2->text(),"yyyy-MM-dd hh:mm");
    QDateTime time;
    QTime relativeTime;


    int nbrValues = texte.count(":");//déterminer nombre de
valeurs
    texte.remove(" ");
    if(texte.right(1) != "\n"){//ajouter un retour à la ligne à la
fin si nécessaire
     texte.append("\n");
    }
    //extraire les temps (grâce au :) et les valeurs (grâce au \n)
    for(int i = 0 ; i < nbrValues ; i++){
        int pos = texte.indexOf(":");
        QString timeString = texte.mid(pos-2,5);
        texte.remove(0,pos+3);
        pos = texte.indexOf("\n",1);
        int j = pos-1;
        while((j>=0) && (texte.at(j).isDigit() ||
(texte.at(j)=='.'))){
            j--;
        }
        QString value = texte.mid(j+1, pos-j-1);
        texte.remove(0,pos);
        relativeTime = QTime::fromString(timeString,"hh:mm");
        time = startTime.addSecs(relativeTime.minute()*60 +
relativeTime.hour()*3600);
```

```
        body.append("{\"tstamp\":\"" +
time.toUTC().toString("yyyy-MM-dd hh:mm:ss.000") + "\",\"value
\":"+ value + "},");
        display.append(time.toLocalTime().toString("yyyy-MM-dd
hh:mm") + ", " + value + " kW\n");
    }
    body.remove(body.size()-1,1);//enlever la dernière virgule
    body.append("]");
    //envoyer profil de consigne
    QString stringAnswer = putRest("https://api.energyon.ch/api/
asset/00002HQ/timeseries/control-max-loading-power",body);
    //envoyer dans les prévisions
    putRest("https://api.energyon.ch/api/asset/00002HQ/timeseries/
forecast-loading-power",body);
    ui->TE_profil_consigne_envoye->setText(stringAnswer + "\n" +
display);
    ui->BP_profil_consigne->show();
}

void MainWindow::on_BP_heure_actuelle_2_clicked()
{
    QDateTime now = QDateTime::currentDateTimeUtc();
    ui->LE_starttime_2->setText(now.toLocalTime().toString("yyyy-
MM-dd hh:mm"));
}

void MainWindow::on_BP_actualiser_etat_clicked()
{
    ui->BP_actualiser_etat->hide();
    QString answer = getRest("https://api.energyon.ch/api/asset/
00002HQ/timeserieslast/control-charge-enabled");
    ui->TE_etat->setText(answer);
    ui->BP_actualiser_etat->show();
}

QString getRest(QString stringUrl){

    QString stringAnswer = NULL;
    QUrl url = (QUrl)(stringUrl);

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);

    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray("Bearer " + access_token.toUtf8()));
```

```
    //requête get
    QNetworkReply *reply = mgr.get(request);

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = reply->readAll();
        delete reply;
    }
    else{
         //échec
        qDebug() << "échec demande de la puissance actuelle";
        stringAnswer = reply->readAll();
        delete reply;
    }
    return stringAnswer;
}

void MainWindow::on_BP_activation_clicked()
{
    ui->BP_activation->hide();
    QString body = "{\"tstamp\":\"2017-07-17 17:19:00.00000\",
\"value\":1}";
    putRest("https://api.energyon.ch/api/asset/00002HQ/
timeserieslast/control-charge-enabled", body);
    ui->BP_activation->show();
}

void MainWindow::on_BP_desactivation_clicked()
{
    ui->BP_desactivation->hide();
    QString body = "{\"tstamp\":\"2017-07-17 17:19:00.00000\",
\"value\":0}";
    putRest("https://api.energyon.ch/api/asset/00002HQ/
timeserieslast/control-charge-enabled", body);
    ui->BP_desactivation->show();
}

QString putRest(QString stringUrl, QString body){
    QString stringAnswer = NULL;
    QUrl url = stringUrl;

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);
```

```
    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray("Bearer " + access_token.toUtf8()));
    request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json");

    //requête put
    QNetworkReply *reply = mgr.put(request,body.toUtf8());

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = "Das Profil der Sollwerte wurde
erforlgreich am " + QDateTime::currentDateTime().toString("yyyy-
MM-dd hh:mm") + " gesendet:"+ reply->readAll();
        delete reply;
    }
    else{
         //échec
        stringAnswer = "Sendefehler des Sollwert-Profils" + reply-
>readAll();
        delete reply;
    }
    return stringAnswer;
}
```

Anhang 5:

Smart-Charge Programm

- SmartChargeSystem.pro
- main.cpp
- maininterface.h
- maininterface.cpp
- restclient.h
- restclient.cpp
- electricvehicle.h
- electricvehicle.cpp
- threadupdatevalues.h
- threadupdatevalues.cpp

```
#-------------------------------------------------
#
# Project created by QtCreator 2017-08-07T08:06:41
#
#-------------------------------------------------

QT       += core gui
QT      += network #for QtNetworkManager

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = SmartChargeSystem
TEMPLATE = app


SOURCES += main.cpp\
        maininterface.cpp \
    restclient.cpp \
    electricvehicle.cpp \
    threadupdatevalues.cpp

