EVLibSim: A Tool for the Simulation of Electric Vehicles' Charging Stations Using the EVLib Library

Emmanouil S. Rigas[a], Sotiris Karapostolakis[a], Nick Bassiliades[a], Sarvapali D. Ramchurn[b]

[a]*Aristotle University of Thessaloniki, Thessaloniki , 54124, Thessaloniki, Greece*
{erigas, skarapost, nbassili}@csd.auth.gr
[b]*Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK,*
sdr1@soton.ac.uk

Abstract

Electric Vehicles (EVs) are considered an efficient alternative to internal combustion engined ones, aiming to reduce global CO_2 emissions. In the last years, EVs are entering the market in an increasing pace. In contrast to conventional cars, EVs have a more complicated recharging procedure. Thus, the development of tools for the efficient simulation of the charging of large numbers of EVs is critical. In this vein, EVLibSim is a tool for the simulation of EV activities at a charging station level. EVLibSim unifies EVLib's primary functions such as the charging and dis-charging of batteries, battery swapping, as well as parking/inductive charging. EVLib is a Java library that provides a simple, yet efficient framework for the management of a number of Electric Vehicle (EV) activities, at a charging station level, within a Smart Grid. EVLibSim provides a great variety of configuration options such as the types and number of chargers, the available energy, the waiting queues, etc. Furthermore, through plots and overview dashboards each user can supervise the operation of the tool in real time. Both EVLib's and EVLibSim's efficiency and scalability have been tested in realistic scenarios, while the correctness and safety of the code have been verified using state of the art tools. Finally, the user experience of the EVLibSim has been evaluated and improved through a detailed user evaluation.

Keywords: `Electric Vehicle, Java Library, Simulation, Charging, Dis-charging, Battery Swap, Parking, Smart Grid`

1. Introduction

Electric vehicles (EVs) are entering our daily lives fast. It is estimated that more than 1 million EVs have already been deployed worldwide, while the target set by the International Energy Agency is 30% market share for fully electric vehicles by 2030 [18]. However, in order to ensure that the large-scale deployment of EVs results in a significant reduction of CO_2 emissions, it is important that they are charged using energy from renewable sources (e.g., wind, solar). Crucially, given the intermittency of these sources, mechanisms as part of a Smart Grid [14], need to be developed to ensure the smooth integration of such sources in our energy systems. EVs could potentially help by storing energy when there is a surplus, and feed this energy back to the grid when there is demand for it [20].

Indeed, the ability of EVs to store energy while being used for transportation [23] represents an enormous potential to make energy systems more efficient. On the one hand, given that vehicles drive only for a small percentage of the day, and considering the fact that EVs are equipped with large batteries, they could be used as storage devices when parked (i.e., as part of Vehicle-to-Grid (V2G) schemes [20]), and thus dramatically increase the storage capacity of the network. On the other hand, given that large numbers of EVs need to charge on a daily basis, (e.g., 40% of EV owners in California travel daily further than the range of their fully charged battery [24]) if EVs charge as and when needed, they may overload the network. For this reason, new scheduling mechanisms are required to be able to manage the charging of EVs –Grid-to-Vehicle (G2V)– while considering the constraints of the distribution networks within which EVs need to charge [29]. Similarly, battery swap schemes which can be used instead of simple battery charge and can minimize waiting times at the charging stations [27], [28], must also be efficiently designed and smoothly integrated to the grid.

Advanced Artificial Intelligence (AI) techniques (e.g., mathematical programming based optimization, electronic markets and coalition formation) have been proven to be efficient in solving EV-related problems, and a large number of such solutions already exists [30]. However, one of the key findings of [30] is that interoperability between various technologies and techniques is missing, and it is vital for successful large scale EV deployments. Thus, there is a need EV technologies to be able to work seamlessly and efficiently together. Different types of chargers should be able to work with all EV models, and data exchanged between entities (EVs, charging points, network operators) should have an understandable by all format and meaning.

EVLib library tries to solve such problems, as it provides simple, yet efficient, tools for the management of the basic EV activities (i.e., charging, dis-charging, battery swap) in a charging station level. On top of this, the simulation framework called EVLibSim applies all major features of EVLib. EVLibSim unifies the Smart Grid benefits along with electric vehicles needs and characteristics, giving the users the ability to simulate many operations of electric vehicles such as charging or discharging, alongside the use of some advanced Smart Grid techniques, such as pricing policies.

Another observation of [30] is that most of the research works in the Electric Vehicle and the Smart Grid sector are developed from scratch by the researchers. This leads to works that are difficult to be re-developed and cross-evaluated by other researchers in the community. The EVLib library aims to be used as a base for the development of sophisticated EV-related algorithms. In this way, the core structure of the code will be the same and only the logic will have to be programmed separately. Thus, the interoperability between various works will increase. Moreover, the EVLibSim tool can be useful in two main ways: 1) The tool can receive as input an EVs' charging or discharging plan, execute the plan and then provide the results in csv files. In this way, the output of many different algorithms can be given to the tool and the results can then be easily evaluated and compared to each other. This is crucial in the direction of comparing different algorithms, which as discussed previously, is missing in the literature. 2) Due to the wide range of researchers dealing with EV-related problems (e.g., computer scientists, electrical engineers, civil engineers, mechanical engineers) the EVLibSim tool gives them the opportunity to construct and compare different settings without the need to write code and develop their own implementations. This can engage more people in doing research in this interdisciplinary sector. Apart from the scientific sector, EVLibSim can help real world deployments of EV charging stations being simulated and evaluated with different configurations. This can have significant economic gains for the charging stations owners.

The rest of the paper is organized as follows: Section 2 presents a review of a number of vehicle-related simulation frameworks. Section 3.1 presents the mathematical formulation of the setting and Section 3.2 presents an overview of EVLib as well as a detailed description of its main classes and functions. Moreover, Section 4 introduces the EVLibSim simulation tool and describes its functionality. Section 5 provides details about the code security and safety testing of EVLib and EVLibSim and the user evaluation of EVLibSim. At the same time, Section 6 presents a detailed usage example of EVLibSim. Finally, Section 7 concludes and Section 8 provides ideas for future work.

2. Related Work

After a thorough study of the existing literature we realized that a number of frameworks concerning the simulation of different aspects of vehicle travelling and modelling exist. We distinguish two major categories: 1) Transportation-related simulation tools and 2) EV-related simulation tools and we present representative works of both categories in what follows.

2.1. Transportation-related simulation tools

SUMO [21] is an open traffic simulation suite. SUMO allows modelling of intermodal traffic systems including road vehicles, public transport and pedestrians. It contains a number of supporting tools which handle tasks such as route finding, visualization, network import and emission calculation. SUMO can be enhanced with custom models and provides various APIs to remotely control the simulation.

Moreover, MatSim [17] is a framework for large-scale, agent-based transport simulations. Each agent has a transport demand represented by a chain of activities it has to perform during one day at different times and locations. The decisions on how to travel between places to perform these activities are planned before the mobility simulation. It is an open source project developed in Java. A user can design a detailed network including millions of vehicles. MatSim provides the ability for demand-modelling in parallel with a controller to start, stop, re-plan or iteratively run the simulations. MatSim has been used in a number of occasions such as in [25] for transport energy demand modelling, or in [36] to increase the transferability of transport demand models.

In a slightly different domain, VIVUS (Virtual Intelligent Vehicle Urban Simulator) [15] simulates vehicles and sensors, taking into consideration their physical structure, while applying artificial intelligence algorithms such as platoon solutions [13] and obstacle avoidance techniques [16]. The urban vehicle simulator is based on two main engines: one for physics simulation and one for 3D immersion in a topological environment. These engines allow a precise simulation of vehicle dynamics and a wide range of physical sensors. This simulator has been successfully used as a prototyping tool for sensor designing and positioning. Experiments presented in this paper show simulation results close to reality. The differences with the real conditions are essentially due to the precision of the world topology and the model chosen for the wheel/road contact.

Finally, battery autonomy is the major pursuit of [32]. They propose an on-board assembling energy system, particularly a photovoltaic system (PVS). A PVS includes a solar array, batteries, regulator and load. A number of variables (e.g., radiance and ambient temperature) are essential inputs to estimate the power production of a PVS. A mobile system is in continuous movement, therefore an additional analysis in changing conditions, due to the variability of the solar exposure, had to be done. They present ways of measuring solar radiation, which define the power production of the system. The simulation tool was implemented using Matlab Simulink. Given that the path followed by a mobile PVS is known, the Simulink tool calculates both the energy produced by PVS and the energy consumption.

2.2. EV-related simulation tools

AVL CRUISE [1] is a tool to analyze fuel efficiency, vehicle emissions, energy transmission modeling, controller design, and vehicle electrification. In the case of EV design, the most important part is the controller design to determine the optimal operating point for minimizing energy consumption. It facilitates this task in a MATLAB/Simulink environment. In a similar vein, EMCAS [4] (Electricity Market Complex Adaptive System) is an agent-based simulation tool to analyze complex power systems. EMCAS can simulate electric vehicle technologies, renewable energy generation subsystems and their integration, as well as thermal power generation. It is used to analyze the external technical, operational and economical impacts on the electricity sector where the simulation is carried out on an hourly basis over a user-specified timescale.

Moreover, Caspoc [2] simulates electric power generation, conversion, distribution, power electronics and mechatronics systems. CASPOC is also useful for the design and analysis of electric vehicles. It can model the entire drive cycle (e.g., gearbox, brakes, wheel slip, etc.) of an EV, the electric drive control, and thermal models of electric machines. It can also help to design and analyze grid connections from renewable energy sources (e.g. solar and wind) and the distribution grid.

FASTSim [5] is a vehicle simulation tool to analyze and design conventional vehicles and EVs. FASTSim evaluates the impact of technology upgrades on EV performance, battery lifetime and overall efficiency. For example, it can calculate the most cost-effective battery size for EVs. It deals with the vehicle powertrain, regenerative braking, energy management strategy, cost estimation, and battery life.

The tools presented so far are not designed solely for EV modelling and simulation, but this is a subset of their features. However, tools related exclusively to EVs exist. For example, [10] presents a framework for modelling EVs' operations and services. The authors developed a simulation platform that is based on the SUMO (vehicular traffic simulator) [21] and OMNET++ (network simulator) [33] and can simulate system and communication protocols for better usage of city-wide mobile services. They evaluated the operations of EVs on a large-scale scenario, by analyzing the effectiveness of the charging reservation process and the impact on the Smart Grid. The second contribution is the Mobile Application Zoo sandbox. Electric Mobility-related mobile applications can be seamlessly combined within the simulation framework in order to be tested on virtual environments before their distribution in real situations.

In addition, [12] study the use of EVs as a distributed energy resource (DER). They propose a fleet aggregator model using a stochastic formulation of DER-CAM [3], an optimization tool used to address DER investment and scheduling problems. This is used to assess the impact of EV interconnections on optimal DER solutions considering uncertainty in EV driving schedules. Results indicate that EVs can have a significant impact on DER investments. Furthermore, results suggest that uncertainty in driving schedules has little impact on total energy costs. In a similar vein, [35] propose a framework for microgrid planning with EV charging demand to find the most economical configuration through which to maximally utilize renewable generation. The algorithm uses a renewable generation-following EV charging scheme and HOMER [6]. Through simulations, they show that the microgrid constructed by the proposed algorithm reduces the investment cost and CO2 emission.

Moreover, V2G-Sim [7] (Vehicle-to-Grid Simulator) addresses vehicle-to-grid integration by incorporating driver-specific sub-modules, charging systems and vehicle powertrains. The major applications are: automotive (e.g., battery, vehicle powertrain), electricity grid (future grid, charging and discharging, renewable generation, electricity market etc.), policy and business (e.g., regulatory, economic), and end users (e.g., V2G-Sim mobile app). It addresses the potential barriers and uncertainties in vehicle-to-grid integration through a systematic quantitative method. It predicts the real-world scenario (drive system, trip distance and time) of an EV and analyzes the impact or opportunity of these vehicles to the electricity grid. To analyze impact on the grid, it models the charging and discharging control approach and driving behavior of an individual EV. V2G-Sim designs the vehicle-to-grid energy exchange system and analyzes it in second-by-second time steps.

In addition, [11] demonstrates step-by-step mathematical calculations and simulation modeling of EVs. The simulation model is equipped with recuperation capability of EV. Its performance has been investigated by simulation results for various vehicle velocity inputs and summarized in terms of vehicle range and Battery State of Charge (SOC). Furthermore, vehicle range and SOC are evaluated and summarized with variation of input parameters at the end of this paper.

