European Commission

JRC TECHNICAL REPORT

Smart Grid Interoperability Laboratory A toolkit for smart energy management

Towards the smart house

Covrig, C.F.

Munoz Diaz, M.A.

Georgiopoulos, S.

Marinopoulos, A.

2020

Joint Research Centre

EUR 30211 EN

EU Science Hub
https://ec.europa.eu/jrc

JRC120540

EUR 30211 EN

PDF ISBN 978-92-76-18820-9 ISSN 1831-9424 doi:10.2760/822668

Luxembourg: Publications Office of the European Union, 2020

How to cite this report: Covrig, C.F., Munoz Diaz, M.A., Georgiopoulos S. and Marinopoulous, A., *Smart Grid Interoperability Laboratory: A toolkit for smart energy management*, EUR 30211 EN, Publications Office of the European Union, Luxembourg, 2020, ISBN 978-92-76-18820-9, doi:10.2760/822668, JRC120540, EUR 30211 EN.

Contents

Abstract 2
1 Introduction 3
2 The Smart Grid Interoperability Laboratory 4
2.1 Interoperability definition 6
3 An open toolkit for Energy Management Systems 7
3.1 Energy Management System definition 8
3.2 Introduction to PyEMS 8
4 PyEMS insight 11
4.1 Main features 11
4.2 Modes of operation 11
4.3 Time in PyEMS 12
4.4 Units of physical magnitudes 12
4.5 Main abstract entities 12
4.5.1 Simulation 13
4.5.2 System 14
4.5.3 System Components 14
4.5.4 Forecaster 14
4.5.5 Data handler 15
4.5.6 Optimizer 15
4.6 Forecasting PV generation and building load 15
4.6.1 Introduction to the problem 15
4.6.2 PV generation forecasting 16
4.6.3 Building consumption forecasting 22
4.7 Optimization model 25
5 Technical aspects 28
5.1 Installation 28
5.2 Deployment 28
5.3 Code Structure 29
5.4 Implementation examples 30
6 Conclusion 34
References 35
List of abbreviations and definitions 37
List of figures 37
List of tables 39

Authors

Catalin Felix Covrig

Miguel Angel Munoz Diaz

Antonios Marinopoulos

Stavros Georgiopoulos

Aknowledgments

Thomas Seldis

Ulrik von Estorff

Abstract

In this technical report, we introduce a Python based toolkit intended to act as an Energy Management System (EMS) for Smart Buildings. EMS are called to play an important role in the buildings of the future by addressing interoperability problems, improving energy efficiency and user comfort. The package presented in this report provides the mathematical and data pre-processing tools to leverage existent Home Automation software into fully-fledged EMS. The package is developed in the Smart Grid Interoperability Laboratory (SGIL) of the JRC in Petten, the Netherlands, where it has been tested. An example that is deployed in the laboratory is provided in this report.

1 Introduction

The demanding environmental challenges our society is facing, appeal for greener and more efficient energy usage in our homes and buildings. Renewable energy (e.g. solar, wind) paves the way to solve these issues. The intermittent availability of renewable energy due to factors like lack or excess of sun or wind, could be addressed with the introduction of grid storage that will help in balancing the grid by providing or capturing the missing/excess power.

At the same time, the proliferation of smart devices, electric vehicles, battery energy storage and the capability to produce and store their own electricity, transforms the consumers into active flexible prosumers. The very same reason also raises new interoperability issues regarding the coordination and joint operation of these devices on both inside and outside of smart houses, i.e. in the overall smart grid. These interoperability issues come from the amalgamation of protocols and technologies, smart equipment currently use.

Interoperability is the ability of a device to be integrated in a system and exchange useful information, understand the information exchanged and comply with the system rules maintaining the quality of service. It is a fundamental element for the Smart Grid where integration is a key challenge affecting components, information, systems and implementations. Next generation electricity grids will integrate interoperable technologies – particularly in the energy, transport, information and communication fields – with the aim to increase reliability, affordability and sustainability of grid and market operations.