HEADERS  += maininterface.h \
    restclient.h \
    electricvehicle.h \
    threadupdatevalues.h


FORMS    += maininterface.ui
```

```cpp
#include "maininterface.h"
#include <QApplication>
#include "electricvehicle.h"
#include <QDebug>

MainInterface *externMainInterface=NULL;

int main(int argc, char *argv[])
{


    QApplication a(argc, argv);
    MainInterface w;
    externMainInterface = &w;
    w.show();

    return a.exec();
}
```

```
#ifndef MAININTERFACE_H
#define MAININTERFACE_H

#include "restclient.h"
#include "threadupdatevalues.h"
#include "electricvehicle.h"

#include <QDialog>
#include <QDateTime>

#define MIN_LOADING_POWER 4.14 //kW
#define MAX_LOADING_POWER 22.14 //kW
#define DEFAULT_LOADING_POWER 22.14 //kW
#define FAST_CHARGE 0//charging mode
#define SOLAR_CHARGE 1//charging mode
#define SMART_CHARGE 2//charging mode
#define DEFAULT_CHARGING_TIME 12 //h
#define ELECTRICITY_DAY_PRICE 0.2 //Fr./kWh
#define ELECTRICITY_NIGHT_PRICE 0.15 //Fr./kWh

#define TEST_MODE false //test mode simulates measures



namespace Ui {
class MainInterface;
}

class MainInterface : public QDialog
{
    Q_OBJECT

public:
    explicit MainInterface(QWidget *parent = 0);
    ~MainInterface();

    //functions
    void updateValues();
    void integrateEnergy();
    void startFastCharge();
    void stopFastCharge();
    void startSolarCharge();
    void updateSolarCharge();
    void stopSolarCharge();
    void startSmartCharge();
    void updateSmartCharge();
    void stopSmartCharge();
    void showValues();
    void startWorkInAThread();
    void resetValuesAtStart();
    void resetValuesAtStop();


    //variables
    float setpointMaxLoadingPower, loadingPower,
loadingTotalEnergy, loadingStartEnergy, loadingEnergy,
```

```
loadingEnergyDisplay,
    pvPower, netloadPower, excessPower, loadingCost,
solarLoadingEnergy, dayNetLoadingEnergy, nightNetLoadingEnergy;
    int batteryLevelEstimated, batteryLevelWanted, chargeMode,
ratioSolarLoadingEnergy, timeLeftToCharge;
    bool chargeStarted, setLoadingStartEnergy, tripMode;
    RestClient restClient;
    QDateTime currentDateTime;
    QDateTime chargingEndTime;
    ThreadUpdateValues threadUpdateValues;
    ElectricVehicle ev;

private slots:

    void on_CB_Mode_currentIndexChanged(int index);

    void on_HS_BatteryLevelAtStart_valueChanged(int value);

    void on_HS_BatteryLevelAtEnd_valueChanged(int value);

    void on_PB_StartStop_clicked();

    void on_DI_MaxLoadingPower_valueChanged(int value);

    void timer_1s_event();

    void on_DT_EndTime_dateTimeChanged(const QDateTime &dateTime);

    void on_CB_Trip_toggled(bool checked);

private:
    Ui::MainInterface *ui;


};


#endif // MAININTERFACE_H
```

```
#include "maininterface.h"
#include "ui_maininterface.h"
#include <QDebug>
#include <QMessageBox>
#include <QTimer>

MainInterface::MainInterface(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::MainInterface)//constructor
{
    ui->setupUi(this);
    resetValuesAtStop();
    resetValuesAtStart();

    loadingEnergyDisplay = 0;
    pvPower = 0;
    netloadPower = 0;
    excessPower = 0;
    loadingPower = 0;

    on_CB_Mode_currentIndexChanged(ui->CB_Mode->currentIndex());//
set mode at beginning
    on_HS_BatteryLevelAtStart_valueChanged(ui-
>HS_BatteryLevelAtStart->value());//set default battery level at
start at betinning
    setpointMaxLoadingPower = DEFAULT_LOADING_POWER;//set default
loading power
    chargeMode = 0;
    threadUpdateValues.start();

    //set up timer
    QTimer *timer_1s = new QTimer(this);
    connect(timer_1s,SIGNAL(timeout()),this,
SLOT(timer_1s_event()));
    timer_1s->start(1000);//start timer every 1s

}

MainInterface::~MainInterface()//destructor
{
    delete ui;
}

//timer Event
void MainInterface::timer_1s_event(){
    //display current time
    currentDateTime = QDateTime::currentDateTime();
    ui->LB_Time->setText(currentDateTime.toString("hh:mm:ss"));

    if(currentDateTime.time().second()==2){//every minute at the
second 02
       if(chargeStarted){//if charge started
            switch(chargeMode){
                case SOLAR_CHARGE:
                    updateSolarCharge();
                break;
```

```
                case SMART_CHARGE:
                    updateSmartCharge();
                break;

                case FAST_CHARGE:
                    threadUpdateValues.start();
                break;

                default:
                    threadUpdateValues.start();
                break;
            }
        }
        else{//if charge not started
            threadUpdateValues.start();//updates the values
        }