In a similar vein, [8] proposes a modelling of electric and parallel-hybrid electric vehicle using Matlab/Simulink environment and the authors focus on different aspects of the vehicle such as engine power, type and size of the battery or weight and try to observe how changes can affect the performance and the distance travelled. The model was simulated in order to obtain the electric vehicle's autonomy. Through the use of a Geographic Information System together with a mathematic algorithm based on genetic algorithms the planning of charging stations was obtained aiming to minimize cost and maximize service for the customers.

In a slightly different domain, where the aim is the optimal placement of charging stations, [34] presents a simulation model that uses mathematical optimization techniques to determine the optimal location of electric vehicle charging facilities aiming to maximize their use by privately owned EVs. Applying this model to the central-Ohio region, the authors demonstrate that a combination of level-one and level-two chargers is preferable to level-two chargers only. They further explore interactions between the optimization criterion used and the budget available and they show that although the optimal location is sensitive to the specific optimization criterion considered, overall service levels are less sensitive to the optimization strategy.

Finally, in [9] the proposed model of an electric vehicle charging station is suitable for the fast DC charging of multiple electric vehicles. The station consists of a single grid-connected inverter with a DC bus where the electric vehicles are connected. The control of the individual electric vehicle charging processes is decentralized, while a separate central control deals with the power transfer from the AC grid to the DC bus. The electric power exchange does not rely on communication links between the station and vehicles, and a smooth transition to vehicle-to-grid mode is also possible. Simulations are performed in Matlab/Simulink to illustrate the behavior of the station.

Table 1 summarizes the tools presented in this subsection and compares them based on a set of criteria. After studying and comparing these tools and considering the findings in [22], we argue that a simulation tool regarding EV activities at a charging station level that considers all the aspects of both the EV and its interconnection to the smart grid is missing. Thus, we argue that EVLibSim is an innovative tool and greatly important for the research community, as well as for the industry.

In the next section, the problem formulation as well as the EVLib library that supports the creation, configuration and handling of such a charging station are presented.

Cit.	**Vehicle modelling/ analysis**	**Charging station modelling/ analysis**	**V2G**	**Electricity pricing**	**Electricity scheduling**	**Renewable energy**	**Commercial/ free**	**Technology used**
EVLibSim	Modelling	Yes	Yes	Yes	Partially	Yes	Free	Java
[1] AVL CRUISE	Yes	No	Yes	No	No	No	Commercial	MATLAB/ Simulink
[2] CASPOC	Yes	No	Yes	Yes	No	Yes	Commercial	No info
[4] EM-CAS	No	No	Yes	Yes	Yes	Yes	Commercial	No info
[5] FAST-Sim	Yes	No	Yes	No	No	No	Free	Excel and Python
[10]	Yes	No	No	Yes	Yes	Yes	Free	OMNET++ and SUMO
[12]	Yes	No	Yes	Yes	Yes	Yes	Free	DER-CAM
[35]	Modelling	No	No	Yes	Yes	Yes	Free (restricted)	HOMER
[7] V2G-Sim	Modelling	No	Yes	Yes	Yes	Yes	Free (restricted)	No info
[34]	Modelling	Yes	No	No	Yes	No	Free	No info
[8]	Yes	Yes	No	Yes	No	No	Free	MATLAB/ Simulink
[9]	Yes	Yes	Yes	No	Yes	No	Free	MATLAB/ Simulink
[11]	Yes	No	No	No	Yes	No	Free	MATLAB/ Simulink

Table 1: Comparison of EV modelling / simulation tools

3. Setting Formulation and the EVLib Library

In this section, we describe the mathematical formulation of the problem and later we describe how this is translated into the EVLib library.

3.1. Setting formulation

In our setting, we study the modeling and simulation of an EV charging station. The station holds charging infrastructure, as well as electric energy and the EVs announce their requests for charging or discharging over time. Later, the station decides on whether it will service each EV based on energy and charger availability. A large number of parameters (e.g., the types and numbers of chargers, as well as the energy sources and the available energy from each source) can be configured based on the scenario that the user needs to simulate. The time $t \in T \subseteq \mathbb{R}$ is continuous and is measured in seconds or minutes and the energy $e \in \mathbb{R}$ is measured in Wh or kWh.

We denote a charging station $i \in I \subseteq \mathbb{N}$. Each charging station has a number of slow chargers $s_i \in S_i \subseteq \mathbb{N}$, fast chargers $f_i \in F_i \subseteq \mathbb{N}$, battery exchange handlers $eh_i \in EH_i \subseteq \mathbb{N}$, inductive chargers $ind_i \in IND_i \subseteq \mathbb{N}$, dischargers $d_i \in D_i \subseteq \mathbb{N}$ and parking slots $prk_i \in PRK_i \subseteq \mathbb{N}$. In more detail, there are two main categories of EV chargers, namely the slow and the fast chargers. The slow chargers can fully charge an EV's battery in 6-8 hours, while the fast chargers in around 1 hour. However, fast charging increases the load on the grid and reduces the life-time of the battery. Apart from the traditional chargers that use a cable, inductive charging also exists. In this case, the EV simply stops on top of an inductive charger and the energy transfer takes place contactless. In addition, battery exchange is an efficient alternative to battery charging, where instead of charging the EV's battery, a fully charged one is swapped into the EV. This technique leads to very fast recharging, but it is costly and raises the problem of battery ownership, as once an EV unloads the battery it may never get the same battery again. In order to support this function, the station has a number of batteries $b_i \in B_i \subseteq \mathbb{N}$. Each battery has a maximum capacity $e_{b_i}^{max}$ and a current charge level $e_{b_i}^{cur}$. Finally, the dischargers support the discharging of an EV's battery when there is excess energy in it.

Now, the slow chargers have a charging rate $s_i^{rate} \in \mathbb{R}$, the fast chargers $f_i^{rate} \in \mathbb{R}$ (i.e., $s_i^{rate} < f_i^{rate}$), the inductive chargers $ind_i^{rate} \in \mathbb{R}$, the dischargers have a discharging rate $d_i^{rate} \in \mathbb{R}$ and the battery exchange handlers have an exchange duration $eh_i^{dur} \in \mathbb{R}$. In addition, we denote binary variables $avlb_{s_i,t} \in \{0,1\}$, $avlb_{f_i,t} \in \{0,1\}$, $avlb_{eh_i,t} \in \{0,1\}$, $avlb_{ind_i,t} \in \{0,1\}$, $avlb_{d_i,t} \in \{0,1\}$, $avlb_{prk_i,t} \in \{0,1\}$ that capture whether the equivalent chargers, exchange handlers, inductive chargers, dischargers, or parking slots are available at time point t.

Moreover, the charging station has available energy $e \in E$ for EV charging from a set of one or more of the following energy sources: Solar energy $e_{i,t}^{sol} \in \mathbb{R}$, wind energy $e_{i,t}^{wind} \in \mathbb{R}$, wave energy $e_{i,t}^{wave} \in \mathbb{R}$, geothermal energy $e_{i,t}^{geo} \in \mathbb{R}$, hydroelectric energy $e_{i,t}^{hydro} \in \mathbb{R}$, and energy from other non-renewable sources $e_{i,t}^{nonR} \in \mathbb{R}$. Note that available energy may vary over time based on EV consumption. Finally, a price for each charging and discharging type and per energy unit is defined: $p_{i,t,s} \in \mathbb{R}$ for slow charging, $p_{i,t,f} \in \mathbb{R}$ for fast charging, $p_{i,t,eh} \in \mathbb{R}$ for battery exchange, $p_{i,t,ind} \in \mathbb{R}$ for inductive charging, and $p_{i,t,d} \in \mathbb{R}$ for discharging.

Apart from the charging station, a number of EVs $j \in J \subseteq \mathbb{N}$ also exist. Each EV has a battery capacity $e_j^{cap} \in \mathbb{R}$, a remaining amount of energy $e_{j,t}^{rem} \in \mathbb{N}$, an energy demand $e_{j,t}^{dem} \in \mathbb{N}$ and a request for a particular type of charging or discharging, as well as a maximum waiting time t_i^{max} for it to be serviced. The acceptance of the charging or discharging of an EV depends on the availability of energy and chargers or dischargers. Once an EV makes a request to the station, an equivalent event $evnt \in EVNT \subseteq \mathbb{N}$ is created (i.e., $evnt_{i,j,s}$, $evnt_{i,j,f}$, $evnt_{i,j,d}$, $evnt_{i,j,eh}$, $evnt_{i,j,ind}$, $evnt_{i,j,p}$). For the handling of waiting events, a number of equivalent lists $levnt \in LEVNT$ exist (i.e., $levnt_{i,s}$, $levnt_{i,f}$, $levnt_{i,d}$, $levnt_{i,eh}$, $levnt_{i,ind}$, $levnt_{i,p}$). For each event in any list, a waiting time t_{levnt}^{wait} can be calculated based on the number of events in the list and the execution time of each one. The lists operate in a FIFO (First In First Out) manner. However, the user has the ability to manually override the order. Finally, each event has a current state $evnt_{i,j,d}^{status} \in \{\text{arrived}, \text{accepted}, \text{rejected}\}$.

In the next section, we describe the EVLib library.

3.2. The EVLib library

EVLib[1] is implemented in the Java programming language and its main goal is to simulate the charging, discharging, battery swap and inductive charging functions in the level of a single charging station i [19]. There

[1]http://intelligence.csd.auth.gr/files/systems/EVLib/evlib.jar

are four main functions, as well as a number of secondary ones, while each function is executed in 2 phases, namely the pre-processing and the execution phase. Figure 1 provides the library's detailed class diagram.

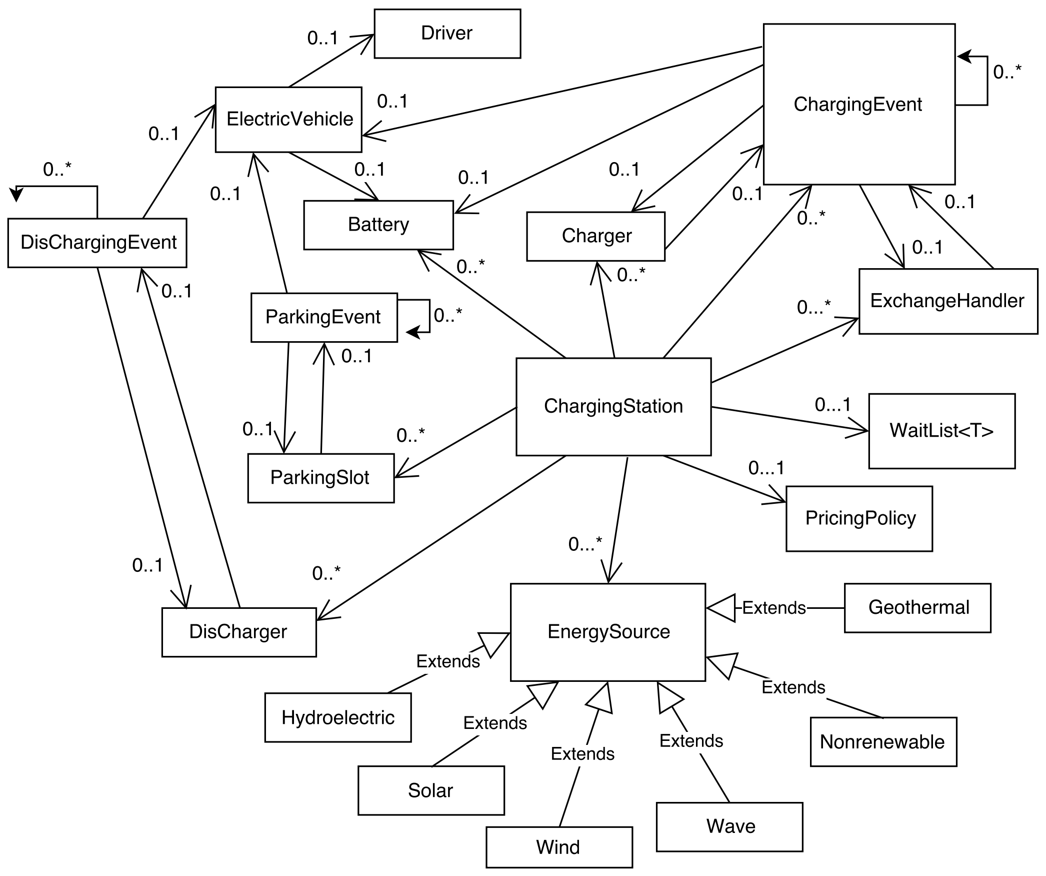


Figure 1: EVLib Class Diagram

3.2.1. EVLib's main functions

The main functions are the following:

Charging (G2V technology): There are 2 types of charging depending on the charging time, namely the fast and the slow charging. Equivalently, slow s_i and fast f_i chargers exist. It is possible to define the rate for each type of charging (i.e., s_i^{rate} and f_i^{rate}). The pre-processing phase requires a request for an empty charger and available energy. If the pre-processing phase is successful, the execution phase can begin.

