Energy Management System Implementation for Electrical Vehicle Charging

JONAS KARLSSON

Bachelor of Science Thesis
Stockholm, Sweden 2012

TRITA-ICT-EX-2012:116
Electrical vehicle charging
Energy Management System Implementation

Jonas Karlsson
2012-06-20

Kungliga Tekniska Högskolan
School of Information and Communication Technology (ICT)
Elektronik och Datorteknik 180 hp
Abstract

In the quest for lowering Carbon dioxide emissions and creating a more sustainable future living environment, personal transportation and its cost on both economy and environment is a key part and problem to be solved.

We can witness more and more electrical vehicles on our roads world-wide, and further more and more households are investing in solar energy in the form of Solar Photovoltaic (PV) cells. The arrays combined with the electrical vehicles could possibly cut down carbon dioxide emissions and monthly gas/fuel costs on households.

This report is a Bachelor thesis work at ABB, Corporate research, Västerås, Sweden. For Kungliga Tekniska Högskolan (KTH).

The report will present a prototype software-solution implementation for scheduling the start of electrical vehicle charging, based on parameters such as the current supply of PV Generated energy, price, CO₂ emission and user preferences. A key functionality is to give the household in question the power of choice, to choose when and how their electrical vehicle should be charged, be it with the environment or the wallet in mind.
# TABLE OF CONTENTS

1 INTRODUCTION 4
   1.1 PURPOSE 5
   1.2 SCOPE 5
   1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS 5
   1.4 PEOPLE AND ROLES 6
   1.5 OVERVIEW 6

2 PROTOTYPE DEFINITION AND ARCHITECTURE 7
   2.1 BACKGROUND 7
      2.1.1 Limitations 7
   2.2 EV CHARGING USE-CASE 8
      2.2.1 User experience and interaction 8
   2.3 ARCHITECTURE 9
      2.3.1 System description 9
      2.3.2 Incoming data from outside 10
      2.3.3 Predicting PV generated energy 10

3 SYSTEM MODULES 11
   3.1.1 Logic And Data Collection module 11
   3.1.2 Evaluator Module 14
   3.1.3 PV / EV Data Module 16
   3.1.4 Price and CO2 Module 19
   3.1.5 User Interface 20

3.2 COMMUNICATION AND PROTOCOL 25

4 CONCLUSIONS AND DISCUSSION 27
   4.1 DEVELOPMENT ENVIRONMENT 27
   4.2 KNOWN ISSUES 28
      4.2.1 Data from EV 28
   4.3 TESTING 29

5 FUTURE WORK 30
1 INTRODUCTION

An Energy Management System is based on a home/building automation system and its integration of:

- Electrical Vehicle charging units
- Solar Photovoltaic cells
- Smart appliance
- Local energy storage

The central piece of such architecture is the customer’s home automation system receiving information regarding electricity cost, CO₂ emissions and load reduction / load increase requests. The Energy Management System will, with this information, perform energy management of the household in terms of, e.g. determining an “optimal” start time to charge an electrical vehicle.
1.1 Purpose

This report is a Bachelor thesis work, by Jonas Karlsson, student at KTH, ICT.

The purpose of this document is to provide a comprehensive architectural overview of the designed and implemented solution for scheduling of a start of EV charging. It is intended to capture and convey architectural decisions which have been made during the thesis work. The audience of this document includes ABB, Examiner and Opponents at KTH.

1.2 Scope

This report describes the Energy Management System prototype software including a User Interface. This report does not describe nor include information in depth regarding the services or application supplying data to the prototype.

This report will describe the energy management system implementation for scheduling the start of electrical vehicle charging, based on parameters such as the current supply of PV generated energy; feed-in tariff for solar PV produced electricity and cost of buying electricity. The aim is to optimally utilize the PV Produced electricity considering current electric loads, current electricity feed-in tariff and electricity cost.

The prototype application is intended to work as independent software, but also as a part of a larger application suite.