    }
}

void MainInterface::resetValuesAtStart(){

    loadingCost = 0; //Fr.
    ratioSolarLoadingEnergy = 0;
    solarLoadingEnergy = 0;
    dayNetLoadingEnergy = 0;
    nightNetLoadingEnergy = 0;
    loadingEnergy = 0;
    loadingEnergyDisplay = 0;

    showValues();
}

void MainInterface::resetValuesAtStop(){
        ui->CB_Trip->setChecked(false);//reset Trip mode

        ui->HS_BatteryLevelAtStart->setValue(30);//reset battery
level at start to 30%
        on_HS_BatteryLevelAtStart_valueChanged(30);

        ui->HS_BatteryLevelAtEnd->setValue(90);//reset battery
level at end to 90%
        on_HS_BatteryLevelAtEnd_valueChanged(90);

        chargingEndTime =
QDateTime::currentDateTime().addSecs(3600*DEFAULT_CHARGING_TIME);
// set default charging time
        ui->DT_EndTime->setDateTime(chargingEndTime);

        chargeStarted = false;
        setLoadingStartEnergy = true;

}

void MainInterface::updateValues(){
```

```
    if(setLoadingStartEnergy){//save energy counter value at the
start of loading process
            setLoadingStartEnergy = false;
            restClient.getMeasurementLoadingTotalEnergy(&loadingSt
artEnergy);
            loadingEnergy = 0;
    }
    else{
         restClient.getMeasurementLoadingTotalEnergy(&loadingTotal
Energy);
         if(TEST_MODE&&chargeStarted) loadingEnergy++;//kWh for
test mode
         else loadingEnergy = loadingTotalEnergy -
loadingStartEnergy;

    }

    restClient.getMeasurementLoadingPower(&loadingPower);
    restClient.getPVPower(&pvPower);
    restClient.getNetloadPowerAverage(&netloadPower);
    excessPower = pvPower - netloadPower;
    if(excessPower < 0){
        excessPower = 0;//kW
    }

    showValues();
}

void MainInterface::integrateEnergy(){//must be called exactly
every minute!

    if(chargeStarted){//integrate solar and net energy
        if(loadingPower <= excessPower){//if loading at 100% with
solar electricity
            solarLoadingEnergy += loadingPower/60;
        }
        else if(0 < excessPower){//if loading partially with solar
electricity
            solarLoadingEnergy += excessPower/60;

        }
        //do nothing if no solar electricity at all


        //calculate ration Solar- / Loading-Energy
        if(loadingEnergy > 0){
            ratioSolarLoadingEnergy = (int)(solarLoadingEnergy/
loadingEnergy*100);
        }
        else{
            ratioSolarLoadingEnergy = 0;
        }

        //calculate cost of the charge
        if
```

```
((6<=currentDateTime.time().hour())&&(currentDateTime.time().hour(
)<22) ){//day electricity
            if(excessPower < loadingPower) {//if netpower used
                loadingCost += (loadingPower-excessPower)/
60*ELECTRICITY_DAY_PRICE;
            }
            qDebug() << "day";
        }
        else{//night electricity
            loadingCost += loadingPower/
60*ELECTRICITY_NIGHT_PRICE;
            qDebug() << "night";
        }
    }

    showValues();
}

void MainInterface::showValues(){
    if(chargeStarted){
        loadingEnergyDisplay = loadingEnergy;//to hold the energy
of the last session until a new charge is started
    }
    ui->LB_Information->setText("Loading power: " +
QString::number(loadingPower)+" kW \n" +
                                "Loading energy: " +
QString::number(loadingEnergyDisplay) + " kWh\n" +
                                "Loading cost: " +
QString::number(((int)(100*loadingCost + 0.5))/100.0) + " Fr.\n" +
                                "Ratio Solarenergy: " +
QString::number(ratioSolarLoadingEnergy) + " %\n" );
//                                  "Solar power: " +
QString::number(pvPower) + " kW \n" +
//                                  "Household consumption: " +
QString::number(netloadPower) + " kW\n" +
//                                  "Excess power: " +
QString::number(excessPower) + " kW \n"
//                                  "Max loading power: " +
QString::number(setpointMaxLoadingPower) + " kW");
}

// charge mode functions ------------------------------------
void MainInterface::startFastCharge(){
    ui->DI_MaxLoadingPower->setDisabled("true");
    on_DI_MaxLoadingPower_valueChanged(ui->DI_MaxLoadingPower-
>value());
    restClient.setControlMaxLoadingPower(setpointMaxLoadingPower);
    restClient.setControlChargeEnabled(true);
    showValues();
}

void MainInterface::stopFastCharge(){
    ui->DI_MaxLoadingPower->setEnabled("true");
    //return restClient.setControlChargeEnabled(false);
}
```

