DEMKit: a Decentralized Energy Management Simulation and Demonstration Toolkit

Gerwin Hoogsteen, Johann L. Hurink, Gerard J. M. Smit
Department of EEMCS, University of Twente
Enschede, the Netherlands
{g.hoogsteen, j.l.hurink, g.j.m.smit}@utwente.nl

***Abstract*—The energy transition requires simulators that aid the research on and validation of energy management systems. Hereby, these systems not solely focus on electricity, but involve the optimization of multiple energy carriers in a multi-energy system. This paper presents DEMKit, an open-source toolkit with device, grid and control components for such a multi-energy management solution. The modular design allows DEMKit to test different optimization algorithms on the same scenario. Device components can be replaced by adapters to control real hardware in a deployment or hardware-in-the-loop simulation. Results of a demonstration setup of DEMKit in a smart home show the potential of using DEMKit to bridge the gap between scientific research and test new solutions in practice.**

***Index Terms*—simulation, smart grid, modeling, micro grid**

I. Introduction

The energy transition is one of the greatest challenges that our society has to accomplish. Smart grid technology is the key enabler to support the integration of renewable energy sources (RES). With this technology, electricity production and usage can be matched, e.g. by applying decentralized energy management (DEM) systems. Electricity is only one of the relevant energy carriers, other carriers such as heat and (bio) gas also play an important role.

In order to support the energy transition, innovative energy management systems (EMS) and scenarios have to be developed. However, before implementing such new approaches, it is important to evaluate them thoroughly to assure correctness of their operation. To carry out such an evaluation, tools are required in which the developed control concepts can be simulated and validated. Such a tool must encompass the support for multiple energy carriers to study the interplay and conversion between the different energy carriers, so-called *multi-energy systems* (MES). Furthermore, hardware-in-the-loop (HIL) simulation capabilities are required to test and validate developed EMS concepts in practice.

Many simulation tools and frameworks have been proposed in literature the last few years. A popular tool is EnergyPLAN [1], which allows for macro-scale modeling and optimization of a multi-energy system to evaluate different scenarios. The Mosaik framework [2] is a simulation platform in which other models and simulators can be linked together to perform co-simulation. A novel co-simulation framework with HIL capabilities that integrates with a smart home system is presented by Kochannek et al. in [3].

Next to a HIL (co-)simulation framework, scenarios consisting of usage patterns of neighbourhood power consumption, production and mathematical constraints are required to carry out analysis. Large scale deployment of key enabling technologies, such as electric vehicles (EVs) and battery storage systems, are not yet available. Therefore, to analyse the energy system of the future, artificial load and generation profiles are required for such a framework. Various tools for generation of such artificial energy usage patterns are proposed, such as [4]. However, these tools focus on load curves only and result in a static profile. Hence they lack the operation constraints, such as the minimum charge and charging deadline for an EV, that are required as input for optimization algorithms. This problem is tackled with the open-source Artificial Load Profile Generator (ALPG) [5].

This paper presents the software architecture behind the simulation and demonstration framework for future multi-energy control systems: DEMKit (short for Decentralized Energy Management Toolkit). The focus of DEMKit is to provide the tools to analyse optimization algorithms for MES using discrete time simulations. Hence, the focus is on the implementation of lightweight optimization algorithms, tailored to abstract device models that describe the operation constraints. DEMKit is the successor of the TRIANA simulator presented in [6] and leverages the knowledge developed in the last decade. In contrast to e.g. the Mosaik framework [2] DEMKit does provide the optimization algorithms, and compared to EnergyPLAN [1] it models a MES using individual devices. Note that the focus is on testing optimization algorithms and not to analyse and detect phenomena such as faults.

DEMKit (written in Python) offers the tools to use the same code-base and environment for pure simulation studies, HIL validations, and deployment of control mechanisms in real life systems. The remainder of this paper presents the software architecture and concepts in order to do so. Due to the tight integration of components for simulation and deployment, researchers benefit from availability of models and measurement data from field tests, while state-of-the-art optimization algorithms can be deployed in test-sites easily. Alternatively, DEMKit also supports importing ALPG data. Initial results of a smart house running this platform are presented in Section IV. DEMKit is available with an open-source license for research purposes on request. We refer the reader to [7] for more information and an overview of studies and projects that make use of the DEMKit software.