1.3 Definitions, Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>PV</th>
<th>Photo Voltaic</th>
</tr>
</thead>
<tbody>
<tr>
<td>EV</td>
<td>Electrical Vehicle</td>
</tr>
<tr>
<td>QT</td>
<td>Cross-platform application and UI Framework</td>
</tr>
<tr>
<td>Slot</td>
<td>A QT Term, function executed when called by a Signal</td>
</tr>
<tr>
<td>Signal</td>
<td>A QT Term, function emitted from code, connected to a Slot</td>
</tr>
<tr>
<td>C++</td>
<td>Object oriented programming language</td>
</tr>
<tr>
<td>PHP</td>
<td>Hypertext Preprocessor, server-side script language for web-based applications</td>
</tr>
<tr>
<td>jSon</td>
<td>Data encoding protocol</td>
</tr>
<tr>
<td>GNU/Linux</td>
<td>An open source operating system</td>
</tr>
</tbody>
</table>
1.4 People and roles

<table>
<thead>
<tr>
<th>Person</th>
<th>Roll</th>
<th>Contact information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jonas Karlsson</td>
<td>Author of this report and author of the prototype software</td>
<td><a href="mailto:jonkarls@kth.se">jonkarls@kth.se</a></td>
</tr>
<tr>
<td>Larisa Rizvanovic</td>
<td>Supervisor</td>
<td><a href="mailto:Larisa.rizvanovic@se.abb.com">Larisa.rizvanovic@se.abb.com</a></td>
</tr>
<tr>
<td>Pia Stoll</td>
<td>Project Leader</td>
<td><a href="mailto:pia.stoll@se.abb.com">pia.stoll@se.abb.com</a></td>
</tr>
<tr>
<td>Bengt Molin</td>
<td>examiner</td>
<td><a href="mailto:bengtm@kth.se">bengtm@kth.se</a></td>
</tr>
<tr>
<td>Johannes Thorén</td>
<td>faculty examiner</td>
<td><a href="mailto:jotho@kth.se">jotho@kth.se</a></td>
</tr>
<tr>
<td>Iris Sandin</td>
<td>faculty examiner</td>
<td><a href="mailto:isandin@kth.se">isandin@kth.se</a></td>
</tr>
</tbody>
</table>

1.5 Overview

Part 1 is an introductory part, with explanations regarding the environment which the prototype software implementation resides.

Part 2 will discuss the general idea and provide deep enough conceptual understanding about the actual prototype. Part 2 also provide an overview of the prototype architecture.

Part 3 in this report will take the concepts of modules discussed in Part 2 and explain them in more detail. Part 3 also explains the architecture, prototype and protocols in more detail.

Part 4 consists of the author’s conclusions and discussions.

Part 5 discuss possible future work and ideas.

Appendix consists of API, communication information and file listings for the prototype.
2 PROTOTYPE DEFINITION AND ARCHITECTURE

2.1 Background

In the frame of a smart electricity grid the home and residential building is an important part. The power load on the grid, in the perspective of homes and residential buildings fluctuates depending on time of day and month of the year.

The Energy Management System, presented in part 1 requires input from both outside sources and from the user of the system. With the information provided and the automation the Energy Management System could provide, the power load of the grid can be altered by e.g. scheduling the charging of an electrical vehicle from a time of high power consumption to a time of lower power consumption, in the perspective of the electricity grid.

By scheduling the charging of an electrical vehicle, or any other high power consumption house hold item for that matter, the power grid would see a change in consumption patterns, flattening out the peaks and making it a more stable system. The homeowner would notice not only a lowered cost of electricity but also less CO₂ emissions as a result.

The introduction of solar photovoltaic cells in a home or residential building will further decrease the power load on the energy grid. And it would also decrease both the cost of electricity consumption and CO₂ emissions by either storing the energy produced by the solar PV cells for use in the near future, or charging the electric vehicle directly.

2.1.1 Limitations

The energy management system will be developed to run on an embedded platform. The embedded platform has limited resources such as clock speed and memory size. Code optimization and timings needs to be taken into consideration.

Due to time limitation, 10 week scheduled thesis, the pre-study, development and documentation has to be done within the given time period 2012-04-10 - 2012-06-18.

During development of the software, there have been some difficulty to gain access to certain network-based databases, also some issues has been found with the thin-client-development-environment, this has resulted in lost development time.
2.2 EV Charging Use-Case

As the intended target audience of this product may be “anyone”, the user experience and interaction should not be complicated nor overly informative. A clean and easy-to-use interface with easy-to-understand instructions is needed.