```
void MainInterface::startSolarCharge(){

    //the netloadPowerAverage is set as the consumption (just for
testing! the real consumption is not known)
    float powerSurplus = pvPower-netloadPower;

    if(powerSurplus>=MIN_LOADING_POWER){//if enough power
available to charge car
        if(powerSurplus>=MAX_LOADING_POWER)
            setpointMaxLoadingPower = MAX_LOADING_POWER;
        else{
            setpointMaxLoadingPower = powerSurplus;
        }
        //restClient.setControlChargeEnabled(true);
    }
    else{//if not enough power available to charge car
        setpointMaxLoadingPower = 0;
        //restClient.setControlChargeEnabled(false);
    }

    showValues();
}

void MainInterface::updateSolarCharge(){

    updateValues();
    //the netloadPowerAverage is set as the consumption (just for
testing! the real consumption is not known)
    float powerSurplus = pvPower-netloadPower;

    if(powerSurplus>=MIN_LOADING_POWER){//if enough power
available to charge car
        if(powerSurplus>=MAX_LOADING_POWER)
            setpointMaxLoadingPower = MAX_LOADING_POWER;
        else{
            setpointMaxLoadingPower = powerSurplus;
        }
        restClient.setControlChargeEnabled(true);
    }
    else{//if not enough power available to charge car
        setpointMaxLoadingPower = 0;
        //restClient.setControlChargeEnabled(false);
    }
}

void MainInterface::stopSolarCharge(){

}

void MainInterface::startSmartCharge(){
    ui->DT_EndTime->setDisabled(true);
    ui->HS_BatteryLevelAtStart->setDisabled(true);
    ui->HS_BatteryLevelAtEnd->setDisabled(true);
    ui->CB_Trip->setDisabled(true);

    ev.setBatteryLevel(batteryLevelEstimated);
```

```
    int minTimeToCharge =
ev.getTimeToCharge(21.5,batteryLevelWanted);//calculate min time
to charge

    if(minTimeToCharge > timeLeftToCharge){

        QMessageBox::critical(this,"Smart Charge impossible",
("The time limit is impossible to respect! \nRequested charging
time: "+QString::number(timeLeftToCharge/60)+" h
"+QString::number(timeLeftToCharge%60)+" min\nMinimal charging
time: "+QString::number(minTimeToCharge/60)+" h
"+QString::number(minTimeToCharge%60)+" min\nThe time limit will
be modified."));
        QDateTime minChargingEndTime =
QDateTime::currentDateTime().addSecs(minTimeToCharge*60 + 300);
        ui->DT_EndTime->setDateTime(minChargingEndTime);
    }

    updateSmartCharge();
}
void MainInterface::updateSmartCharge(){

}

void MainInterface::stopSmartCharge(){
    ui->DT_EndTime->setEnabled(true);
    ui->HS_BatteryLevelAtStart->setEnabled(true);
    ui->HS_BatteryLevelAtEnd->setEnabled(true);
    ui->CB_Trip->setEnabled(true);
}

// slots
------------------------------------------------------------------
------
void MainInterface::on_CB_Mode_currentIndexChanged(int index)
{
    switch(index){
        case FAST_CHARGE:{//Fast
            ui->setpointMaxLoadingPower->show();
            ui->batteryLevel->hide();
            ui->timeSetpoint->hide();
            ui->information->show();
            ui->previsions->hide();
            break;
        }
        case SOLAR_CHARGE:{//Solar
            ui->setpointMaxLoadingPower->hide();
            ui->batteryLevel->hide();
            ui->timeSetpoint->hide();
            ui->information->show();
            ui->previsions->hide();
            break;
        }
        case SMART_CHARGE:{//Smart
            ui->setpointMaxLoadingPower->hide();
            ui->batteryLevel->show();
```

```
            ui->timeSetpoint->show();
            ui->information->show();
            ui->previsions->show();
            break;
        }
    }
    chargeMode = index;
}

void MainInterface::on_HS_BatteryLevelAtStart_valueChanged(int
value)
{
    batteryLevelEstimated = value;
    ui->PB_BatteryLevelEstimated->setValue(value);
    on_HS_BatteryLevelAtEnd_valueChanged(ui->HS_BatteryLevelAtEnd-
>value());
}

void MainInterface::on_HS_BatteryLevelAtEnd_valueChanged(int
value)
{
    batteryLevelWanted = value;
    if(!tripMode&&(batteryLevelWanted==100)){//do not allow to
exceed 99% if not in tripMode
        batteryLevelWanted = 99;
        ui->HS_BatteryLevelAtEnd->setValue(99);
    }
    if(batteryLevelWanted<=batteryLevelEstimated){
        batteryLevelWanted = batteryLevelEstimated;
        ui->HS_BatteryLevelAtEnd->setValue(batteryLevelEstimated);
    }
    ui->LB_BatteryLevelAtEnd-
>setText(QString::number(batteryLevelWanted)+"%");
}

void MainInterface::on_DI_MaxLoadingPower_valueChanged(int value)
{
    setpointMaxLoadingPower = MIN_LOADING_POWER + 0.5*value;
    int addHalfDigit = (int)(2*(setpointMaxLoadingPower - (int)
(setpointMaxLoadingPower)));//indicates if 0.5 is added after the
rounded value

    if(addHalfDigit==1){//shows the number with a precision of 0.5
        ui->LB_MaxLoadingPower->setText("Max loading power:
"+QString::number((int)setpointMaxLoadingPower+0.5) + " kW");
    }
    else{
        ui->LB_MaxLoadingPower->setText("Max loading power:
"+QString::number((int)setpointMaxLoadingPower) + ".0 kW");
    }
}

void MainInterface::on_DT_EndTime_dateTimeChanged(const QDateTime
&dateTime)
{
    timeLeftToCharge = (int)((dateTime.toMSecsSinceEpoch()-
```