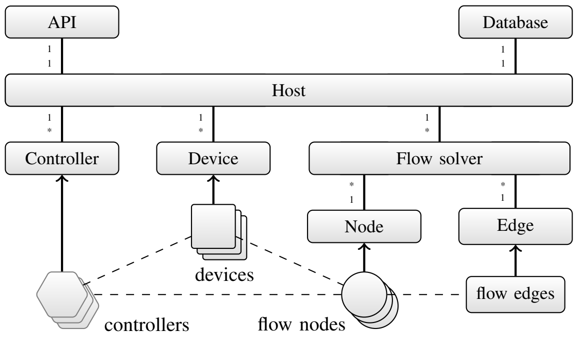


Fig. 1. Diagram of DEMKit with object references between devices (squares), controllers (hexagons) and infrastructure (circles) in dashed lines [8].

II. DEMKit Architecture

The goal of DEMKit is to provide a toolkit for the development of intelligent multi-energy concepts and their control systems. An important requirement is to provide a smooth transition path from system validation through simulation studies towards testing components using HIL simulations in demonstration projects. Therefore, a broad range of energy carriers, their infrastructure, and optimization concepts need to be supported. Furthermore, the tool needs to be scalable to run both small scale demonstrators and large scale simulations.

To accomplish this, the DEMKit architecture follows a bottom-up modeling approach where each device, controller (or agent) and piece of physical infrastructure is modeled individually (Fig. 1). Herein, a cyber-physical systems architecture is used for a strict separation between control algorithms and physical (device) models. This separation ensures that the device models can be replaced by their real-world equivalent for HIL simulation, without the need to replace the cyber part. Each of these models is called a *component* and multiple unique instances of a component can exist within a scenario. Each instance of such a component is called an *entity*. Subsequently, entities are linked together by means of object or unique name references to allow interaction between e.g. devices and control algorithms. Such a composition of entities and their references results in a *scenario*, which is a model of e.g. a household or neighbourhood. More details on the different types of components and their interfaces to allow interaction between entities are given in Section III.

A. Platform Host System

The most notable class of the DEMKit framework is the *Host* class, which represents the (virtual) system on which the framework is running and provides a basic interface for the simulation and entities. Standard functionality of a host includes read- and write access to a database and signals when the (virtual) system boots or shuts down. Also the current time is requested through this interface such that the host can return either the simulated time or the actual system time depending on the configuration. In both cases a UNIX-timestamp is returned, which ensures that a simulation scenario can easily be reconfigured to run in a demonstrator. The host contains object references to all entities within the scenario.

Another aspect is parallel processing on a single scenario to e.g. demonstrate the scalability of optimization approaches or to deploy a system that has to control devices in different physical locations. A scenario can be decomposed into multiple smaller scenarios mapped to different processes and/or (virtualized) computer systems. The different host instances take care of inter-host communication, for which websockets or TCP/IP communication is used. Each host provides an API that allows external processes to access variables and functions of any entity associated to the local host. All calls are handled through the host class, which figures out whether an entity exists locally or externally. Function calls on multiple external entities are processed automatically in parallel.

B. Input and Output Format

A configuration of a scenario, describing which components to instantiate and how to link them, is required to execute a simulation. Many tools support generic configuration file formats, such as JSON or XML, together with an user interface to create a scenario. However, manually creating larger scenarios is time consuming and involves many repetitive tasks. For this reason, we use Python scripts as input for DEMKit scenarios such that automated generation of scenarios is possible. Components are imported in these scripts and entities can be created, of which parameters can be configured. Furthermore, predefined building blocks of e.g. devices with controllers and complete households are available to quickly compose a scenario. Data generated by the open-source ALPG [5], in the form of load patterns and optimization constraints, can be imported directly into such a scenario.

Each entity produces valuable output for each time step of a simulation, such as the current state or power consumption. The open-source InfluxDB [9], database is used for data storage, which is designed to store time-series data from e.g. measurement sensors without the need to define a storage scheme on beforehand. Functionality is added to DEMKit to easily retrieve and store data in this database. This way, data resulting from experiments can be imported back into simulation models to re-evaluate certain circumstances with new optimization mechanisms. Results can be visualized with external tools such as Grafana [10], which allow for user-friendly querying, analysis and live visualization of results. Grouping and aggregation of data is possible with Grafana to produce high level overviews and compare the most important metrics for a scenario.