2.2.1 User experience and interaction

A possible user of this system would be presented with the user interface installed at the user premises. On startup the user is presented with options regarding the EV Charging schedule.

- Charge Now
- Environmental
- Best Price

By choosing either *Environmental* or *Best Price* the user is asked to supply a deadline, for example 12 hours during the night before the user needs to drive to work.

The choices the user makes is evaluated by the system and depending on the user preferences a start time and the duration for charging the electrical vehicle is shown on the user interface.

The *Charge Now* option will simply start the charging *now*. A message will be shown if the deadline chosen are to close in time to charge the EV fully.
2.3 Architecture

![Architecture Diagram](image)

**Figure 1, Architectural overview**

2.3.1 System description

The system is a collection of modules, with one or more classes communicating with one another. The modules are depending on the flow of information from the other parts of the system to work properly. Each module will be explained in detail in the following parts of this report.

Incoming data from the outside are shown above as a separate frame (incoming data from outside) in figure 1. These sources of data are accessible via TCP/IP\(^1\) and DBus\(^2\).

---

\(^1\) Transmission Control Protocol – An Internet communication protocol

\(^2\) Interprocess communication Protocol – An Internet communication protocol
2.3.2 Incoming data from outside

For the system to work properly, the information from outside systems and databases are required. The information is then used for calculations and evaluations. An uninterrupted flow of incoming data is critical for the prototype.

Price and CO$_2$ Database provides the following information to the prototype:

- 24 hours in advance electricity cost
- 24 hours in advance CO$_2$ emissions

PV and EV Database provides the following information to the prototype:

- PV Data
  - Current power production
  - Current power consumption
  - Total power production
  - Total power consumption
- EV Data
  - Current state of charge
  - If the EV is Charging
  - If the EV is Connected

The databases contain more information, and the modules responsible for handling and storing the information within them have functionality to access and supply the rest of the system with it. However, the data displayed in the table above is the data that is used in this system, additional functions are implemented for future use.

2.3.3 Predicting PV generated energy

Currently, the system is not receiving data about predicted energy production from the PV Cells$^3$, predicting solar production 24 hours in advance is outside of the scope of this report. However, this is a possible feature for future development.

Without correct data, estimations can be made. Following the logic of the sun, a bell-like curve should be able to represent the production of solar power 24 hours in advance, with the peak of the bell-curve representing the point during the day where there is a maxima of solar power produced.

Following this idea, this information could be used as a scaling factor when calculating the preferred time period of charging by multiplying the inverted values in the scale from 0 to 1 with the incoming (price and CO$_2$ emissions). As this is not fact and real world data, this is not fully supported in the system and is not shown in Fig 1. The system is using “dummy-data” to emulate real world PV Generated energy.

---

$^3$ The local solar panel(s)
3 SYSTEM MODULES

The modules displayed in section 2.2 are groups of classes. These classes are written in either C++ or PHP depending on platform, and will be explained in this chapter, one module at a time.

The modules have specific purposes and usage, for example the Price and CO₂ Data Module, handles the connection to the Price and CO₂ Database, stores and supplies the rest of the system with the data.

The system as a whole is an event based system. QT Slots and Signals are used to handle the flow of execution and the passing of information within the modules.

3.1.1 Logic And Data Collection module

Purpose
The logic and data collection module is the central module, the brain. The purpose of this module is to pass on information between all the other modules. Also to make decisions depending on the information passed. The module is event based and do not take any actions on its own, except during startup and initialization.

The Logic and Data collection module also stores information regarding current preference and session-depending variables such as current cost of electricity, current CO₂ emission caused by bought electricity and EV and PV status.

Some parsing of information, both incoming and outgoing are needed, this is also handled by the Logic and Data Collection module.
Figure 2, Logic and Data Collection module startup.
Usage
This module is the first one to be started, reason being, it is the one which connects everything else in the system.

At startup connection attempts are made to:

- User Interface
- PV / EV Data Server
- Price and CO₂ Data Server

“Dummy” settings and preferences are set as to not confuse the system before real values have been collected from the outside, this to prevent the user to be shown false information.

Upon successful connection to the PV / EV Data Server and Price and CO₂ Data Server, real values are set. And a first run of the evaluator is executed and transmitted to the user interface.

As this module is event based, after initialization, it will listen for signals from the connected modules, and wait.