```
QDateTime::currentMSecsSinceEpoch())/60000.0);
}

void MainInterface::on_CB_Trip_toggled(bool checked){
        tripMode = checked;

        if(tripMode){
            QMessageBox::information(this,"Trip mode", "The trip
mode allows you to charge your car at 100% but will take much
longer (+3h) !");
        }

        else {//to reset to 99% if the slider is at 100%
            on_HS_BatteryLevelAtEnd_valueChanged(ui-
>HS_BatteryLevelAtEnd->value());
        }
}

void MainInterface::on_PB_StartStop_clicked()
{
    ui->PB_StartStop->setDisabled(true);//deactivate

    if(!chargeStarted){//click on start button
        ui->CB_Mode->setDisabled(true);//lock mode selection while
charging process
        ui->PB_StartStop->setStyleSheet("background-color:
rgb(255,150,150)");
        ui->PB_StartStop->setText("Stop");
        resetValuesAtStart();
        if(!restClient.validAccessToken){//obtain new access token
if not valid anymore
            restClient.sendAccessRequest();
        }
        switch(chargeMode){
            case FAST_CHARGE:
                startFastCharge();
            break;

            case SOLAR_CHARGE:
                startSolarCharge();
            break;

            case SMART_CHARGE:
                startSmartCharge();
            break;
        }
        chargeStarted = true;

    }
    else{//click on stop button

        ui->PB_StartStop->setStyleSheet("background-color:
rgb(150,255,150)");
        ui->PB_StartStop->setText("Start");

        switch(chargeMode){
```

```
            case FAST_CHARGE:
                stopFastCharge();
            break;

        case SOLAR_CHARGE:
                stopSolarCharge();
            break;

        case SMART_CHARGE:
                stopSmartCharge();
            break;
        }

        ui->CB_Mode->setEnabled(true);//unlock mode selection
after charging process finished
        resetValuesAtStop();
    }
    ui->PB_StartStop->setEnabled(true);//reactivate StartStop
button
}
```

```
#ifndef RESTCLIENT_H
#define RESTCLIENT_H

#include <QString>
#include <QObject>
#include <QDateTime>

struct DATE_AND_VALUE{
    float value;
    QDateTime dateTime;
};

class RestClient: public QObject
{
    Q_OBJECT
public:

    //functions
    RestClient();
    bool sendAccessRequest();
    QString getRest(QString);
    QString putRest(QString, QString);
    bool getProfile(QString, QDateTime, QDateTime, QString,
DATE_AND_VALUE* , int*);

    bool getMeasurementLoadingPower(float*);
    bool getMeasurementLoadingTotalEnergy(float*);
    bool getPVPower(float*);
    bool getNetloadPower(float*);
    bool getNetloadPowerProfile(QDateTime, QDateTime, QString,
DATE_AND_VALUE*, int*);
    bool getNetloadPowerAverage(float*);

    bool setControlMaxLoadingPower(float);
    bool setControlChargeEnabled(bool);

    //variables
    QString access_token;
    QString username;
    QString password;
    bool validAccessToken;

public slots:
    void access_tokenTimerEvent();

private:

};



#endif // RESTCLIENT_H
```

```
#include "restclient.h"
#include "maininterface.h"
#include <QString>
#include <QDebug>
#include <QUrl>
#include <QUrlQuery>
#include <QJsonDocument>
#include <QJsonArray>
#include <QJsonObject>
#include <QEventLoop>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QTimer>
#include <QObject>
#include <QMessageBox>
#include <QDateTime>
#include <QTimeZone>

RestClient::RestClient()//constructor
{
    //initialise variables
    username = "---";
    password = "---";

    access_tokenTimerEvent();

    //set up timer
    QTimer *access_tokenTimer = new QTimer(this);
    connect(access_tokenTimer,SIGNAL(timeout()),this,
SLOT(access_tokenTimerEvent()));
    access_tokenTimer->start(270000);//start timer every 4min and
30s (access_token expires after 5 min)

}

//Basic REST request
-------------------------------------------------
bool RestClient::sendAccessRequest(){

    QString stringAnswer = "";
    QString body = "username="+username+"&password="+password
+"&client_id=direct-access&grant_type=password";
    QUrl url = (QUrl)"https://auth.energyon.ch/auth/realms/lauber-
iwisa/protocol/openid-connect/token";

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
```

```
    QNetworkRequest request(url);

    //ajouter un header
    request.setHeader(QNetworkRequest::ContentTypeHeader,"applicat
ion/x-www-form-urlencoded");

    //requête post avec body
    QNetworkReply *reply = mgr.post(request, body.toUtf8());