Battery swap: The pre-processing phase searches for a battery with enough range $e_{b_i}^{cur} \geq e_{j,t}^{dem}$ and a battery swapping handler eh_i to be available in the charging station. If those requirements are met, the execution function can be called and the battery is swapped into the EV.

Discharge (V2G technology): Similarly to a charging event, the pre-processing phase makes a request for an empty discharger d_i. If this phase is successful then the execution can start.

Parking/Inductive charging (G2V technology): The parking function can be merged with the inductive charging. Each parking slot is able to support inductive charging. The user either demands to park, or to park and charge inductively. The pre-processing method searches for an available parking slot prk_i and an inductive charger ind_i. If there is one, then looks for available energy in case of inductive charging. A successful pre-processing phase leads to calling the execution method.

Algorithm 1 describes the steps for the handling of a slow charging event. Initially the charging station receives the request for charging from an EV j. Once the request has been received, the equivalent charging event $evnt_{i,j,s}$ is created and its state is set to *arrived* (line 1). After the event has been created, the station checks whether the type of charger needed for the requested charging (i.e., slow charger in this case) exists. If no such charger exists, then the event is rejected (lines 2-3). If the necessary infrastructure exists, the charging time $t_{i,j}^{chrg}$ is calculated based on the energy demand $e_{j,t}^{dem}$ and the charging rate s_i^{rate} of the charger. If the charging time is longer than the maximum time the EV can stay at the station, the event is rejected (lines 4-8). Then the availability of charger and energy to start the charging immediately is checked (lines 9-12). If both charger and energy exist, then the price to pay for the charging p_j is calculated and the event state is set to *accepted* (lines 13-17). If no charger and/or no energy is available for immediate charging, then the scheduling of the event for later charging is considered. Initially, an available charger for time t_i^{chrg} and for the time window $t_{cur} + t_j^{max}$ (t_{cur} is the current time point and t_j^{max} is the latest time point EV j can be at the charging station) is searched. If a charger available for at least the number of time points needed to complete the charging is found, then a search for the availability of energy takes place. If enough energy is found, then the charging can take place, the event is added to a waiting list $levnt_{i,s}$ and the event state is set to *accepted* (lines 18-30). If no charger or no energy can be found, then the event is *rejected* (lines 31-34). Finally, the state of the event is returned.

Regarding the complexity of Algorithm 1, the most time consuming activities are the search of available chargers and energy. Note that the search is a simple access to a number of cells in the equivalent tables. Given the flow of actions as described in Algorithm 1, the case where the searches take place more times is when an EV is scheduled for later charging. In this case, lines 1, 2, 4-12, 20-30 are activated. The cost is calculated as follows:

$$Cost = 1{+}1{+}1{+}1{+}1{+}t_{i,j}^{chrg}{+}t_{i,j}^{chrg}{\times}|E|{+}1{+}t_{j}^{max}{\times}2{+}1{+}1{+}t_{j}^{max}{\times}|E|{+}1{+}1{+}1 \quad (1)$$

Note that the term $t_j^{max} \times 2$ refers to the lines 21 and 22. Given that the lines with cost equal to 1 do not play a significant role in the overall complexity, Equation 1 becomes:

$$Cost = (1 + |E|) \times t_{i,j}^{chrg} + (2 + |E|) \times t_j^{max} \quad (2)$$

All the parameters in Equation 2 do not scale and are upper bounded and for this reason the worst case complexity of Algorithm 1 is $O(1)$. The algorithms and equivalent complexity analysis for the rest of the event types are similar and are skipped. Note that in the case of discharging, no need for energy exists and in the case of battery swapping, instead of available energy, an available battery must be found.

3.2.2. EVLib's secondary functions

The library also supports a number of secondary functions: The creation of multiple charging stations, as well as the creation and integration of a charger s_i or f_i, discharger d_i, battery swapper eh_i, parking slot prk_i or inductive charger ind_i in every station. Additional operations are the recharging of batteries b_i which are later to be swapped into EVs, as well as the ability to add new batteries to the storage for the seamless operation of the battery exchange process to be achieved. Finally, the cost of the charging $p_{i,t,s}$ or $p_{i,t,f}$, discharging $p_{i,t,d}$ and battery swapping $p_{i,t,eh}$ can be calculated based on various cost types (e.g. energy cost) defined by the user.

During the creation of the charging station i, 4 waiting lists are created. A list for the charging events which want slow charging $levnt_{i,s}$, a list for the charging events which want fast charging $levnt_{i,f}$, a list for the discharging events $levnt_{i,d}$, and a list for the vehicles waiting for battery exchange $levnt_{i,eh}$. A list can be managed in 2 ways: Either automatically by the library using the default settings, or programmatically by the user. Moreover,

Algorithm 1 Charging event handling algorithm

```
Require: $e_j^{cap}$, $e_{j,t}^{rem}$, $e_{j,t}^{dem}$, $t_j^{max}$, $e_{i,t}^{sol}$, $e_{i,t}^{wind}$, $e_{i,t}^{wave}$, $e_{i,t}^{geo}$, $e_{i,t}^{hydro}$, $e_{i,t}^{nonR}$, $avlb_{s_i,t}$.
1: Create charging event $evnt_{i,j,s}$ and set event state $evnt_{i,j,d}^{state}$ to *arrived*
2: if $(|S_i| = \emptyset)$ then {No slow chargers exist in the charging station }
3:     Set event $evnt_{i,j,s}$ state $evnt_{i,j,d}^{state}$ to *rejected*
4: {Calculate charging time}
5: $t_{i,j}^{chrg} = e_{j,t}^{dem}/s_i^{rate}$
6: {If the charging time is larger than the maximum time the EV can be at the station, the event is rejected }
7: if $(t_{cur} + t_{i,j}^{chrg} > t_{cur} + t_j^{max})$ then
8:     Set event $evnt_{i,j,s}$ state $evnt_{i,j,d}^{state}$ to *rejected*
9: {Search available charger in $avlb_{s_i,t}$ for the duration $t_{i,j}^{chrg}$ of the event}
10: Calculate $sum_{chrg} = \sum_{t:t \geq t_{cur}, t \leq t_{cur}+t_{i,j}^{chrg}}(avlb_{s_i,t})$
11: {Search available energy in all sources based on demand $e_{j,t}^{dem}$ for $t_{i,j}^{chrg}$}
12: Calculate $sum_{enrg} = \sum_{t:t \geq t_{cur}, t \leq t_{cur}+t_{i,j}^{chrg}}(e_{i,t}^{sol})$. Repeat for all sources
13: {If charger and energy exist, start immediate charging}
14: if $(sum_{chrg} \geq t_i^{chrg}$ AND $sum_{enrg} \geq e_{j,t}^{dem})$ then
15: {Calculate price to pay}
16:     $p_j = p_{i,t,s} \times e_{j,t}^{dem}$
17:     Set event $evnt_{i,j,s}$ state $evnt_{i,j,d}^{state}$ to *accepted*
18: {Schedule for later charging}
19: else
20: {Search charger in $avlb_{s_i,t}$ for the duration $t_{i,j}^{chrg}$ for $t_{cur} + t_j^{max}$ }
21:     Calculate $sum_{chrg} = \sum_{t:t \geq t_{cur}, t \leq t_{cur}+t_j^{max}}(avlb_{s_i,t})$
22:     Add the time points the charger is available to set $\tau_{chrgavlb}$
23: {If available charger exists}
24:     if $(sum_{chrg} \geq t_i^{chrg})$ then
25: {Search energy for the time the charger is available}
26:         Calculate $sum_{enrg} = \sum_{t \in t_{chrgavlb}}(e_{i,t}^{sol})$. Repeat for all sources
27: {If available energy exists}
28:         if $(sum_{enrg} \geq e_{j,t}^{dem})$ then
29:             Set event $evnt_{i,j,s}$ state $evnt_{i,j,d}^{state}$ to *accepted*
30:             Add event to the waiting list $levnt_{i,s}$
31:         else
32:             Set event $evnt_{i,j,s}$ state $evnt_{i,j,d}^{state}$ to *rejected*
33:     else
34:         Set event $evnt_{i,j,s}$ state $evnt_{i,j,d}^{state}$ to *rejected*
35: Return $evnt_{i,j,d}^{state}$
```

each time an event is added to any list, an expected maximum waiting time t^{wait}_{levnt} is calculated, and once an event is executed these times are updated.

Another useful feature is the plethora of energy sources for the supply of energy to the charging station, which are available through the use of an energy warehouse. The library includes 6 different relevant classes, where each class describes a different energy source (i.e., $e^{sol}_{i,t}$, $e^{wind}_{i,t}$, $e^{wave}_{i,t}$, $e^{geo}_{i,t}$, $e^{hydro}_{i,t}$, $e^{nonR}_{i,t}$). The amount of available energy in the warehouse is updated in a regular basis. For this update there are 2 options: In the first one, the user is responsible for calling the energy storage update method. At each invocation only one energy unit is withdrawn from the total energy amount of each energy source, inserting it to the station's energy warehouse. In the second option, the user must initially enable the automatic update of the energy storage and then define a time space. Then the energy update storage method is executed automatically at specified time moments.

The waiting lists give the ability to the charging station to execute some events, following a priority order based on the time of arrival at the station. An event is inserted to the waiting list during the pre-processing phase. As described earlier, the mechanism is the following: The method looks for an empty charger/discharger/exchange handler. In case of not finding one, the method calculates the time the event should wait. The next step is the comparison of this time with the maximum time t^{max}_{i} the event can actually wait. If the calculated waiting time is less than the time the vehicle can wait, the event enters the waiting list and the status is set to "wait". In any other case, the event's status is set to "nonExecutable", meaning that the execution is not possible. A method returning the waiting time t^{wait}_{evnt} of each event is provided. Additionally, for each kind of event a method for the remaining time t^{rem}_{evnt} is provided to constantly inform the users of the state of each event.

Pricing policy for the charging function is a typical Smart Grid feature. Two different options are provided. One with fixed time space for each price and one with varying time space. By time space we actually refer to the time period that a specific price is valid. A method for the definition of the current prices for slow $p_{i,t,s}$ or fast $p_{i,t,f}$ charging, discharging $p_{i,t,d}$, battery swapping $p_{i,t,eh}$ and inductive charging $p_{i,t,ind}$ is included.

ChargingEvent, DisChargingEvent and ParkingEvent classes store all the created events. Based on this, the library is equipped with a method which returns all this information along with some descriptive statistics regarding the operation of the charging station all gathered in a text file.

Synchronizing and scheduling a group of charging events is another feature of EVLib. The user can give, through a text file, a plan of events for execution committing a predefined number of chargers. Each charging is carried out in parts or in whole. There is a detailed description of this feature in Section 4.

In the following subsections the main classes of the EVLib library are presented.

3.2.3. ChargingStation

ChargingStation class describes a charging station and is a central class in EVLib. Technically speaking, this class is the corner stone of the library since everything is attached to it, or is strongly connected with it as shown in Figure 1. The class has several methods, the most important of them being:

1. **updateQueue(ChargingEvent event)** inserts a charging event to the corresponding waiting list.
2. **getChargers()** returns an array with all the chargers of the charging station.
3. **getWaitingTime(String kind)** returns the waiting time for a specific type of event. The values of the argument are: "fast" for fast charging, "slow" for slow charging, "exchange" for the battery exchange operation, "discharging" for the discharging function.
4. **assignCharger(ChargingEvent event)** assigns a charger to the given event.
5. **insertEnergySource(EnergySource source)** inserts one of the already predefined energy sources to the charging station.

3.2.4. Charger

This class represents a charger of a charging station. Charger class' main methods are:

1. **startCharger()** starts the execution of the linked charging event.
2. **handleQueueEvents()** executes the first charging event in the waiting list.

Note that the DisCharger, ExchangeHandler and ParkingSlot classes operate similarly to the Charger class.

3.2.5. ChargingEvent

ChargingEvent class is responsible for the representation of a charging event. The main methods of this class are:

1. **preProcessing()** method which executes the preliminary process before the charging.
2. **execution()** is responsible for starting the execution of the event.