Functionality
The Logic and Data Collection module supports

- Parsing of data to and from jSon
- Forwarding data to other modules
- Receiving notifications if information is incoming from
  - EV
  - PV Cells
  - Price and CO₂ database

\[^4\] JavaScript Object Notation - data-interchangeformat
3.1.2 Evaluator Module

**Purpose**
The Evaluator module is the calculator of the system. The module requires information and input from the user and PV / EV Database as well as the Price / CO₂ Database.

- Information needed for a successful evaluation of time period is
  - 24 hour in advance electricity price array
  - 24 hour in advance CO₂ emission array
  - Current user preference (Environmental/Best price/Charge now)
  - User set deadline
  - Time duration for charge of the EV

The Logic and Data collection module is responsible of supplying the evaluator with this information, and it is the Logic and Data module who calls the evaluator.

The Evaluator does not calculate new time periods if the conditions are the same as during its last evaluation, this to reduce the work load on the system, the stored last evaluated time period is returned. The evaluator returns a jSon string.
Usage

There are four distinct periods of time that needs to be taken into consideration.

- Time period of 24 hours, the supplied arrays from the Price and CO₂ database.
- Deadline, the user set deadline.
- Duration, a calculated time period with length depending on the state of charge of the EV’s battery.
- Evaluated time period for charging.

The evaluator iterates through the 24 hour time period with block size of duration, one step, hour, at a time until either the user set deadline- or the end of the 24 hour array is reached. Each block of size duration has a startTime and endTime assigned to it.
For each step through the 24 hour array with block size duration, the value (price or CO₂ emission) of the hours are summed and then divided with duration to create a mean value.

\[
mean = \left(\frac{1}{\text{duration}}\right) \times \sum_{n=\text{startTime}}^{\text{duration+startTime}} \text{arrayValue}(n)
\]

This calculation is done at each step, if the newly calculated mean is lower than the previous; the new one is stored as it is considered a better time period. This process is repeated until the end or the user set deadline is reached.

As an example, a duration is calculated to 5 hours, the first passing over the 24 hour array is set to start at 0 and end at 4, the values in the array 0,1,2,3,4 are summed and divided by 5. This value is stored. The next step over the array would start at 1 and end at 5, summing the values in the array 1,2,3,4,5 and so on, if this new mean is lower than the one before it, the old one is discarded and the new one is stored.

When the Cycle has completed through the data, the stored time period is passed to the Logic and Data Collection module for further transmission to the user interface module.

**Functionality**
The evaluator module supports:

- Calculating time periods
- Calculating time it will take to charge the EV.
- Parsing to jSon
- Returning information to other classes:
  - Price mean
  - CO₂ emission mean
  - Highest cost
  - Lowest Cost
  - Highest CO₂ Emission
  - Lowest CO₂ Emission
  - Start time of evaluated time period
  - End time of evaluated time period

### 3.1.3 PV / EV Data Module

**Purpose**
The PV/EV Data module is a connection handler and information holder regarding data from the PV Cells and The EV. The Module connects and communicates with PV and EV Database.
The data on the PV and EV Database-server is updated frequently and is made available to the PV / EV Data module via http requests \(^5\) over TCP Socket.

The PV and EV Database-server will reply, given that the request is valid, with a JSON encoded string containing the information regarding the PV or the EV depending on the request sent to the server. At arrival the information is decoded and stored locally in the module.

The information collected by this module is used both in the user Interface Module and the Evaluator.

\( ^5 \) HyperText Transfer protocol, standard internet text and markup communications protocol
Functionality
The PV / EV Data Module supports:

- 2-way Communication between Logic and Data collection module and eMobility server
- Requesting updated data from eMobility
- Returning information to other classes:
  - Total power production
  - Total power consumption
  - Current Power consumption
  - Current Power production
  - EV state of charge
  - If EV is currently charging
  - If EV is currently parked

6 Local source of PV and EV Data at ABB Corporate research premises
3.1.4 Price and CO₂ Module

**Purpose**
The Price and CO₂ Data Module requests and stores information regarding price and CO₂ emissions, 24 hours in advance.

The module connects to the Price and CO₂ database, where information about price and CO₂ are stored.