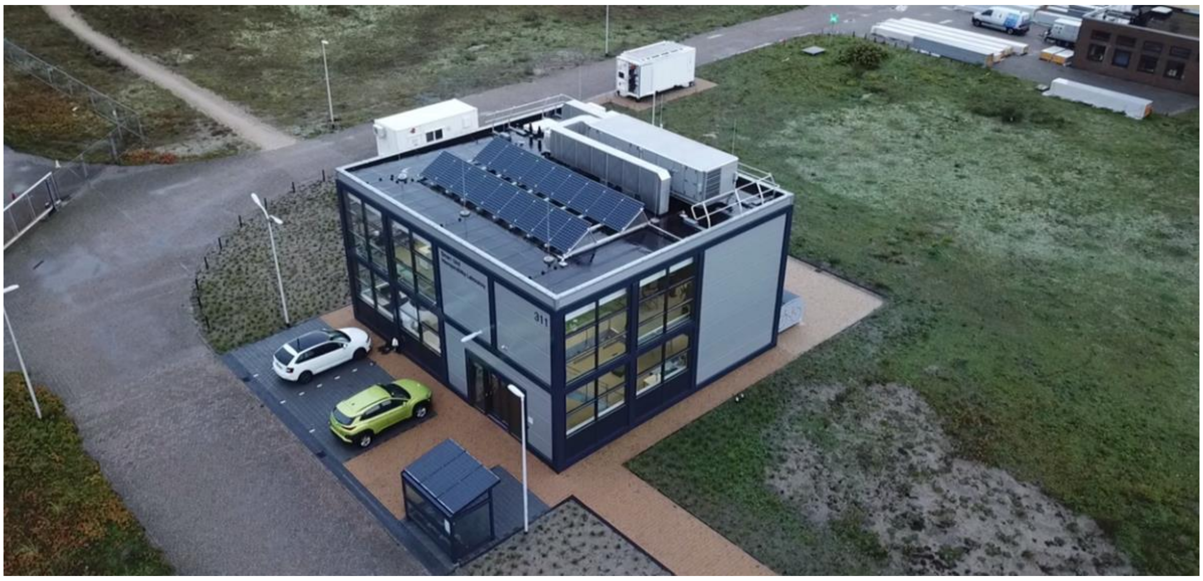
In this context, the implementation of smart systems in our buildings is a necessary task with a threefold outcome, namely: an enhanced energy management, a reduction of interoperability conflicts and an improvement in user comfort level. Home Automation and Energy Management Systems are called to play this role in the houses and buildings of the future.

In this report, we present a toolkit based on Python programming language allowing users to develop tailored EMS for smart buildings. This package makes use of optimization techniques to co-ordinately take decisions on device operation of the building or system considered (like the state of charge of batteries). Furthermore, the package eases data collection and pre-processing by automatically issuing forecasts of energy production and consumption based on past and present information and future predictions. This tool has been developed in the Smart Grid Interoperability Laboratory [26] of the JRC in Petten.

2 The Smart Grid Interoperability Laboratory

The Smart Grid Interoperability Laboratory [26] is located in the Joint Research Centre (JRC) site in Petten, Netherlands. The scope of this newly developed testing facility is the interoperability of smart grid systems, and along with other JRC facilities aims to contribute to policymaking and industrial innovation towards the modernization of the electricity grid. In specific, the goal of the laboratory is to assess different technological implementations of smart systems, mainly in a home environment, according to relevant standards and processes, by means of commonly agreed use cases and applicable reference architectures. The technical activities include the verification of interoperability among home appliances and grid components, the benchmarking of different solutions, and the identification of gaps and challenges.

Fig. 1. Smart Grid Interoperability Lab



The laboratory develops and implements a common EU methodology for testing interoperability of smart grid components and systems [19], following experimental procedures, simulations and emulations, while using the respective standards. In some cases, the testing can even extend to the point of identifying factors that could potentially compromise interoperability. It is important to note here that, the assessment of interoperability has to be done with reference to use cases and assumptions stipulated by industry and standardization bodies (e.g. CEN-CENELEC), to allow for validation of results. Therefore, a large portion of the work is performed in collaboration with industry and research institutions.

Among other devices, the lab is equipped with:

- two solar photovoltaic (PV) installations (one on the roof of the lab building and one on a nearby car port of total installed capacity of approximately 80 kWp)
- two 75kW/150kWh Battery Energy Storage Systems (BESS) installed in outdoor containers
- a microgrid setup, equipped with a 15kW/45kWh indoor BESS
- a real time simulator
- an amplifier and a load simulator
- smart appliances and devices (washing machine, dryer, dishwasher, oven, smart plugs, bulbs, and sensors, etc.)
- a (smart) heat pump
- five electric bicycles and a bike charging station
- a fully electric vehicle
- a Vehicle to Grid (V2G) charging station with ±10kW charging power