C. Simulation Flow

Next to basic functionality, the host also controls the order in which high-level functions are called within a simulation. This flow dictates in which order entities progress to a (partial) new state to avoid synchronization and dependency issues between e.g. control signals issued by the optimization algorithms and the reaction of devices to these (new) signals. The framework makes use of one synchronization clock with a fixed discrete time step (minimum of 1 second). Each component decides whether it needs to act in a certain interval. In this way, multiple configurable time step sizes can be mixed, such as

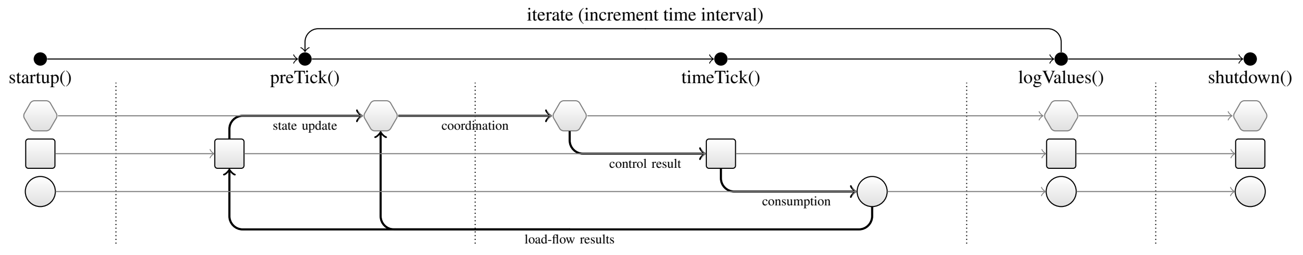


Fig. 2. Diagram of the DEMKit simulation flow and state transitions (grey lines). It also shows the information flow between components throughout a simulation interval (black lines). Devices components are visualized by squares, controllers by hexagons, and grid components by circles.

e.g. devices updating every minute, and optimization every 15 minute intervals to limit the required computing power. The simulation flow and functions, which make up the basic interface implemented by each DEMKit component (Fig. 2), and discussed in the following.

After the complete scenario is loaded (entities are instantiated and linked), the host signals the *startup*-command to all entities to perform initialization steps and parameter checks. After the startup the environment is ready to commence normal operation. For a simulation, the host starts a for-loop to simulate time progression based on the configured simulation time step size and number of intervals. In case of a HIL simulation, the host will periodically send time updates based on the predefined time step value. Within a simulation interval, a *preTick* is issued first to all entities to update their state, e.g. the state of charge of a storage system based on the power consumption set in the previous interval. Then, a *timeTick* is communicated to control entities, which can make decisions based on the current state of the device entities. Subsequently, the device entities receive a *timeTick* to change their consumption based on their behavioural description or control commands. Finally, a load-flow analysis can be executed to determine the steady state of the physical grid infrastructure. At the end of each time step, the *logValues* command is communicated to store statistics in the database. A *shutdown* signal is broadcast over all components when the execution comes to an end.

III. DEMKit Components

Components form the building blocks within DEMKit to create a scenario. Three generic types of building blocks are defined: devices, controllers and energy distribution infrastructure. The motivation for this choice is that the explicit separation between physical models and optimization methods (controllers) makes it possible to swap between optimization algorithms or device components. In this way, different granularities of detail can be used. Furthermore, the separation between device and control allows a smooth transition from simulations to demonstrations, where control algorithms can be tested in practice by replacing device models with adapters that connect to the real device.

1) Device Components: a set of generic *device* components are available that describe the behaviour of a device and its parameters. These components include: smart meters, uncontrollable loads, buffers, etc. as indicated in Table I. The available components not only concern electrical devices, but are abstract device classes similar to [12], that also represent devices using the other energy carriers. For example, a buffer can be used to model a battery or heat storage buffer. Each device component serves as a data container and interface that exposes the current state of the device and provides functions to retrieve historical (measurement) data.

For each device a basic behavioural description, which describes how the device operates when no external control is applied (i.e. only internal logic) is defined. A device is simulated using discrete time steps, triggered through the *timeTick* function call. Hereby, as basis often a greedy strategy is used, e.g. start the timeshiftable (e.g. a dishwasher) as soon as it becomes available or charge an EV as fast as possible. Advanced models, such as a thermal zone, are implemented by inheriting one of the base device classes. For an overview of devices and their models we refer to [8, Chapter 3].

For interfacing with controllers, each device component implements the *setPlan*-function. If a desired power value for the current time interval is available, the device brings itself in a state that satisfies this planned power value. However, the device model is leading, meaning that a device will not follow the planned value when it is unable to do so. Assertions are in place to verify whether comfort constraints are met when desired (i.e. when load shedding is not allowed).