![Diagram](image)

*Figure 5. Price and CO₂ Data communication and collection.*

**Functionality**
The Price and CO₂ Module supports:
- Communication between Price and CO₂ Database and Logic and Data Collection Module
- Updating information from Price and CO₂ Database:
  - 24 hours, Price
  - 24 hours, CO₂ Emissions
  - Single values (not in use currently).
3.1.5 User Interface

Purpose and Usage

The graphical user interface is a web-based application. The server hosting the files constructing the interface is a lighthttpd\(^7\) webserver, with PHP installed. This makes it possible to code the interface in php, design it with CSS, automate it with jQuery\(^8\)/java script and display it in a full-screen browser on the touch screen.

The data to be displayed on the user interface are transmitted via DBus from the Logic and Data collection module to the server, where it is dumped into an XML database. This is done frequently at a set time interval.

\(^7\) Light weight linux based webserver
\(^8\) A java script library.
The interface consists of a main index.php file, which contains HTML <div> elements\(^9\). One div for each part of data to be displayed. Each div is at startup loaded with the proper script from the data folder, as an example, the preference div is loaded with the script

\(^9\) Hypertext markup language element.
data/preference.php and the div displaying current power production is loaded with data/currentPowerProduction.php.

Each script reads its specific data from the local database, and depending on the data the script makes decisions of what to write as output on the touch screen, for example the value “false” or “true” for the node <isCharging> is translated to “The car is charging” or “The car is not charging”.

In the index.php there is a hidden div called actionDiv, this div is loaded with the script that sends and reads data to and from the Logic and Data collection module. Each time this script is executed a request is sent, and a response is accepted, the communication is done as jSon encoded strings.

The divs are refreshed and updated at set intervals, these intervals can be changed and optimized in the index.php file. Different data requires different update intervals. Information regarding the cost of electricity the coming 24 hours is updated from the outside once every hour, so refreshing and updating this information locally on the touch screen is not needed as often as information regarding current PV power production and the state of the cars batteries as they should be updated more frequently. In the index.php file timers are set with jQuery for updating divs.

As a synchronization mechanism and filter, a unique session ID is generated and transmitted as the head of each jSon string. Without this session ID in place, the information will be lost.

Also in the index.php file, three buttons are rendered as navigation; the buttons will load three different screens (divs) and are labeled:

- Main
- Settings
- Graphs

The main screen displays information about the car status, and energy production and consumption. Also values regarding price and CO₂ emissions are shown on this screen.

The Settings screen offers the possibility to choose charging preferences, and deadline.

The Graphs screen displays a graph of the upcoming 24 hours, with the current charge scheduled displayed in a slight lighter green color. The values displayed in the graph depends on the user preference set, environmental will display CO₂ emission, and best price will display actual cost.
The main Tab

- Lists:
  - If the EV is currently parked
  - If the EV is currently charging
  - The EV's current state of charge.
  - PV Cells current production
  - Current Price
  - Current CO₂ Emission

The settings Tab

- Set preference to:
  - Charge now
  - Environmental
  - Best Price

- Change deadline

Graphs Tab

- Displays a graph of the upcoming 24 hours, with the current charging scheme marked in different color.

When the user is issuing a command via the interface, be it change of preference or change of deadline a script is executed. Data depending on what the user clicked on is sent to the script as a url GET variable. As an example, when the user chooses to change his or her preference, the script will send data with the current session id to the

---

10 Hypertext transfer protocol variable
Logic and Data Collection module, with a specific command attached to the Logic and Data Collection module, where actions will be taken depending on the attached command.

A chain of events occur as this action is done, as described above the Logic and Data Collection module recognize the command to change preference to Environmental, it is then storing this information in the current Settings class, and the evaluator is launched with the new preference. When the evaluator has done its job a jSon string is returned to the Logic and Data Collection module, where it is assembled into an even bigger jSon string containing information about PV and EV status. As this jSon string is constructed it is sent back to the script on the user interface side, and the database is refreshed, resulting in the updates on the display.

**Functionality**
The user interface module supports:
- Communication between the graphical user interface and the Logic and Data Collection module
- Parsing
- Updating / Rendering information on the display
- Reading XML\textsuperscript{11} data
- Writing XML data

\textsuperscript{11} Markup language for datastorage.
3.2 Communication and Protocol

The modules use DBus inter-process communication system to exchange data. While exchange of data between objects, that are part of the same application are done via QT Slots and Signals.