Fig. 2. Smart Grid Interoperability Lab facilities

There are many stakeholders who can potentially benefit from the SGIL either directly by being involved in the work activities, or indirectly by the contribution of the outcomes to better policymaking and research and innovation. Manufacturers of smart appliances will benefit from a less fragmented market that opens up global market opportunities and reduces production cost due to economy of scales. Promotion of open standards, through identification of gaps or misalignments and recommendations for further global harmonisation, is also beneficial both for manufacturers and for consumers, who will have more choices for products without restriction by closed proprietary ecosystems. Through the dissemination of testing results, the public will have access to reliable information on interoperable energy-related products and services. Thus, the consumers will be more certain for the plug-and-play functionality of digital energy solutions and the role that these can have in increasing energy efficiency and allowing participation in the energy market. Last but not least, the grid operators could also harvest the benefits from the outcomes of smart grid interoperability testing, e.g. to better integrate distributed energy resources, or explore new opportunities for business models and services.

In conclusion, the Smart Grid Interoperability Laboratory in JRC Petten has a clear objective: to promote the interoperability of digital energy in the interface between smart homes and smart grids. In order to do this, the laboratory:

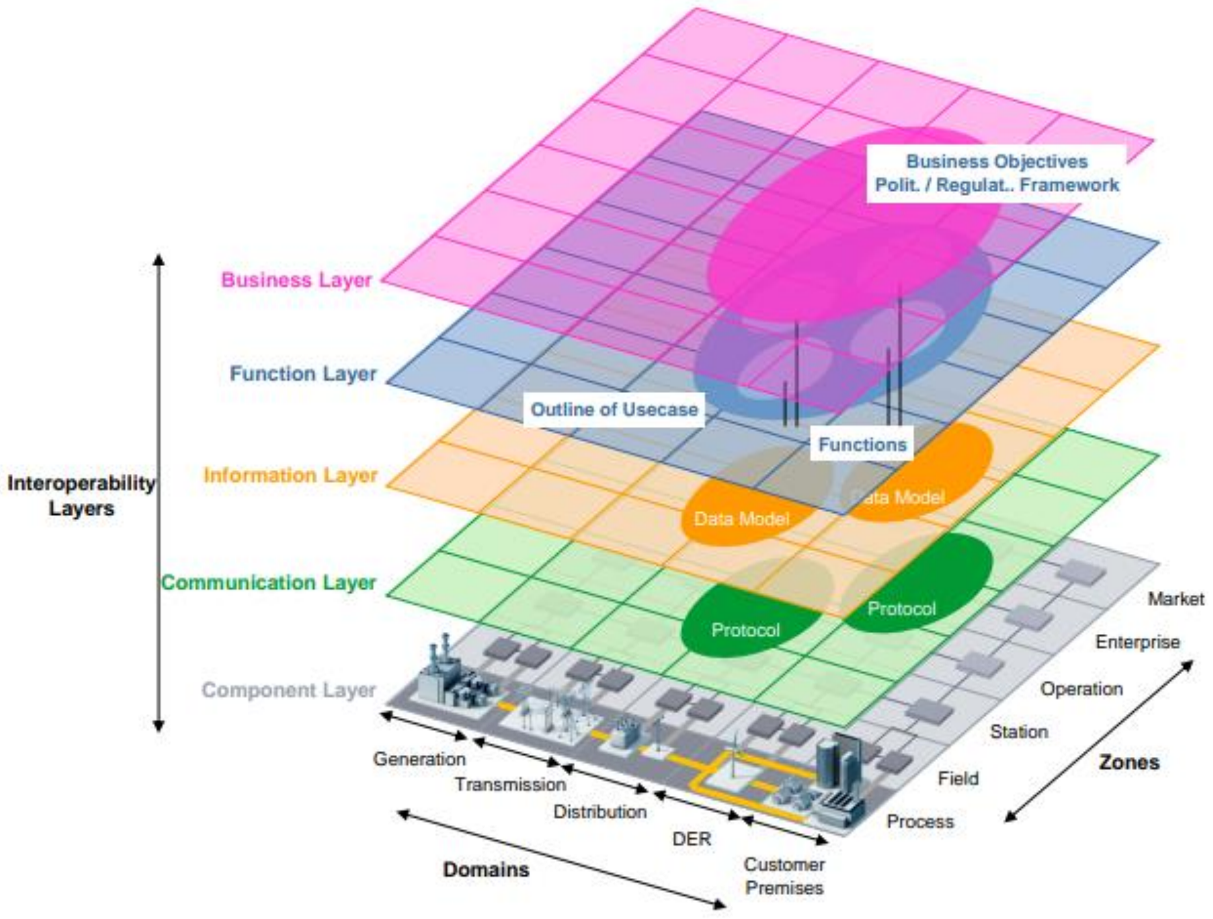
- Tests the interoperability of solutions, from market and research projects.
- Promotes the use of a common interoperability testing methodology based on the CEN-CENELEC-ETSI framework [19].
- Networks with other European laboratories and research centres for common initiatives.
- Networks with European industrial actors in various sectors.
- Disseminates the results of testing campaigns.