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = reply->readAll();
        delete reply;
    }
    else{
         //échec
        QMessageBox::critical(0, "RestClient Error -
sendAccessRequest", "No connection to the plattform\n Check
connection or username and password");
        delete reply;
        return false;
    }
    //créer un objet Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    access_token = jsonObject.value("access_token").toString();

    if(access_token == ""){//if no access_token
                QMessageBox::critical(0, "RestClient Error -
sendAccessRequest", "Failed to obtain valid access_token.\n" +
stringAnswer);
                return false;
    }
    access_token = "Bearer " + access_token;

    return true;
}

QString RestClient::putRest(QString stringUrl, QString body){
    QString stringAnswer = "";
    if(validAccessToken){//to avoid problems if offline
    QUrl url = stringUrl;

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));
```

```
    //requête http REST
    QNetworkRequest request(url);

    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray(access_token.toUtf8()));
    request.setHeader(QNetworkRequest::ContentTypeHeader,
"application/json");

    //requête put
    QNetworkReply *reply = mgr.put(request,body.toUtf8());

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = "Das Profil der Sollwerte wurde
erforlgreich am " + QDateTime::currentDateTime().toString("yyyy-
MM-dd hh:mm") + " gesendet:"+ reply->readAll();
        delete reply;
    }
    else{
         //échec
        stringAnswer = "";
        QMessageBox::critical(0, "RestClient Error - putRest",
"PUT request to the plattform failed");
        delete reply;
    }
    }
    return stringAnswer;
}

QString RestClient::getRest(QString stringUrl){

    QString stringAnswer = "";
    if(validAccessToken){//to avoid problems if offline
    QUrl url = (QUrl)(stringUrl);

    //créer un enentLoop temporaire sur la stack
    QEventLoop eventLoop;

    // quit() pour quitter le eventLoop quand la requête réseau
est finie finished()
    QNetworkAccessManager mgr;
    QObject::connect(&mgr, SIGNAL(finished(QNetworkReply*)),
&eventLoop, SLOT(quit()));

    //requête http REST
    QNetworkRequest request(url);

    //ajouter le header pour l'authentification
    request.setRawHeader(QByteArray("Authorization"),
QByteArray(access_token.toUtf8()));
```

```
    //requête get
    QNetworkReply *reply = mgr.get(request);

    eventLoop.exec(); // bloque la stack jusqu'à ce que finished()
soit appelé

    if (reply->error()==QNetworkReply::NoError){
        //réussite
        stringAnswer = reply->readAll();
        delete reply;
    }
    else{
         //échec
                QMessageBox::critical(0, "RestClient Error -
getRest", "GET request to the plattform failed");
        stringAnswer = "";
        delete reply;
    }
    }
    return stringAnswer;
}

//timer interrupt
void RestClient::access_tokenTimerEvent(){
    validAccessToken = sendAccessRequest();
    qDebug() << "access_token =" << validAccessToken;
}

bool RestClient::getMeasurementLoadingPower(float *loadingPower){
    QString stringAnswer = getRest("https://api.energyon.ch/api/
asset/00002HQ/timeserieslast/measurement-loading-power");

    if(stringAnswer == ""){//error of REST
        return false;
    }
    //analyse Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    QJsonObject powerObject = jsonObject["data"].toObject();
    *loadingPower = (float)
(powerObject.value("value").toDouble());
        if(TEST_MODE) *loadingPower = 21.5;//kWh only in test mode
the real measure will be overwritten
    return true;
}

bool RestClient::getMeasurementLoadingTotalEnergy(float
*loadingEnergy){
    QString stringAnswer = getRest("https://api.energyon.ch/api/
asset/00002HQ/timeserieslast/measurement-loading-totalenergy");

    if(stringAnswer == ""){//error of REST
        return false;
    }
```

```
    //analyse Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    QJsonObject powerObject = jsonObject["data"].toObject();
    *loadingEnergy = (float)
(powerObject.value("value").toDouble());
    return true;
}

bool RestClient::getPVPower(float *pvPower){

    QString stringAnswer = getRest("https://api.energyon.ch/api/
energysystem/108/timeserieslast/pv-power");

    if(stringAnswer == ""){//error of REST
        return false;
    }
    //analyse Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    QJsonObject powerObject = jsonObject["data"].toObject();
    *pvPower = (float)(powerObject.value("value").toDouble());
    return true;
}

bool RestClient::getNetloadPower(float *consumption){
    // bad resolution of energymeter (10kWh) causes a bad power
measure
    QString stringAnswer = getRest("https://api.energyon.ch/api/
energysystem/108/timeserieslast/netload-power");

    if(stringAnswer == ""){//error of REST
        return false;
    }
    //analyse Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    QJsonObject powerObject = jsonObject["data"].toObject();
    *consumption = (float)(powerObject.value("value").toDouble());
    return true;
}

bool RestClient::setControlMaxLoadingPower(float maxPower){
    QString now = QDateTime::currentDateTimeUtc().toString("yyyy-
MM-dd hh:mm:ss.000");
    QString body = "{\"tstamp\":\"" + now + "\",\"value\":" +
QString::number(maxPower) + "}";
    putRest("https://api.energyon.ch/api/asset/00002HQ/
timeserieslast/forecast-loading-power",body); //forecast, is
optional
```