![Diagram showing communication and protocol between modules.](Image)

*Figure 7, Protocol and communication techniques between modules*
**Logic and Data Collection Module**

- `gotDataToSend()`
- `newData()`
- `newDataFromPvEv()`

**Price and Co2 Data Module**

- `newDataAvailable()`

**User Interface**

- `userPreferenceChange()`
- `forwardFromWS()`
- `dataFromWS()`
- `signalHtmlResponse()`
- `slotJsonRequest()`

**PV / EV Data Module**

- `gotDataToSend()`
- `newData()`
- `newDataFromPvEv()`

**Figure 8, QT Signal and Slot connection map.**
4 CONCLUSIONS AND DISCUSSION

This thesis have been conducted within the assigned time frame, a prototype software implementation has been created and documented.

The prototype handles incoming data and evaluates a suitable period of time within the coming 24 hours, where an EV is to be charged. The prototype is controlled by a user with an easy to use interface that also displays overview information about the users PV Cells, EV and charging schedule.

A working compiled version with source code is being supplied with this report to ABB, for possible future use.

4.1 Development Environment

- Operating systems
  - Microsoft Windows
  - Gentoo Gnu/Linux

- Editors and IDE
  - Microsoft Visual Studio 2011
  - Notepad++
  - Vim

- Server applications
  - lighthttpd

Main development of C++ code conducted in Microsoft Visual studio, on a windows 7 emulated WM-ware thin client.

The user interface and emulation of the touch screen, is a lighthttpd webserver running on a Gentoo GNU/Linux distribution connected to the windows 7 with DBus.
4.2 Known Issues

This section will note, and discuss known issues in the system, with the prototype and development environment.

4.2.1 Data from EV

The data transmitted from the EV to the PV and EV Database-server is not correct, data such as state of charge and isConnected is always set to default, 0. This is a known bug. For now the data is taken care of as 0, and where it is needed a dummy variable is set, for example the duration it would take the EV to charge it’s batteries is set to a static time, instead of calculated, because of this bug.

4.2.2 Time

As there are several systems involved, running on separate computers / servers. “current time” is an important factor. Time depending functions are collecting data and issuing actions, if there is a slight difference in time incorrect data might be accessed and resulting in incorrect decisions regarding charging of the EV.

Example, let’s assume local time on the device running the Logic and Evaluator is 12:01 this means that a new value to the data arrays holding CO₂ emission and price information should be updated, however, if the time differs by just a minute or two on the computer the database is running on, incorrect data will be transmitted.

This can be fixed with a synchronization class, this is not implemented and is discussed in part 5.
4.3 Testing

Testing of the software has been conducted during development on Windows 7 and on Gentoo.

Longer period of stress-testing has not been conducted, and memory leaks may occur.
5 FUTURE WORK

Prediction of solar power
As this prototype does not receive information regarding predicted solar power generated electricity, there is no such implementation written.

However, a data structure similar to the one being handled by the prototype regarding electricity cost and CO₂ emission has been introduced, and is being used in the evaluator module. This is included as an easy approach to future development.

Implementation of unhandled Slots and Signal
The prototype is as previously stated an event driven system, with QT Slots and Signals. There are a few loose-ends in the code at this moment, being slots that are called and executed but “does nothing”. Most commonly, notifications from module A to module B. This feature has been implemented but are not in use, this can be used in possible future development.

Time
As this is a time-critical system in the sense of charging an EV depending on price and CO₂ emissions, the time needs to be correct, synched, on systems that this prototype is connected to, e.g.;

- System hosting the Price and CO₂ database
- System hosting the PV and EV database

The reason this is written here, is that during development, issues like this has occurred, and incorrect results regarding time period evaluation has been made by the system.

User Interface
The user interface is a mockup of what it could look like in the future. If the prototype is included in a larger application suite, the design should be altered to look like the rest of the applications on the display.

The updating of information on the display should be optimized for the hardware, this to make it run smooth but still not requiring too much resources resulting in latency issues.

Scheduling
A scheduling module should be added with the simple purpose of keeping time and emitting a signal to the PV / EV Data Module. The PV / EV Data Module supports and has functionality implemented for sending commands to the PV and EV Database-server, which will then issue the command to start and stop the charging of the EV.