2) Grid Components: physical flow of energy can be simulated using grid components that model e.g. cables and pipes, for which the topology is described by a graph. Load-flow solvers access this graph to solve a flow problem. Power can be consumed or produced at each node by coupling a meter entity, which aggregates the power consumption of multiple devices, to a node. Through the interface, the connected components can also obtain the results from the load-flow calculations, e.g. to check if the voltage is within permitted limits. Currently, DEMKit contains an integrated load-flow solver for three phase unbalanced radial electricity networks.

3) Optimization Components: control systems form the cyber-part of DEMKit and implement the logic for data processing, evaluation, and optimization. All controllers implement the generic DEMKit simulation interface. Device controllers also interface with devices to read historical data and the current state and write back optimization results. Five control strategies are currently (being) implemented in

TABLE I
IMPLEMENTED COMPONENT CLASSES AND SUPPORTED CONTROL AND OPTIMIZATION FUNCTIONALITY [8].

Class	**Example devices**	**Online control**	**Prediction**	**Offline optimization**	**Grid limits**	**Curtailment / load shedding**	**Multiple commodities**
Uncontrollable	Baseload		1, 3, 4, 5				
Curtailable	Solar panels, wind turbine	2, 3	1, 3, 4, 5	1, 3, 5	1, 2, 3, 5	1, 2, 3, 5	1, 5
Timeshiftable	White goods	2, 3	1, 3, 5	1, 3, 5	1, 2, 3, 5	1, 2, 3, 5	
Buffer	Battery, heat store	1, 2, 3, 5		1, 3	1, 2, 3, 5	1, 2, 3, 5	1, 5
Buffer-Timeshiftable	Electric vehicle	2, 3, 4	1, 3, 4, 5	1, 3, 4, 5	1, 2, 3, 5	1, 2, 3, 5	1, 5
Converter	Combined heat and power	2, 3			2, 3	2, 3	
Buffer-Converter	Heat pump with storage	2, 3	1, 3, 5	1, 3, 5	1, 2, 3, 5	1, 2, 3, 5	1, 5
Controllers	Energy management system	1, 2, 3, 4	1, 3, 4, 5	1, 3, 4, 5	1, 2, 3, 5	1, 2, 3, 5	1, 5

Currently supported: 1. Profile Steering [11], 2. Double-sided auction [12], 3. Planning based auction, 4. Valley-filling [13], 5. RTP optimization.

DEMKit: the profile steering approach presented in [11], the double-sided auction presented by Kok [12], the combination of profile steering and the double-sided auction (*planning based auction*), the valley-filling approach [13], and Real-time Pricing (RTP) [14], in which the energy prices for the next hours or days are published in advance. In all these approaches, the control structure is formed by a tree, in which the top level (root of the tree) coordinates the whole cluster. Controllers interact with each other through object references in a parent-child(ren) structure. All functions required for control, such as performing predictions or placing bids, are implemented in the control level to preserve the strict separation between device components as a storage container and control components that work with the data. Specific optimization algorithms, such as presented by Van der Klauw [15] are implemented in device controllers. With the use of generic device classes and state variables, it is possible to use an optimization algorithm written for a generic buffer for both electrical and heat storage systems. Table I presents a detailed compatibility matrix of different features of different control mechanisms and supported device types.

IV. EXPERIMENTS

As stated in the introduction, the goal of DEMKit is to provide a software platform for researchers to create, experiment and validate control mechanisms. This section briefly describes how a simulation scenario can be generated and then presents a test-case where DEMKit is deployed in a smart house. We refer the reader to [7] for a comprehensive list of simulation studies and projects executed using DEMKit.

A. Simulations

A scenario within the DEMKit framework is the instantiation and composition of multiple entities, together with the required input data for e.g. parameters and time series data. To aid the process of setting up a (large scale) scenario, DEMKit is able to import data generated by the ALPG [5]. This open-source tool specifies operation constraints and demand patterns using high-level input data. With recent updates, the ALPG also supports heat models based on the work by Van Leeuwen [16] next to electricity profiles. The output produced by the ALPG is provided in the form of generic flexibility classes as implemented in DEMKit and provides a solid basis, which may be fine-tuned to the case-study at hand. Based on this scenario, (HIL) simulation studies can be performed with different optimization algorithms and objectives, and compare the results.