```
    QString StringAnswer = putRest("https://api.energyon.ch/api/
asset/00002HQ/timeserieslast/control-max-loading-power",body);
    if(StringAnswer == ""){
        return false;
    }
    else{
        return true;
    }
}

bool RestClient::setControlChargeEnabled(bool stationOn){
    QString now = QDateTime::currentDateTimeUtc().toString("yyyy-
MM-dd hh:mm:ss.000");
    QString body = "{\"tstamp\":\"" + now + "\",\"value\":" +
QString::number(stationOn) + "}";
    QString StringAnswer = putRest("https://api.energyon.ch/api/
asset/00002HQ/timeserieslast/control-charge-enabled",body);
    if(StringAnswer == ""){
        return false;
    }
    else{
        return true;
    }
}

bool RestClient::getProfile(QString urlPrefix, QDateTime start,
QDateTime end, QString step, DATE_AND_VALUE *answer, int
*arraySize){
    //the seconds are deleted to have whole minutes
    QString url = urlPrefix + "?starttime=" +
QString::number((start.toMSecsSinceEpoch()/60000)*60000) +
"&endtime=" + QString::number((end.toMSecsSinceEpoch()/
60000)*60000) + "&period=" + step;
    QString stringAnswer = getRest(url);

    //create objet Json
    QJsonDocument jsonAnswer =
QJsonDocument::fromJson(stringAnswer.toUtf8());
    QJsonObject jsonObject = jsonAnswer.object();

    //parse Json
    QJsonObject dataObject = jsonObject["data"].toObject();
    QJsonValue valuesValue = dataObject.value("values");
    QJsonArray valuesArray = valuesValue.toArray();

    int dataSize = valuesArray.size();

    if(dataSize>*arraySize){
        QMessageBox::critical(0,"RestClient Error -
getProfile","The there is too much data (" +
QString::number(dataSize) + " datapoints) to fit in the array
passed as parameter. Make array bigger.");
        *arraySize = 0;
        return false;
    }
    *arraySize = dataSize;
```

```
    for(int i = 0; i < dataSize; i++){
        QJsonObject timevalueObject = valuesArray[i].toObject();
        float value =
(float)timevalueObject.value("value").toDouble();

        QString date = timevalueObject.value("tstamp").toString();
        QDateTime dateUTC = QDateTime::fromString(date, "yyyy-MM-
dd hh:mm:ss.00000");
        dateUTC.setTimeZone(QTimeZone::utc());//to avoid the time
zone decalation because data is already in UTC

        answer[i].value = value;
        answer[i].dateTime = dateUTC;
    }
    return true;
}

bool RestClient::getNetloadPowerProfile(QDateTime start, QDateTime
end, QString step, DATE_AND_VALUE *answer, int *arraySize){

    return getProfile("https://api.energyon.ch/api/energysystem/
108/timeseries/netload-power", start, end, step, answer,
arraySize);
}

bool RestClient::getNetloadPowerAverage(float *netloadPower){//
average on 2 hours because of the bad resolution (10kWh) of the
electricity counter, and is only positive (should be changed
soon!)
    int size = 30;//normally 24
    int decalHoraire = 0;
    DATE_AND_VALUE results[size];
    bool ok =
getNetloadPowerProfile(QDateTime::currentDateTime().addSecs(-7140+
decalHoraire),QDateTime::currentDateTime().addSecs(300+decalHorair
e),"PT5M",results, &size);
    //calculate average
    float averagePower = 0;
    for(int i = 0; i < size; i++){
        averagePower += results[i].value;
    }
    averagePower /=size;
    *netloadPower = averagePower;
        if(TEST_MODE) *netloadPower = 5;//kW
    return ok;
    }
```

```
#ifndef ELECTRICVEHICLE_H
#define ELECTRICVEHICLE_H

class ElectricVehicle
{
public:
    ElectricVehicle();

    //public functions
    void reset();
    void setStoredEnergy(float energy);
    void setBatteryLevel(int batteryLevel);
    float charge(float power, int time);
    float chargeProfile(float powerProfile[], int arraySize, int
timestep);
    float getStoredEnergy();
    int getBatteryLevel();
    float getMaxLoadingPower();
    int getTimeToCharge(float power, int batteryLevelAtEnd);
private:
    //private functions
    int calculateBatteryLevel(float energy);
    float calculateMaxLoadingPower(float batteryLevel);
    float calculateCharge(float loadingPowerLimit,float
startEnergy,int time);
    float calculateStoredEnergy(int batteryLevel);

    //private variables
    int evBatteryLevel;
    float evStoredEnergy;

};

#endif // ELECTRICVEHICLE_H
```

```cpp
#include "electricvehicle.h"
#include <QtMath>

ElectricVehicle::ElectricVehicle()//constructor
{
    this->reset();//reset electric vehicle
}