3.2.6. EnergySource

The library contains a set of renewable and non-renewable energy sources that can provide energy to the charging station. Specifically, there are 5 sub-classes each of them representing a renewable source and 1 sub-class for the energy from non-renewable sources. All sub-classes have the same interface and inherit this abstract class. Each of the 6 classes that inherit EnergySource class contain an *energyAmount* field, which stores the energy that is going to be inserted after each update.

Moreover, EnergySource class has 2 methods:

1. **popAmount()** returns the first energy amount to be given in the charging station.
2. **insertAmount(double amount)** inserts an energy amount to the energy source list of amounts.

PricingPolicy Pricing policy class provides two different types. Primarily, a pricing policy with a fixed time space for each price and the second one with changing time spaces.

Some of the most noticeable methods are:

1. **setTimeSpace(long timeSpace)** sets the time space when it is fixed.
2. **getDurationOfPolicy()** returns the duration of the pricing policy.
3. **setSpecificPrice(int index, double price)** modifies a price.
4. **getSpecificPrice(int index)** returns the price in the "index" position, or 0 if "index" is greater than the number of the policy's prices.

3.2.7. WaitingList

WaitingList class is responsible for keeping the events that have not yet been assigned to any charger, discharger or exchange handler during the preprocessing phase and ensuring the right order of them. Technically speaking, waiting list operates similar to the FIFO Queue data structure. Events are ordered based on their arrival time. It contains a variety of methods, providing in this way the tools for the handling of a set of events.

WaitingList class includes the following functions:

1. **add(T object)** inserts an object in the list.
2. **get(int index)** returns the object in the "index" position.
3. **getSize()** returns the number of objects in the list.
4. **moveFirst()** removes the first object and returns it back.
5. **takeFirst()** returns the first object of the waiting list.

In the next section, the EVLibSim simulation environment, which makes use of all EVLib's classes and methods, is presented.

4. EVLibSim

EVLibSim[2] simulation tool, aims to unify EVLib's functions and present them in a simple, yet efficient and comprehensive user interface. EVLibSim provides the ability to configure a charging station depending on the user's demands, along with the configuration of its environment taking advantage of all EVLib's features. The user can initialize a charging, dis-charging, battery swapping or parking event at any time. The simulation tool operates constantly waiting for a new event to be scheduled for execution or an action to be processed. Each time a new EV arrives, Algorithm 1 (or equivalent algorithms for the other event types) is called. Given that the complexity of this algorithm is $O(1)$, the complexity for the simulation tool is $O(n)$ where n is the number of EVs that have already arrived to the station. In the meantime, all the proper tools for the management of energy-related activities are provided. Another aspect of the tool is that the user can keep logs of all executed events utilizing the information for further processing and data analysis. Finally, EVLibSim uses real-time plots for monitoring all the running processes in the station.

4.1. Main View

Through the start screen of the tool the user can have access to all the basic functionality (see also Figure 2). The creation of a new charging station, addition of a new pricing policy, construction of any kind of event and management of energy activities are the main features in the central panel. In

[2]http://intelligence.csd.auth.gr/files/systems/EVLib/evlibsim.jar

Figures 3, 4, 5 we can see the provided choices of the 3 buttons of start screen. The user can also navigate using the menu bar, where all the operations are also included.

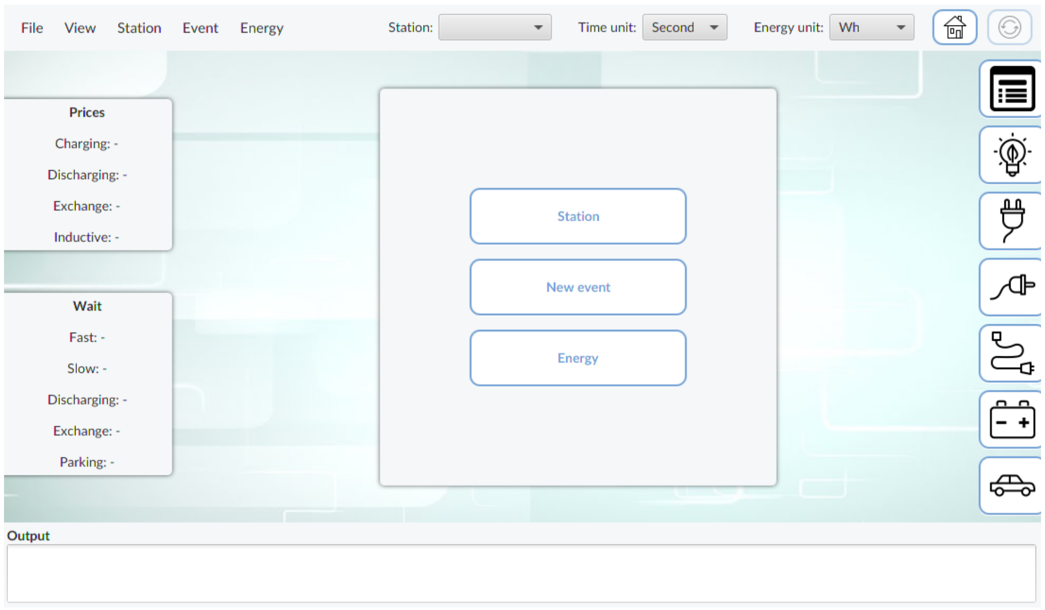
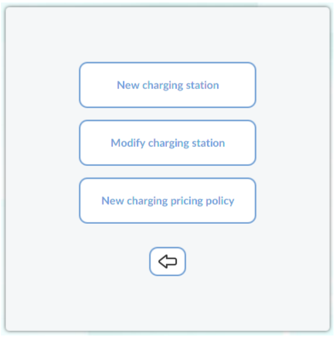
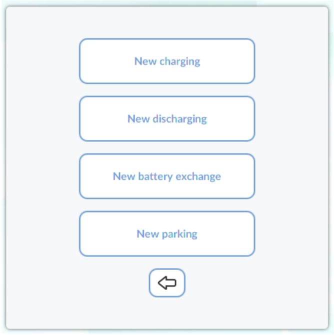


Figure 2: Start screen of EVLibSim





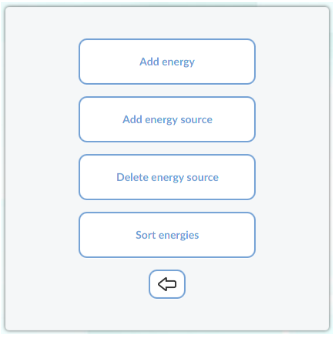
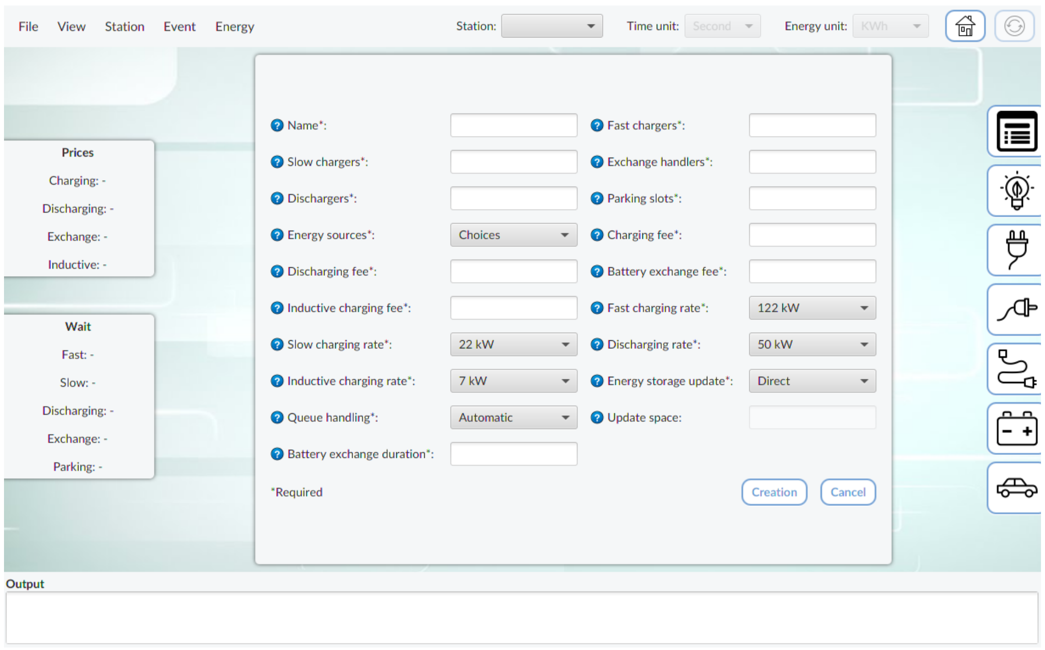


Figure 3: Options of "Station“ button

Figure 4: Options of "New Event“ button

Figure 5: Options of "Energy“ button

The “Station” selection provides 3 options: Creating a new charging station, attaching a new pricing policy to a running station and modifying the running station. The “New charging station” button leads to a new window with the fields shown in Figure 6. The top fields concern the number of chargers, dischargers, battery handlers, parking slots and the types of energy sources linked to the station. The middle and lower ones refer to the basic configuration of the charging station, such as the rates of the various functions (charging rate, discharging rate, inductive charging rate), the prices of energy units for each basic operation, the handling of the waiting lists, the mode of energy storage update and the duration of the battery swapping function. The modification of a station is similar to the creation but the fields related to the infrastructure of the station are deactivated. The user cannot change the number of chargers, dischargers, battery swapping handlers or parking slots because some of them may be occupied. The insertion and deletion of them can be performed through separate menu functions.



Prices
Charging: -
Discharging: -
Exchange: -
Inductive: -

Wait
Fast: -
Slow: -
Discharging: -
Exchange: -
Parking: -

Name*:		Fast chargers*:	
Slow chargers*:		Exchange handlers*:	
Dischargers*:		Parking slots*:	
Energy sources*:	Choices	Charging fee*:	
Discharging fee*:		Battery exchange fee*:	
Inductive charging fee*:		Fast charging rate*:	122 kW
Slow charging rate*:	22 kW	Discharging rate*:	50 kW
Inductive charging rate*:	7 kW	Energy storage update*:	Direct
Queue handling*:	Automatic	Update space:	
Battery exchange duration*:			

*Required

Figure 6: Creation of a new charging station

The “New Event” option leads to a new window for the creation of a new event. The window has 4 basic fields: 1) name of the driver, 2) brand of the vehicle, 3) capacity of the battery and 4) remaining amount of energy. A user can select the vehicle’s brand through a choice box providing only electric vehicles as choices. The capacity always needs to be greater than the remaining amount. Next, depending on the kind of event, there are fields for the proper construction of the event. The user should always be aware of the remaining energy in the station for charging and available batteries for battery swapping. Otherwise, the events will not be executed.

The selection of the “Energy” button opens a new window to manage the activities related to the sources and the available energy in the charging station (see also Figure 7). Each field corresponds to a specific kind of available energy source. The energy source types that are available for a specific station are marked with an asterisk. An amount of energy can be inserted only to those sources that were selected during the creation of the station. When adding an amount, it is recorded to the storage of each source, but not yet to the station’s warehouse. The energy storage update method is responsible for this. The energy storage update is made either directly or it can be scheduled for a later time. Selecting “Direct”, the energy amounts are added immediately to the station’s warehouse. Selecting “Scheduled”, the transfer will be implemented in one of the next scheduled updates. At each update only one energy unit from each source is transferred to the station. Moreover, the option to add or remove an energy source from the charging station is provided.

The sorting of the energy sources is another option included in the “Energy” selection. It provides the ability to select which energy source will be used first to provide energy during a charging. By selecting this option, a window, similar to the one for energy addition, opens. The user needs to define a number for each field that signifies the order. The range of the values is 1 to the number of attached energy sources.

Beyond the start screen’s operations, on the left part of the screen two panels exist (see Figure 2). The “Prices” panel informs the users regarding the current prices per energy unit for the charging, discharging, the inductive charging and the price of battery swapping operation. The “Wait” panel presents the waiting time for each operation based on the state of the equivalent lists.