B. Smart house demonstrator

To demonstrate the performance of the implemented algorithms within DEMKit, a demonstration setup was created within a smart home. The goal for the optimization system is to minimize the power peaks of the household, i.e. we minimize $\sum_{t=1}^{T} P_t^2$, where P_t is the power import/export of the house at time t, and where T is the number of intervals. Creating this setup started with creating a scenario, depicted in Fig. 3, of the household within DEMKit and loading historical measurement data of the load and of solar panels. To add flexibility, a 5 kWh battery was added to this scenario. Model predictive optimization algorithms ([11], [17]) are added to the scenario to verify the operation of the smart house in several conditions.

After model based verification, the scenario is modified for demonstration. For this, the device models are replaced by components that interface with the devices installed in the house. The load device component is replaced by a component that reads out meters using web requests, whereas the solar panel model is replaced by a component to read out an inverter. Additionally, weather data is accessed through the web to obtain weather forecasts from Solcast [18], which are used as input to create prognosis for the PV production. Due to lack of a real battery, the battery is modeled using a virtual device. A washing machine is added to the system for additional flexibility, which can be delayed using a controllable smart

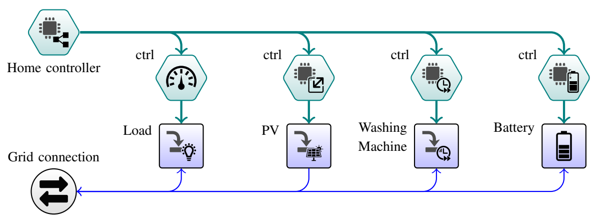
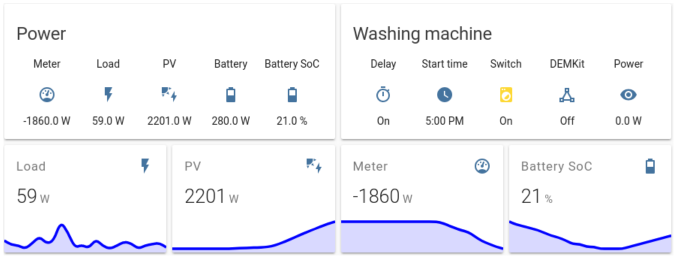


Fig. 3. DEMKIt diagram of demonstrator setup, with all devices (squares), controllers (hexagons) and connection to the grid. The blue lines indicate electricity flows, the green lines indicate communication between controllers.


Power

Meter	Load	PV	Battery	Battery SoC
-1860.0 W	59.0 W	2201.0 W	280.0 W	21.0 %

Washing machine

Delay	Start time	Switch	DEMKit	Power
On	5:00 PM	On	Off	0.0 W

Load 59 W

PV 2201 W

Meter -1860 W

Battery SoC 21 %

Fig. 4. User interface in Home Assistant to interact with the system.

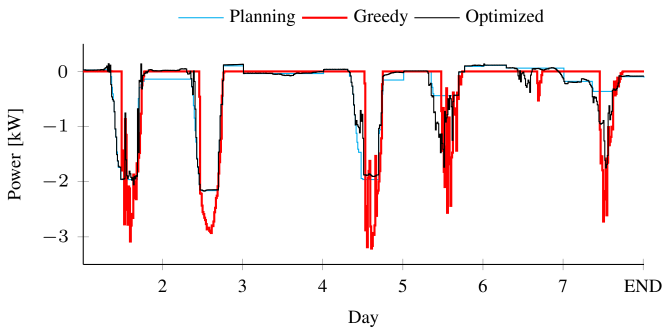


Fig. 5. Resulting power profile for the two control strategies.

power socket. For communication with devices, and to provide a user interface, the open-source home automation software Home Assistant [19] is used. An interface (see Fig. 4) is developed to control the washing machine and monitor the control system. A second HIL simulation runs in parallel, using a greedy strategy for the battery to compare the results.

Analysis of running the control system for over a month show that DEMKit is able to optimize the power profile using model predictive optimization. Prediction of electricity production from PV turns out to be difficult for this setup. Updating the irradiance forecasts every 30 minutes, and performing a new optimization does not always result in an improvement. On some days, the more recent forecasts result in a larger error, as the original day-ahead forecasts proof to be more accurate. As a countermeasure we implemented a strategy to not completely use the battery capacity during day-ahead optimization, leaving headroom for online-control of the battery to resolve the prediction errors. The deployment of this update, after verifying effects using simulations, have resulted in a flatter power profile. The results between the 21st and 28th of March 2019 lead to an RMSE w.r.t. the objective value of 715 for the optimization using [11], which is a significant improvement over the greedy strategy, for which the RMSE is 869. Fig. 5 shows the resulting power profile. The greedy control of the battery leads to a full battery around noon, resulting in a peak production of 3.6 kW. The predictive optimization method is able to shave off the peak production to 2.2 kW through robust scheduling of the battery.