//public functions

void ElectricVehicle::reset(){
    evStoredEnergy = 0;
    evBatteryLevel = 0;
}

void ElectricVehicle::setStoredEnergy(float energy){
    if(energy > 0){
        if(energy <= 100){
            evStoredEnergy = energy;
            evBatteryLevel = calculateBatteryLevel(energy);
        }
        else{
            evStoredEnergy = 25.3;//kWh
            evBatteryLevel = 100;//%
        }
    }
}

void ElectricVehicle::setBatteryLevel(int batteryLevel){
    if((0 <= batteryLevel)&&(batteryLevel <= 100)){
        evBatteryLevel = batteryLevel;
        evStoredEnergy = calculateStoredEnergy(batteryLevel);
    }
    else if(batteryLevel > 100){
        evBatteryLevel = 100;//%
        evStoredEnergy = 25.3;//kWh
    }
    else if(batteryLevel < 0){
        evBatteryLevel = 0;//%
        evStoredEnergy = 0;//kWh
    }
}

float ElectricVehicle::charge(float power, int time){
    float energy = calculateCharge(power,evStoredEnergy, time);
    evStoredEnergy += energy;
    evBatteryLevel = calculateBatteryLevel(evStoredEnergy);
    return energy;
}

float ElectricVehicle::getStoredEnergy(){
    return evStoredEnergy;
}

int ElectricVehicle::getBatteryLevel(){
```

```
    return evBatteryLevel;
}

float ElectricVehicle::getMaxLoadingPower(){
    return calculateMaxLoadingPower(evStoredEnergy);
}

int ElectricVehicle::getTimeToCharge(float power, int
batteryLevelAtEnd){
    float energyAtEnd = calculateStoredEnergy(batteryLevelAtEnd);
    if((power>=(float)4.14)&&(energyAtEnd!=-1)){//can't load under
4.14 kW setpoint limit
        int timeToFullcharge = 0;
        float storedEnergy = evStoredEnergy;
        while(storedEnergy < energyAtEnd){
            storedEnergy += calculateCharge(power, storedEnergy,
1);
            timeToFullcharge++;
        }
        return timeToFullcharge;
    }
    return -1;
}

float ElectricVehicle::chargeProfile(float power[] , int
arraySize, int timestep){
    float energy = 0;
    for(int i = 0; i < arraySize; i++){
        energy += charge(power[i], timestep);
    }
    return energy;
}


// private calculations

float ElectricVehicle::calculateCharge(float loadingPowerLimit,
float startEnergy, int time){
    float energy = startEnergy;
    for(int minute = 1; minute <=time; minute++){
       float loadingPower = calculateMaxLoadingPower(energy);
       if(loadingPower>loadingPowerLimit){//limitate the loading
power if it exceeds the limit
           loadingPower = loadingPowerLimit;
       }
       if(loadingPower > (float)21.5){
           loadingPower = 21.5;//kW
       }
       else if(loadingPowerLimit<(float)4.14){
           loadingPower = 0;//kW
       }
       energy += loadingPower/60;//recalculate new energy
   }

   return (energy-startEnergy);//return energy that has been
loaded
```

```
}

float ElectricVehicle::calculateMaxLoadingPower(float energy){
    float loadingPower;
    if(energy<(float)15.3){//If less than 15.3kWh
        loadingPower = 21.5;//kW
    }
    else if(energy<(float)25.3){//If less than 17.46kWh
        loadingPower = 2 + 19.5*qPow(1-(energy-15.3)/10, 5); //
returns decreasing function
    }
    else {//if battery is full
        loadingPower = 0;
    }

    return loadingPower;
}

int ElectricVehicle::calculateBatteryLevel(float energy){
    int batteryLevel;

    if(energy<=(float)15.3){//for battery level between 0 and 90 %
        batteryLevel = (int)(100*energy/17 +0.0001);//+ 0.0001 to
round. 0.069kWh is the smallest possible step when charging at
4.14kW
    }
    else if(energy<=(float)17.46){//for battery level between 90
and 99 %
        batteryLevel = (int)(100*(energy-15.3)/24 + 90 +
0.0001);//+ 0.0001 to round
    }
    else if(energy<(float)25.3){//for battery level at 99%
        batteryLevel = 99;
    }
    else{//for full battery
        batteryLevel = 100;
    }

    return batteryLevel;
}

float ElectricVehicle::calculateStoredEnergy(int batteryLevel){
    float storedEnergy;
    if(batteryLevel<0){
        return -1;
    }
    else if(batteryLevel <= 90){
        storedEnergy = batteryLevel/100.0*17;

    }
    else if(batteryLevel <=99){
       storedEnergy = (batteryLevel-90)/100.0*24 + 15.3;
    }
    else{//full battery
        storedEnergy = 25.3;
    }
```

```
    return storedEnergy;
}
```

```
#ifndef THREADUPDATEVALUES_H
#define THREADUPDATEVALUES_H

#include <QtCore>
#include <QObject>

class ThreadUpdateValues: public QThread
{
public:
    ThreadUpdateValues();
    void run();
};

#endif // THREADUPDATEVALUES_H
```

```
#include "threadupdatevalues.h"
#include "maininterface.h"
#include <QDebug>
#include <QtCore>


extern MainInterface *externMainInterface;

ThreadUpdateValues::ThreadUpdateValues()//constructor
{
}

void ThreadUpdateValues::run(){
    //here is the code that should be run as a thread
    qDebug() << "Thread started";
    externMainInterface->updateValues();
    externMainInterface->integrateEnergy();//must be called
exactly every minute!
}
```