Moreover, there is an output box located at the bottom of the screen. It shows messages about the completion of an event which typically consists of the event type and ID, followed by the name of the driver, the brand of the vehicle, the station and the “OK” keyword. A plan execution provides some extra information, such as a message for the completion of a specific charger’s plan. Furthermore, the end of all the chargers’ plans signifies the completion of the plan, leading to a message display. Notice that EVLib library does not allow the simultaneous application of more than one “Plan

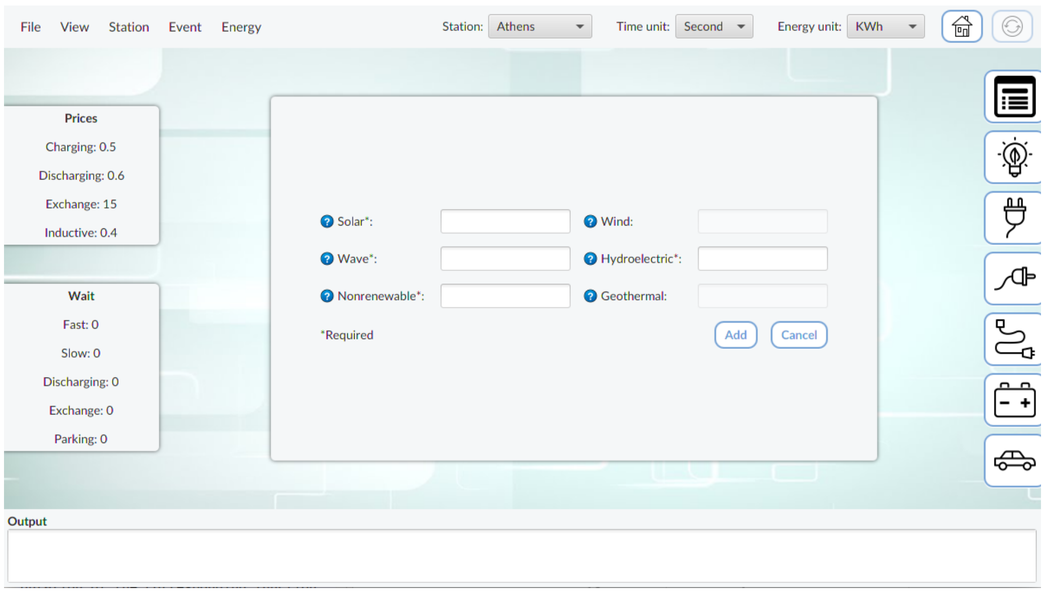


Figure 7: Addition of energy

execution" functions.

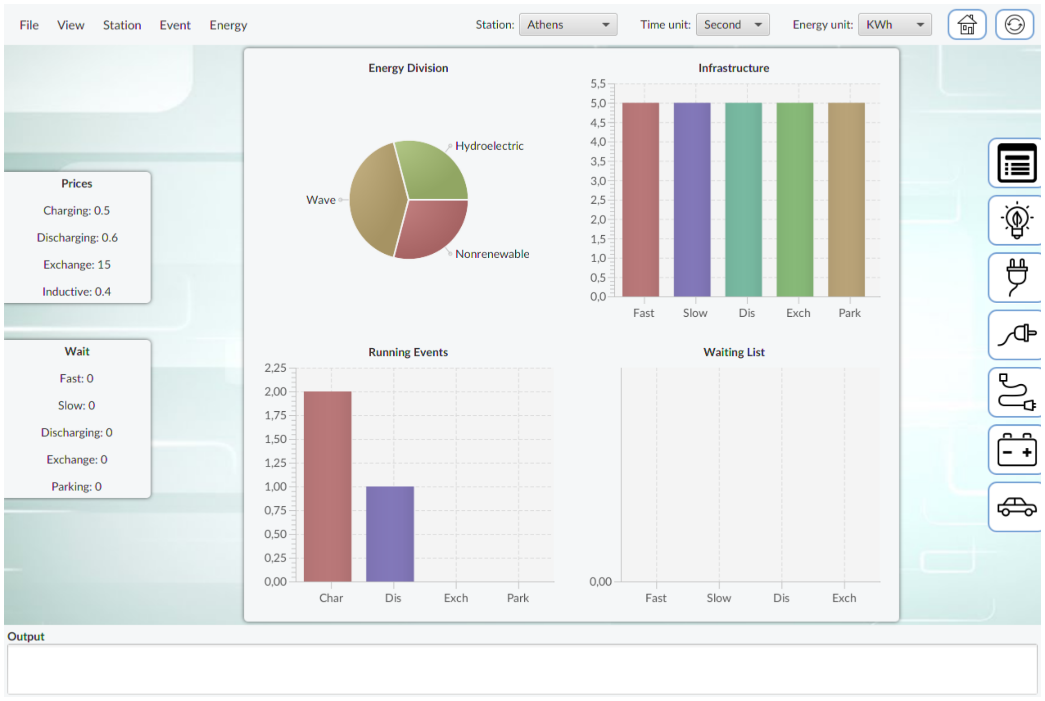
4.2. Menu Bar Functionality

The menu bar provides the complete functionality of the simulation tool. Beyond the categories with the features, the tool has several options to control the simulation environment, through the use of selection boxes as shown in Figure 2 in the upper right corner. The first one is for controlling the running charging station. It contains all the created stations, giving the ability to select one of them. By saying *running* we mean that every action in the EVLibSim concerns that station. The next selection box refers to the time metering unit. The user can choose between two units, the second and the minute. Finally, the last one is for the energy unit. The available energy units are the Wh and the kWh. The metering unit can be updated at any time and the values in all related fields and operations are updated automatically. Additional functions provided from the menu bar are described in what follows.

The user can save all the previously created stations with their amounts of energy and attached batteries, as well as all the events' history with their progress and load all these whenever he/she wants using "Save" and "Save as..." operations. Additionally, he/she can start over the operation of the tool using the “New” feature, at any time. The “Events report” operation is responsible for exporting all the created events since the beginning of a session. “Station report” gives an analytical report of a selected charging station, which contains information for the state of the charging station. Both types of reports are text files.

The user can also enable or disable the suggestions box showing up in every event’s creation window from the “Suggestions” selection. Suggestions box refers to some information being given by the tool, trying to consult the users to select the best charging station. The recommendations are categorized based on energy, prices and waiting times. The main idea is to sort the energy, prices, and waiting times of each station. Recommendations include only the first five stations.

The supervision of the different actions taking place in the station is provided through an overview window with 4 plots as shown in Figure 8. The upper plots show the energy partition and the infrastructure of the station respectively, while the lower charts show the number of the running events and the state of the waiting lists respectively.



Station: Athens
Time unit: Second
Energy unit: KWh

Prices
Charging: 0.5
Discharging: 0.6
Exchange: 15
Inductive: 0.4

Wait
Fast: 0
Slow: 0
Discharging: 0
Exchange: 0
Parking: 0

Figure 8: Overview plots of a charging station

The ability to insert or delete a new charger, discharger, battery handler, a new parking slot or a new battery for the efficient operation of the battery swapping function is also provided through “Station” menu. For the chargers, only the supported type of charging needs to be defined. For a battery creation, three fields need to be completed, namely the battery capacity, the remaining energy in the battery and the upper number of battery’s chargings. The battery swapping function demands the charging of the attached to the station batteries.

Pricing policy is a solution for balancing the excessive demand and workload across a set of charging stations [29]. By selecting this function a pop-up window is shown, giving two choices. The “Fixed space” selection denotes that each price will be valid for a fixed time space. In contrast, the “Changing space” choice means that every price’s time space will be different. In both cases, after the user’s initial choice, a window with two fields is displayed. For the fixed time space option, the user must define the time space and then a set of prices. For the changing time space option, the user should set the time spaces in the first field and the prices in the second one. There is no restriction regarding the number of prices the user sets. After the end of pricing policy’s duration, the charging price returns to the default value. In Section 6.1 there is a detailed example of the implementation of this operation.

The “Plan execution” selection is for the simulation of a set of events the scheduling plan of which has been calculated externally of the EVLibSim. Events use a predefined set of chargers, that will be shown as occupied until the end of execution of this set of events.

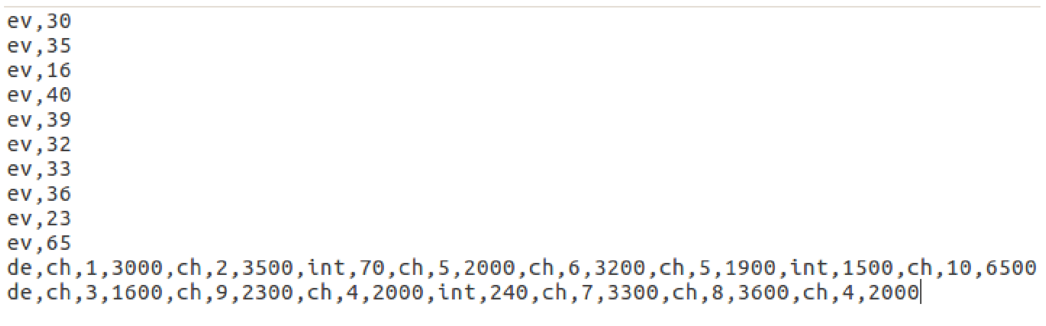
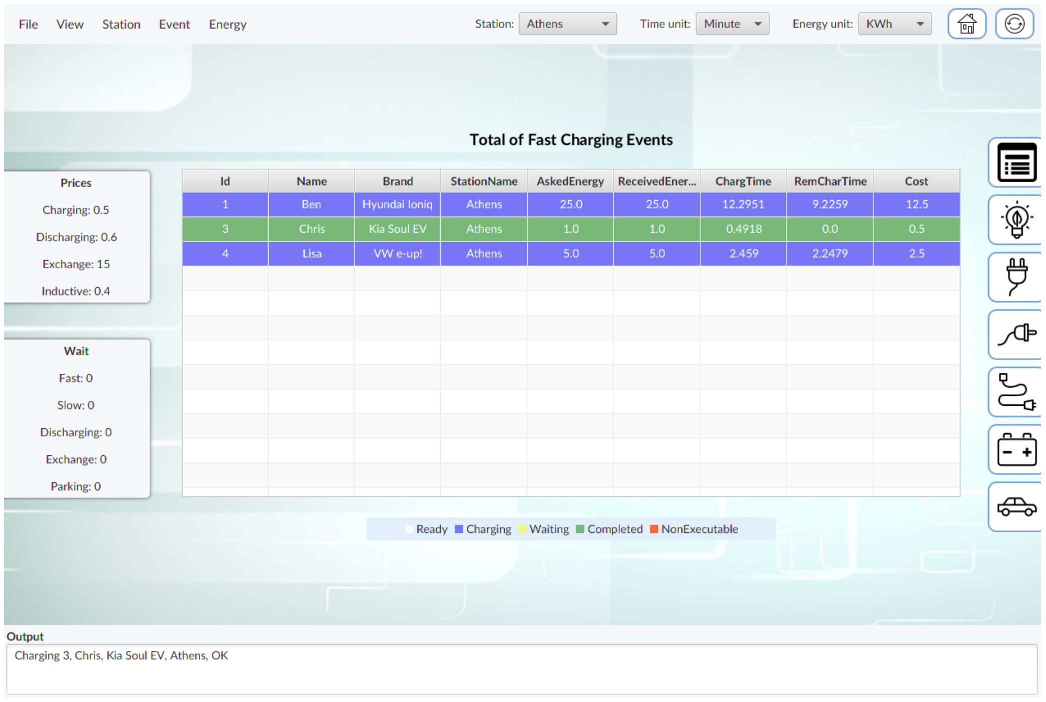

```
ev,30
ev,35
ev,16
ev,40
ev,39
ev,32
ev,33
ev,36
ev,23
ev,65
de,ch,1,3000,ch,2,3500,int,70,ch,5,2000,ch,6,3200,ch,5,1900,int,1500,ch,10,6500
de,ch,3,1600,ch,9,2300,ch,4,2000,int,240,ch,7,3300,ch,8,3600,ch,4,2000
```

Figure 9: Plan execution template - example

Plan execution is defined through a text file as the one displayed in Figure 9. At each line, symbols or numbers are separated with comma. Initially, we define the charging events and then the chargers' plans. Every charging is executed using only one charger. The symbol "ch" denotes a charging, followed by the energy to be received. At the end of the charging events, the plan of every charger follows. Symbol "de" signifies the start of a charger plan. Inside the line of a charger plan there are also some other symbols. Symbol "ch" means that a part of a charging starts. The number shows the exact charging from the initial list with the charging events. Symbol "int" implies an interruption and its duration follows after the comma. In our case, the plans that tested by our simulation tool were calculated from an already existing scheduling algorithm [31].