V. Conclusion

This paper presented the DEMKit simulation and demonstration framework for multi-energy systems. The developed and presented framework is flexible to support new technologies and a wide variety of optimization mechanisms. In this way these approaches can be compared to each other. For the generation of such scenarios, the open-source ALPG is used. Experiences from various projects using DEMKit show that it is flexible to prototype and validate new energy and control concepts. For more information and contact details concerning DEMKit we refer the reader to [7].

Acknowledgment

We thank all students and staff involved at the University of Twente and partners for their testing, bug reports and effort put into the development of DEMKit. This work is supported by Dutch national program TKI iDeego (project ORTEP).

References

[1] EnergyPLAN, [Online] Available: http://www.energyplan.eu, [Accessed: 01-Apr-2019].

[2] S. Schütte, S. Scherfke, and M. Tröschel, “Mosaik: A framework for modular simulation of active components in smart grids,” in *2011 IEEE First International Workshop on Smart Grid Modeling and Simulation (SGMS)*, October 2011, pp. 55–60.

[3] S. Kochanneck, I. Mauser, K. Phipps, and H. Schmeck, “Hardware-in-the-loop co-simulation of a smart building in a low-voltage distribution grid,” in *IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), Sarajevo, Bosnia and Herzegovina*. IEEE Power & Energy Society, October 2018.

[4] N. Pflugradt, J. Teuscher, B. Platzer, and W. Schufft, “Analysing low-voltage grids using a behaviour based load profile generator,” in *International Conference on Renewable Energies and Power Quality*, vol. 11, 2013, p. 5.

[5] G. Hoogsteen, A. Molderink, J. L. Hurink, and G. J. M. Smit, “Generation of flexible domestic load profiles to evaluate demand side management approaches,” in *2016 IEEE International Energy Conference (ENERGYCON), Leuven*, April 2016, pp. 1–6.

[6] V. Bakker, “Triana: a control strategy for smart grids: Forecasting, planning & real-time control,” Ph.D. dissertation, University of Twente, Enschede, January 2012.

[7] “University of Twente: Energy in Twente,” [Accessed: 01-Apr-2019]. [Online]. Available: https://utwente.nl/en/eemcs/energy/

[8] G. Hoogsteen, “A cyber-physical systems perspective on decentralized energy management,” Ph.D. dissertation, University of Twente, Enschede, December 2017, CTIT Ph.D. thesis series no. 17-449.

[9] InfluxData, [Online] Available: https://influxdata.com, [Accessed: 01-Apr-2019].

[10] Grafana Labs, [Online] Available: https://grafana.com, [Accessed: 01-Apr-2019].

[11] M. E. T. Gerards, H. A. Toersche, G. Hoogsteen, T. van der Klauw, J. L. Hurink, and G. J. M. Smit, “Demand side management using profile steering,” in *PowerTech, 2015 IEEE Eindhoven*, June 2015, pp. 1–6.

[12] K. Kok, “The PowerMatcher: smart coordination for the smart electricity grid,” Ph.D. dissertation, Vrije Universiteit Amsterdam, July 2013.

[13] M. E. T. Gerards and J. L. Hurink, “Robust peak-shaving for a neighborhood with electric vehicles,” *Energies*, vol. 9, no. 8, 7 2016.

[14] M. Albadi and E. El-Saadany, “A summary of demand response in electricity markets,” *Electric Power Systems Research*, vol. 78, no. 11, pp. 1989 – 1996, 2008.

[15] T. van der Klauw, M. E. T. Gerards, and J. L. Hurink, “Resource allocation problems in decentralized energy management,” *OR Spectrum*, vol. 39, no. 3, pp. 749–773, Jul 2017.

[16] R. P. van Leeuwen, “Towards 100% renewable energy supply for urban areas and the role of smart control,” Ph.D. dissertation, University of Twente, Enschede, May 2017, cTIT Ph.D. thesis series No. 17-433.

[17] T. van der Klauw, “Decentralized energy management with profile steering - resource allocation problems in energy management,” Ph.D. dissertation, University of Twente, Enschede, The Netherlands, May 2017, cTIT Ph.D. thesis Series No. 17-424.

[18] Solcast, [Online] Available: https://solcast.com.au/, [Accessed: 01-Apr-2019].

[19] Home Assistant, [Online] Available: https://home-assistant.io, [Accessed: 01-Apr-2019].