Station: Athens Time unit: Minute Energy unit: KWh

Total of Fast Charging Events

Prices
Charging: 0.5
Discharging: 0.6
Exchange: 15
Inductive: 0.4

Wait
Fast: 0
Slow: 0
Discharging: 0
Exchange: 0
Parking: 0

Id	Name	Brand	StationName	AskedEnergy	ReceivedEner...	ChargTime	RemCharTime	Cost
1	Ben	Hyundai Ioniq	Athens	25.0	25.0	12.2951	9.2259	12.5
3	Chris	Kia Soul EV	Athens	1.0	1.0	0.4918	0.0	0.5
4	Lisa	VW e-up!	Athens	5.0	5.0	2.459	2.2479	2.5

⬜ Ready 🟦 Charging 🟨 Waiting 🟩 Completed 🟥 NonExecutable

Output
Charging 3, Chris, Kia Soul EV, Athens, OK

Figure 10: Summary of chargings

4.3. Toolbox

At the right hand side of EVLibSim a toolbar is located and one of its choices is the ability to provide information about the energy in the station. By selecting it, a window is shown containing all the connected, to the charging station, energy sources and their remaining energy units. Moreover, the toolbar includes 5 event selections. Each of them corresponds to a different type of event, namely fast charging events, slow charging events, discharging, battery swaps, and parking events. Each selection displays the state of all events of the same type in a table-like form (see Figure 10), where every line represents a different event. The event's displayed information is the received energy, the waiting time of the vehicle, the cost, the station they were serviced, the time of their execution and the remaining time for the completion.

5. User Evaluation and Testing

In this section we present the evaluation of the EVLib library and EVLibSim simulation environment.

5.1. Code security and safety testing

EVLib and EVLibSim were checked for errors regarding the code development, errors about the view of the code, or security holes that may exist using state of the art tools. In so doing, FindBugs[3] and CheckStyle[4] were utilized. FinbBugs is a tool which carries out static testing on Java code. Alongside, CheckStyle reports all cases which do not follow patterns of proper programming, related to the view and logical development of code. In addition to these programs, a number of unit tests, related to all classes were developed using JUnit 5. All the errors that were discovered were corrected.

5.2. Evaluation of the EVLibSim

The evaluation phase of the EVLibSim included a questionnaire[5] with 8 questions of varying difficulty that was assigned to 21 persons from the IT sector. Each question required a task to be executed using one of EVLibSim's features. The user had to rate from 1 (very difficult)-to-5 (very easy) the difficulty of the task. Comments about the corresponding task could be freely added by the users. In the end, the user could also add general comments about the tool as well as suggestions for changes and / or new functionality.

The questionnaire had two parts. The first part included 4 questions about the background of every respondent. The questions referred to the

[3]http://findbugs.sourceforge.net/

[4]http://checkstyle.sourceforge.net/

[5]https://goo.gl/forms/9GYAeK0hK5sI0qZU2

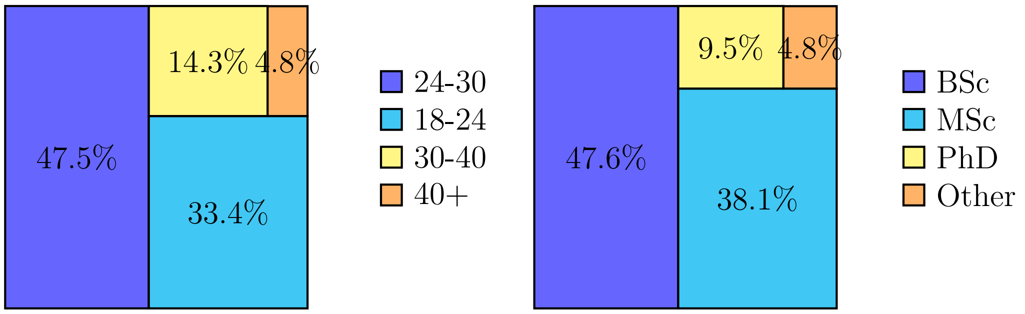


Figure 11: Age and education of participants

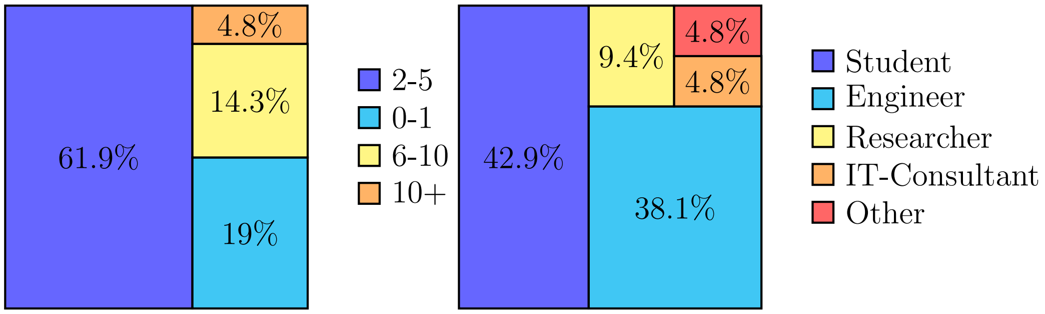


Figure 12: Years of experience and profession of participants

age, education, profession and experience of the participants. The answers are presented in Figures 11 and 12.

Task	**Group 1**	**Group 2**	**Group 3**	**Overall**
Create a new charging station.	4	3.5	3.5	3.7
Add an amount of energy to an existing energy source.	3.8	4.2	3.7	3.9
Update/modify an existing charging station.	4	3.7	3.4	3.7
Create an (charging) event.	3.8	3.8	3.7	3.8
Using the overview buttons on the right-hand side of the tool, evaluate the available information and how useful they are in understanding the current state of the station.	4.2	4	4	4
Evaluate your understanding regarding the waiting queue of events and how it relates to the waiting time panel.	3.4	4	3.2	4
Construct a new pricing policy using both options. Evaluate your understanding of the difference between fixed time space and changing time space.	2.7	2.8	3	2.8
Export the information related to all events to a csv (comma separated value) file, using the button inside the File menu.	4.5	4.5	4	4.4

Table 2: Questionnaire's tasks with the average answers.

The second part included the tasks/questions about the simulation tool. Table 2 presents the questionnaire's tasks and the average grade for each task. In conducting this evaluation, we followed the Delphi method [26] and we separated our set of participants to 3 groups. After the end of each group's evaluation, the tool was improved based on their comments and the updated version was provided to the next group.

The first group completed the questionnaire having the initial version. This version did not show tips, the rates of each (charging or discharging) function at the charging station creation panel had to be manually defined by the user, events' history export was not available and the user had to define the time he/she could wait until serviced. Among the most crucial suggestions from the first set of participants were the addition of color code in events' history panels, tips for better understanding of the terms and time not to be measured in milliseconds but in a more comprehensible unit i.e., seconds or minutes. The proposed features were taken into account by making the corresponding improvements to the tool. The field for waiting time at every event's creation panel was removed, a choice box for time measuring units having as options seconds or minutes was added in the menu and tips were added for each field in the tool that needs information to be entered by the user. The rest of the requested features were also implemented.

The second group of participants evaluated the improved version of EVLibSim. The users of this phase outlined two big problems at the modification of the charging station and at the sorting of energy sources. The suggested improvements were the addition of the feature to restart the application, the addition of a home button and the addition of a choice box for energy measurement units. Furthermore, users had an inconvenience to comprehend the construction of a pricing policy. All the user suggestions were implemented along with the addition of an explanatory text for the pricing policy function.

The third set of respondents evaluated the most improved edition of EVLibSim. Most of the comments had to do with the pricing policy feature, the weakness to understand the terms at the charging station creation panel (i.e., charging rate) and the range in which the required values vary. Some other participants proposed the addition of an example of how to construct a pricing policy, the vehicle brand field to be substituted with a choice box containing only electric vehicles brands, more informative error messages and a better positioning in the screen for the selection of changing the running charging station.

To sum up, the most important reported issues were the following:

1. Better explanation of the values required in the charging station creation panel.
2. Provision of a more explanatory example to facilitate the pricing policy feature.
3. Update of all fields in the modification of a charging station panel.

4. Difficulty in understanding waiting time field in association with left panel that shows waiting times.
5. Exporting of events' log.
6. Difficulty to comprehend the energy storage update options.
7. Provision of a manual/video with examples.
8. Insufficient display time of tips.
9. Better explanatory messages in case of fields completion errors.
10. Moving the choice box for charging station in an easier to read position in the screen.

In the latest revision of the tool, we positioned the choice box for selecting charging station in the menu bar and we provided an example for constructing a pricing policy at the top of each creation panel. In addition, we altered the brand field with a choice box giving as options only electric vehicles brands. At the charging station creation panel, the rate fields (charging rate, discharging rate, inductive charging rate) were replaced by choice boxes with values that are valid in real stations.

In table 2 the average values of users' answers are presented. Users' ratings did not increase among the 3 groups as long as new features and corrections were inserted. The lowest average rating was for the creation of a pricing policy followed by the creation of a new charging station and its modification. The main reason for this is the requirement to comprehend and provide values for a lot of new terms. In contrast, exporting reports has the highest average rating.

The user evaluation results are considered satisfactory. The participants were interested in testing an innovative application and to learn new terms concerning electric vehicles. Positive comments include the wide range of functions, appropriate presentation and the ability to export reports on the operation of EVLibSim. In contrast, most users complained about the lack of documentation for some novel terms (i.e., charging rate or remaining energy in battery) and had difficulties to construct a pricing policy. Through their comments we had the opportunity to enhance EVLibSim. We believe that the version that came out of the suggestions is more compact and user-friendly.

6. Using EVLibSim

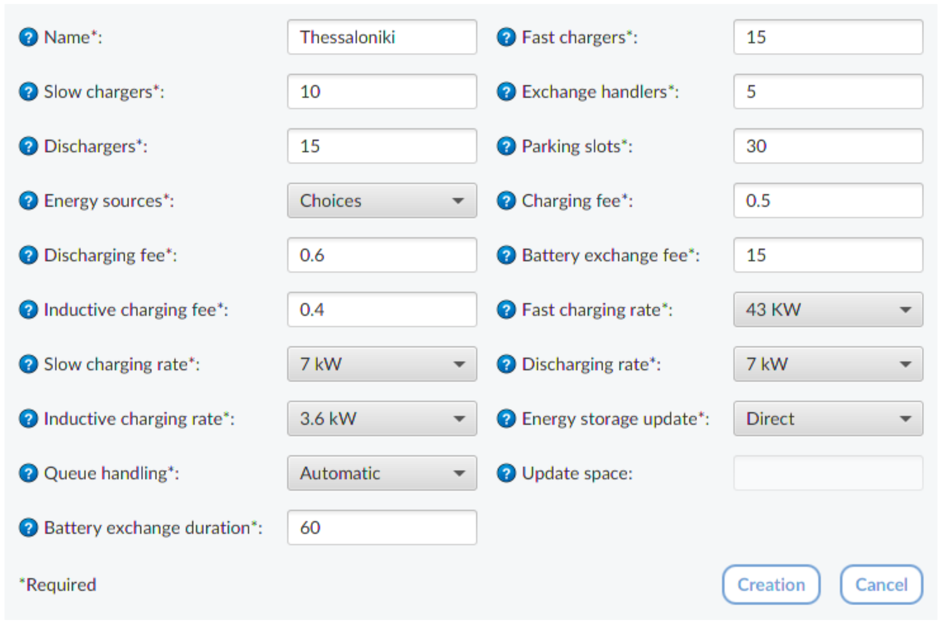
In this section, examples related to the use and performance of EVLibSim are provided.

6.1. Example of Use

The objective of this section is the presentation of EVLibSim's basic functions, in order to provide guidelines for the proper use of the simulation tool and the required execution order of some activities.

6.1.1. Charging Station

The integration of a charging station is the most significant function in EVLibSim. To create a new charging station, the user should select "Station" → "New charging station" from the main panel or the menu bar. In Figure 13 an example configuration of a charging station is presented. The fees refer to the cost of the Wh or kWh for each operation, except for the battery swapping function where the cost is for the whole process. The rates (charging, discharging and inductive) define the energy that is transferred per hour, and determine the duration of the corresponding function. The linked energy sources are wind, wave and solar.



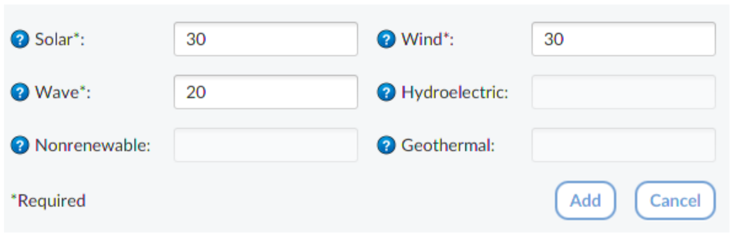
Name*:	Thessaloniki	Fast chargers*:	15
Slow chargers*:	10	Exchange handlers*:	5
Dischargers*:	15	Parking slots*:	30
Energy sources*:	Choices	Charging fee*:	0.5
Discharging fee*:	0.6	Battery exchange fee*:	15
Inductive charging fee*:	0.4	Fast charging rate*:	43 KW
Slow charging rate*:	7 kW	Discharging rate*:	7 kW
Inductive charging rate*:	3.6 kW	Energy storage update*:	Direct
Queue handling*:	Automatic	Update space:	
Battery exchange duration*:	60		

Figure 13: Charging station example

6.1.2. Energy Addition

The addition of energy is the second required action. Energy is necessary for charging operations. At each update, an amount of energy is selected from each associated source with the charging station.

To add energy, we select “Energy” → “Add energy” from either the main screen or the menu bar. Figure 14 presents an example of this selection, where 30 kWh of Solar energy, 30 kWh of Wind and 20 kWh of Wave energy become available.



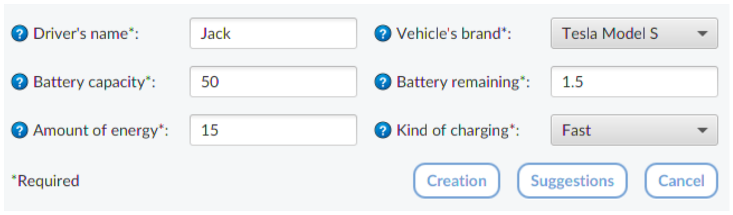
Solar*:	30	Wind*:	30
Wave*:	20	Hydroelectric:	
Nonrenewable:		Geothermal:	

*Required

Figure 14: Energy addition example

6.1.3. *Charging Event*

Charging is implemented either by selecting “New event” → “New charging” from the main screen or from the menu bar. Figure 15 shows the given values for a charging event. The vehicle’s brand is Tesla Model S, the battery capacity is 50 kWh, the remaining energy in the battery is 1.5 kWh, the requested amount of energy is 15 kWh to be charged using fast charging.



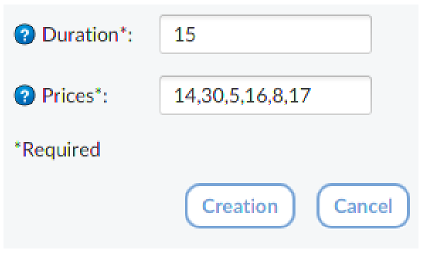
Driver's name*:	Jack	Vehicle's brand*:	Tesla Model S
Battery capacity*:	50	Battery remaining*:	1.5
Amount of energy*:	15	Kind of charging*:	Fast

*Required

Figure 15: Charging event example

6.1.4. *Pricing Policy*

The integration of a pricing policy function is there to regulate the congestion of a charging station. To use this feature, we select “Station” → “New charging pricing policy” from either the main screen or the menu bar. In Figure 16, such a policy with 6 values, each of them lasting for 15 minutes, is presented.



Duration*:	15
Prices*:	14,30,5,16,8,17

*Required

Figure 16: Pricing policy example

6.2. Usage scenarios

This section presents a comparison between a number of common usage scenarios, in order to better comprehend EVLibSim's functionality. The scenarios focus on the way the charging station handles a number of charging events when there is not enough energy or available chargers to complete all events. The state of waiting queues, energy sources and running charging events are presented through graphs.

The configuration of the charging station, as well as the amount of available energy is the same for all scenarios. The charging station's configuration is the one shown in Figure 13. Regarding the energy, we assume that the station has stored 50 kWh of solar energy, 50 kWh of wind energy and 20 kWh of wave energy.

Scenario	**Slow Events**	**Fast Events**	**Requested Energy**	**Energy Sources Order**
1	2	2	56	Wave, Solar, Wind
2	5	5	96	Solar, Wind, Wave
3	5	5	215	Wind, Solar, Wave

Table 3: Scenarios of EVLibSim

Table 3 shows the first 3 examined scenarios. According to scenario 1, the first slow charging requests 15 kWh and the second 17 kWh. The first fast charging event requests 14 kWh and the second one 10 kWh. Scenario 2 has 5 slow charging events and 7 fast charging events. For all charging events the requested energy is 8 kWh. In scenario 3 the order and the requested energy measured in kWh are the following: slow(20), fast(20), slow(20), fast(20), slow(30), fast(20), slow(15), slow(20), fast(25), fast(25). Figures 17 and 18 demonstrate the number of charging events in the waiting lists and the remaining energy amounts of each energy source, after the insertion of all charging events at each occasion.

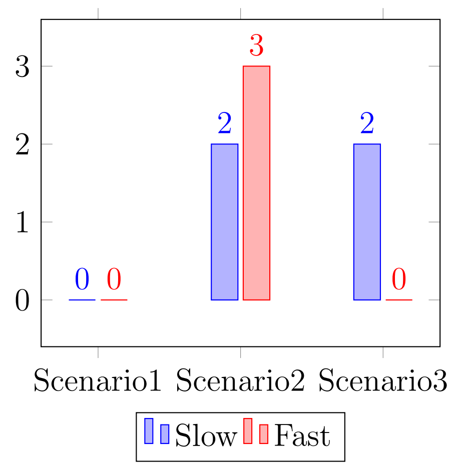


Figure 17: Waiting queues state

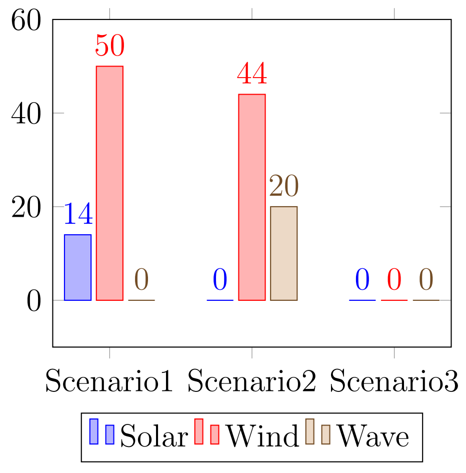


Figure 18: Energy sources state

The first result to be outlined is that the required energy is committed as soon as the event starts to be executed. In scenario 2, the total requested energy from all events is 96 kWh. The total charging station's energy is 120 kWh, meaning that there is enough energy for all charging events. Considering that every charging event demands 8 kWh and all running chargers are 7 that means 56 kWh of energy, which is the result of the total energy minus the sum of remaining wind and wave energy amounts shown in Figure 18. Therefore, the charging station provided energy only to those events that charge at this moment. The remaining 40 kWh that is the total required energy amount of waiting lists' events will be committed when events' execution starts.

It is interesting to note that the number of events in the waiting lists is always 0, or less or equal to the total number of events minus the total number of chargers in the station. In scenario 1, the number of slow and fast chargers are greater than the number of slow and fast charging events respectively. Thus, 0 events are in the waiting lists as shown in Figure 17. In scenario 2, the number of charging events is 12, while the number of chargers are 7, meaning that the events in the waiting list are 5. Figure 17 outlines this difference.

Scenario 3 corresponds to the third occasion where the number of waiting charging events is smaller than the total number of events minus the total number of chargers. Observing the created events, it is clear that the charging station's energy has been totally consumed after the insertion of the third fast charging event. Until here, all the slow chargers are occupied, but one fast charger is available. The next 2 slow charging events enter the waiting list, while the next 2 fast chargings do not. Although there is an available fast charger, the events are rejected and declared as non executable. The reason is the unavailability of energy to charge.

Scenario	4	5	6	7	8	9
Slow chargings	8	17	32	30	25	50
Fast chargings	12	13	15	25	55	50
Non executable chargings	14	27	35	49	60	85
Energy(kWh) demand per charging	20	40	10	20	6	8

Table 4: Scenarios where there is not enough provided energy

Scenario 3 revealed that the remaining energy is a crucial factor that defines charging events' execution. In order the escalation of this phenomenon to be examined, 6 extra scenarios were created, while the number of chargers (3 slow chargers and 4 fast chargers) and remaining energy (120 kW) in the charging station are unchanged. Each scenario has a greater number of events than the previous one. Table 4 shows the number of slow charging events, the number of fast charging events, the number of non executable charging events and the requested energy for every scenario.

Some limitations are introduced. The energy demand of every charging event at a specific scenario is the same so as the charging time of all events to be the same. In scenario 4, the input sequence is first a fast charging and then a slow charging and so on, while in scenario 7 it is the opposite. In scenario 5, the first action is the addition of 3 slow chargings consuming all the provided energy. The input sequence is important because the requested energy is committed only when the event starts charging. In addition, in these 3 scenarios the product of the total number of chargers per charging event's energy requirement is greater than the charging station's remaining energy which will be consumed before all chargers become occupied. Figure 19 demonstrates the total number of charging events that has been executed for each scenario. Figure 20 shows waiting lists' state after the insertion of all events.

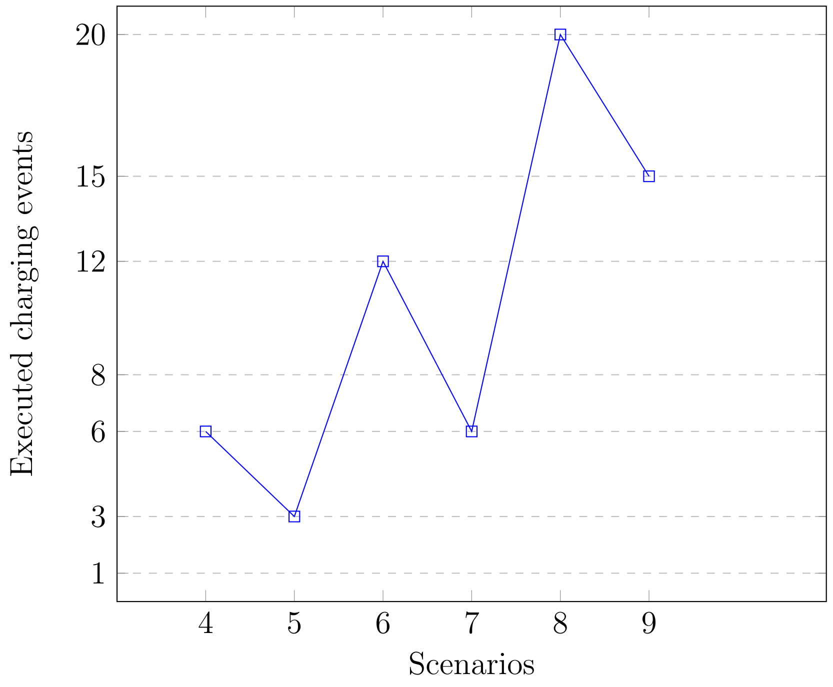


Figure 19: Charging events that have been executed

It is important to note, that the number of events that will be executed is always the (rounded if not integer) quotient of the total remaining energy in the charging station to the requested energy per charging event. If the number is rounded, the last executed event receives just a proportion of its requested energy. In scenarios 4, 6 and 8, an increase in the number of charging events to be executed is observed. In these cases, the descending requested energy per charging event makes the difference.

Figure 20 provides some interesting information. In scenarios 4 and 7, the requested energy per charging event is the same in both cases. In scenario 4, the input sequence consists of pairs of a fast charging event followed by a slow charging event. After the insertion of the third slow charging event, the number of available slow chargers and the remaining energy have been depleted. All remaining slow charging events enter the waiting list. On the

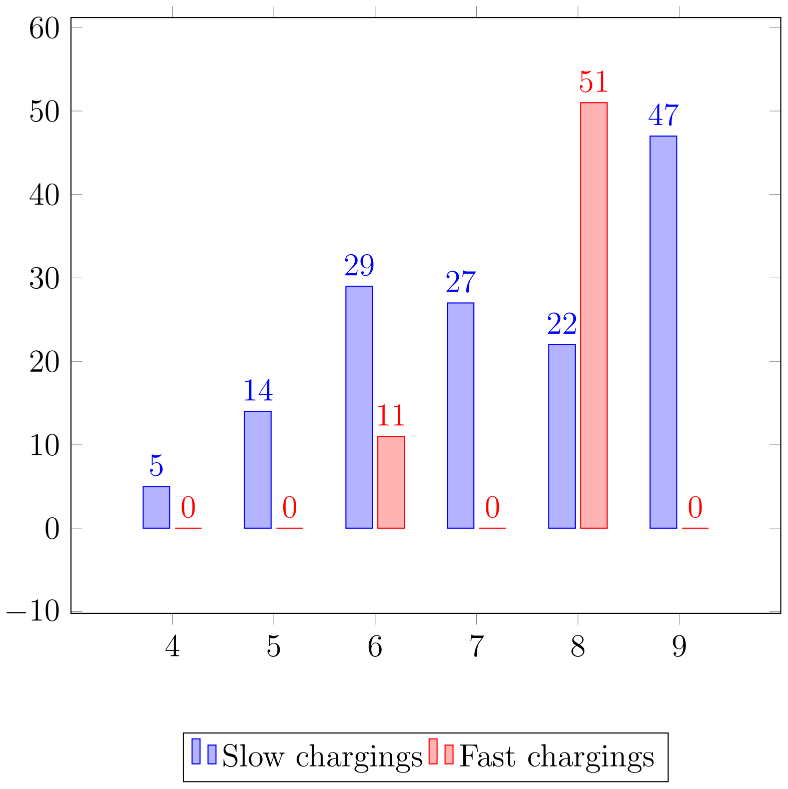


Figure 20: Waiting lists' state

other hand, for every subsequent fast charging there is an available charger, but not enough energy. These vehicles will not be charged and are declared as non executable. The same occurs for scenario 7. The waiting list for slow charging events contains the remaining slow charging events, while the fast charging waiting list is empty.

To sum up, assuming that there is not enough remaining energy in the charging station two cases exist: In case there is an available charger, then the event does not enter the respective waiting list and will never be executed. Instead, in case there is not any available charger, the event is inserted or remains in the waiting queue. Through these execution scenarios, we aim to outline the importance of EVLibSim in studying the performance of a station given different demand schemes and available resources. In this way, a charging station's owner can better configure the station based on prediction of EV demand.

7. Conclusions

As mentioned earlier in the text, Electric Vehicles is a continuous growing sector that is strongly related to the Smart Grid and the extended use of renewable energy sources. The bidirectional communication between EVs and the Smart Grid creates great opportunities for the IT and the power systems sectors. EVLibSim provides a simple, yet efficient interface for the management of all major EV-related activities such as the charging and discharging of batteries, as well as the battery swapping and parking, while using energy from renewable sources.

In developing the EVLib and the EVLibSim, a major research challenge was to design and develop both tools in such a way to be able to cover a wide range of technologies involved in charging/discharging of EVs. This is crucial so as to provide with a tool that would be able to cover a wide range of EV-related applications and be useful for a large number of researchers. Another consideration was the data that the tool will produce as output, as well as the format of them. This is important so as to assist researchers in easily comparing their work with others in the literature.

From a technical point of view, and regarding the EVLib library the challenge was to cover the larger possible set of EV-related entities and functions, while at the same time to provide a simple interface for the users. In terms of the simulation tool EVLibSim, the main challenge was to design a tool that provides the correct balance between the availability of the necessary features to construct and configure an EV charging station, while at the same time it is not over-constraint and can be useful in many different applications. Moreover, given that the EVLibSim uses the classes and functions of the EVLib, we tried to detach the simulation tool from the basic methodology, so as any changes to functions of the library to have little affect to the operation of the simulator. Later, the management of the parallel execution of many events was another issue to be taken care of. In this case, we used a multi-thread application in order to achieve a close to reality handling of events.

The main modelling consideration was on deciding on the main actors of the model. These are the "Charging Station", the "EV", the "chargers", "dis-chargers", "inductive chargers" and "battery swappers", the "Energy", and the "Events". After selecting the main actors, the attributes of them should also be selected. For example, a charger has a "charging rate", or an energy source has a number of available "energy units". In addition, we had to consider whether we would include the modelling of the electricity grid.

Given that the tool focuses on the charging station level, we decided to model only the basics that are needed for the operation of the stations such as the available energy per time point. The detailed modelling and simulation of the electricity grid is out of the scope of this work.

Another aspect of both tools is the extendability of them. Initially, the EVLib was developed. As discussed earlier, this library contains several classes that model all major EV-related actors such as the Charging Station, the EV, the Energy Source amongst others. Every class contains a large number of methods that handle different functions. The library itself is written in Java, it is opensource and has a detailed description of each class and function using Java docs. Thus, it can easily be used and extended to fit the needs of each user. As far as the EVLibSim is concerned, the functionality of the tool is based on the EVLib. For every option in the UI one or more EVLib methods are called and return the equivalent results. However, the UI is detached from the logic of the methods called. For example, the energy source that is being used to charge and EV can be selected with many different criteria, but the UI will present the result no matter which selection procedure has been used. The detachment of the simulation methodology from the basic operation of the simulation is an important contribution as the simulation can easily be re-used in case the library changes. The UI of the simulation tool was developed using Java FX which has rich documentation, and any extension can take place is a relatively straight forward manner. The tool itself is also opensource.

Finally, the decisions taken, and the knowledge obtained in modelling the EV charging station can be useful in other domains as well. For example, insights can be taken for the modelling and simulation of a smart building operating within a smart grid, or the simulation of electricity-powered unmanned aerial vehicles (UAVs). Both domains, although at first hand seem different, have many features in common, as in all cases the central actor of the system is an electric appliance needing to be powered or charged.

In the next section, ideas for future work are presented.

8. Future work

Concerning future improvements of the tool, congestion management can be faced up by using algorithms for optimal distribution of electric vehicles to the available charging points [29]. Also, the logs of charging and discharging events that EVLibSim keeps can be exploited using statistical packages.

EVLibSim would be able to provide better prices for frequent users. Moreover, for every charging station the energy demand and the number of energy storage updates can be better measured.

Future work will also focus on the integration of more advanced AI techniques, such as electronic markets, agent-based negotiation and coalition formation to further expand the library's and EVLibSim's capabilities [30]. Finally, mechanisms and techniques for managing uncertainty and increase fault tolerance will also be studied.

9. References

[1] Avl cruise. https://www.avl.com/cruise. Accessed: 05/06/2018.

[2] Caspoc. https://www.integratedsoft.com/Products/caspoc.aspx. Accessed: 05/06/2018.

[3] Der-cam. https://building-microgrid.lbl.gov/projects/der-cam. Accessed: 15/06/2018.

[4] Emcas. https://ceeesa.es.anl.gov/projects/emcas.html. Accessed: 05/06/2018.

[5] Future automotive systems technology simulator. https://www.nrel.gov/transportation/fastsim.html. Accessed: 05/06/2018.

[6] Homer. https://www.homerenergy.com/index.html. Accessed: 15/06/2018.

[7] V2g-sim. http://v2gsim.lbl.gov/home. Accessed: 05/06/2018.

[8] Susana Alegre, Juan V Míguez, and José Carpio. Modelling of electric and parallel-hybrid electric vehicle using matlab/simulink environment and planning of charging stations through a geographic information system and genetic algorithms. *Renewable and Sustainable Energy Reviews*, 74:1020–1027, 2017.

[9] A. Arancibia and K. Strunz. Modeling of an electric vehicle charging station for fast dc charging. In *2012 IEEE International Electric Vehicle Conference*, pages 1–6, March 2012.

[10] L. Bedogni, L. Bononi, M. Di Felice, A. D'Elia, R. Mock, F. Morandi, S. Rondelli, T. Salmon Cinotti, and F. Vergari. An integrated simulation framework to model electric vehicle operations and services. *IEEE Transactions on Vehicular Technology*, 65(8):5900–5917, Aug 2016.

[11] Aalok Bhatt. Planning and application of electric vehicle with matlab®/simulink®. In *Power Electronics, Drives and Energy Systems (PEDES), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[12] G. Cardoso, M. Stadler, M.C. Bozchalui, R. Sharma, C. Marnay, A. Barbosa-Pvoa, and P. Ferro. Optimal investment and scheduling of distributed energy resources with uncertainty in electric vehicle driving schedules. *Energy*, 64:17 – 30, 2014.

[13] Jean-Michel Contet, Franck Gechter, Pablo Gruer, and Abderrafiaa Koukam. *Bending Virtual Spring-Damper: A Solution to Improve Local Platoon Control*, pages 601–610. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[14] H. Farhangi. The path of the smart grid. *Power and Energy Magazine, IEEE*, 8(1):18–28, January 2010.

[15] Franck Gechter, Jean-Michel Contet, Stephane Galland, Olivier Lamotte, and Abderafia Koukam. Virtual intelligent vehicle urban simulator: Application to vehicle platoon evaluation. *Simulation Modelling Practice and Theory*, 24(Supplement C):103 – 114, 2012.

[16] Franck Gechter, Jean-Michel Contet, Pablo Gruer, and Abderrafiaa Koukam. Car-driving assistance using organization measurement of reactive multi-agent system. *Procedia Computer Science*, 1(1):317 – 325, 2010. ICCS 2010.

[17] Andreas Horni, Kai Nagel, and Kay W Axhausen. *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press, United Kingdom, 2016.

[18] IEA. Global ev outlook. Technical report, 2017.

[19] Sotiris Karapostolakis, Emmanouil S. Rigas, Nick Bassiliades, and Sarvapali D. Ramchurn. Evlib: A library for the management of the electric vehicles in the smart grid. In *Proceedings of the 9th Hellenic Conference*

on Artificial Intelligence, SETN '16, pages 13:1–13:4, New York, NY, USA, 2016. ACM.

[20] Willett Kempton and Jasna Tomic. Vehicle-to-grid power fundamentals: Calculating capacity and net revenue. *Journal of Power Sources*, 144(1):268 – 279, 2005.

[21] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.

[22] Khizir Mahmud and Graham E Town. A review of computer tools for modeling electric vehicle energy requirements and their impact on power distribution networks. *Applied Energy*, 172:337–359, 2016.

[23] W. J. Mitchel, C. E. Borroni-Bird, and L. D. Burns. *Reinventing the automobile: Personal urban mobility for the 21st century*. MIT Press, 2010.

[24] Michael A Nicholas, Gil Tal, and Justin Woodjack. California statewide charging survey: What do drivers want? *92nd Annual Meeting of the Transportation Research Board*, 2013.

[25] T. Novosel, L. Perkovi, M. Ban, H. Keko, T. Pukec, G. Krajai, and N. Dui. Agent based modelling and energy planning utilization of matsim for transport energy demand modelling. *Energy*, 92:466 – 475, 2015. Sustainable Development of Energy, Water and Environment Systems.

[26] Chitu Okoli and Suzanne Pawlowski. The delphi method as a research tool: An example, design considerations and applications. 42:15–29, 12 2004.

[27] Emmanouil S. Rigas, Sarvapali D. Ramchurn, and Nick Bassiliades. Algorithms for electric vehicle scheduling in mobility-on-demand schemes. In *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*, pages 1339–1344, Sept 2015.

[28] Emmanouil S. Rigas, Sarvapali D. Ramchurn, and Nick Bassiliades. Algorithms for electric vehicle scheduling in large-scale mobility-on-demand schemes. *Artificial Intelligence*, 262:248 – 278, 2018.

[29] Emmanouil S. Rigas, Sarvapali D. Ramchurn, Nick Bassiliades, and George Koutitas. Congestion management for urban ev charging systems. In *Smart Grid Communications (SmartGridComm), 2013 IEEE International Conference on*, pages 121–126, 2013.

[30] E.S. Rigas, S.D. Ramchurn, and N. Bassiliades. Managing electric vehicles in the smart grid using artificial intelligence: A survey. *Intelligent Transportation Systems, IEEE Transactions on*, 16(4):1619–1635, Aug 2015.

[31] Andreas Seitaridis, Emmanouil S Rigas, Nick Bassiliades, and Sarvapali D Ramchurn. Towards an agent-based negotiation scheme for scheduling electric vehicles charging. In *Multi-Agent Systems and Agreement Technologies*, pages 157–171. Springer, 2015.

[32] Giuseppe Marco Tina and Cristina Ventura. Simulation tool for energy management of photovoltaic systems in electric vehicles. *Energy Conversion and Management*, 78(Supplement C):851 – 861, 2014.

[33] Andrs Varga and Rudolf Hornig. An overview of the omnet++ simulation environment, 01 2008.

[34] Xiaomin Xi, Ramteen Sioshansi, and Vincenzo Marano. Simulation–optimization model for location of a public electric vehicle charging infrastructure. *Transportation Research Part D: Transport and Environment*, 22:60–69, 2013.

[35] Sung-Guk Yoon and Seok-Gu Kang. Economic microgrid planning algorithm with electric vehicle charging demands. *Energies*, 10(10), 2017.

[36] Dominik Ziemke, Kai Nagel, and Chandra Bhat. Integrating cemdap and matsim to increase the transferability of transport demand models. *Transportation Research Record: Journal of the Transportation Research Board*, (2493):117–125, 2015.