Fig. 3. Electric vehicle and Vehicle to Grid (V2G) charger

2.1 Interoperability definition

The Smart Grid exhibits is a highly complex system in terms of organizational and technological aspects, as the Smart Grid Architecture Model (SGAM) illustrates, see Fig. 4. Some of the key challenges towards a well-functioning smart grid ecosystem are the integration of systems, components, information, and applications. Common interfaces and functionality must be ensured for enabling high-level processes. The interconnection of every smart grid component will create a network in which communication and analysis take place in near real-time. Therefore, for the transition from the classical power system towards a modernized smart grid, it is necessary to exploit all the capabilities of Information and Communication Technologies (ICT), such as Machine to Machine Communication, Agent technologies, and Internet of Things.

Fig. 4. SGAM framework [6]



As far as the smart grid migration process is concerned, interoperability is an essential requirement. Due to the large scale of the future interconnected smart grid and its economy, any operational, architectural or functional failure resulting from lack of interoperability will be much costlier. Interoperability is also crucial for a deployment of the smart grid that is open to all manufactures, vendors and integrators, so that the grid operators can concentrate on top-level functions, independently of proprietary solutions.

3 An open toolkit for Energy Management Systems

Nowadays, the demanding environmental challenges call for greener and more efficient power systems. Concurrently, we are facing a change of paradigm in power systems as we move from the previously centralized design towards a decentralised architecture where boundaries between consumers and producers become diffuse.

Integrating distributed energy resources in buildings and houses is a way to move towards this direction as they produce cleaner energy while at the same time reducing energy losses due to electricity transport. Moreover, the expansion of the electric vehicle as an active part of smart grids and buildings is a reality that is not only a major step into a more sustainable mobility but a game changing actor in smart buildings [33]. On the other hand, the continuous development of ICT technologies is swiftly reaching the daily devices we use in houses and buildings, bringing the so called “smart appliances” and a wealth of inexpensive devices based on modern communication protocols [32], like Zigbee [7], to the general public.

These devices achieve an improvement in user’s comfort level, as well as provide the possibility to automatically respond to changes in electricity prices, weather conditions, or various custom triggers defined by the user. The combination of the ability to produce their own energy and the responsiveness/flexibility to exogenous stimuli can transform the users from classic consumers to modern prosumers. On this increasingly complex context, some sort of intelligent system is required to coordinate and exploit the flexibility of these smart appliances and at the same time to deal with the intrinsic uncertainty of most renewable sources.

As until recently electric energy could not be easily stored, electric power companies matched demand and supply in a regional or national level by either increasing or decreasing production rate of their power plants or importing power from other electric utilities. Since there are limits on what can be accomplished on the supply side, bearing in mind that generating units can take a long time to produce full power and demand can at times exceed the available supply, demand response aims to better match the demand for power in a more efficient way and in different levels.

Electric utilities can send demand requests to customers in a variety of ways, like off-peak metering where power is cheaper at certain times of the day and smart metering where explicit requests or price changes can be directly communicated to customers. From the customers’ point of view, postponing electric power consuming tasks to adjust power demand, paying a higher price for electricity or switching part of their consumption to alternative sources like photovoltaic panels and batteries are some possible reactions to an electric utility demand request.

The energy management system (EMS) of the SGIL monitors the power consumption of the lab appliances like the white goods (fridge, washing machine, clothes dryer, oven, coffee machine, dishwasher etc.), the water heater, the load emulator, the EV charger and the ventilation system and up to some degree can also control their operation.

The demand response scenario is defined as a period in which the customer demand is controlled in a way that supports the power system. We assume that the external signal sent by the electric utility is received by the EMS in a form of a demand power request in kW and time duration in minutes/hours. The EMS algorithm will have to keep the total laboratory power consumption below the requested level during the specified time duration.

The SGIL, which in our experiments serves the purpose of a model smart house, is equipped with three type of devices. There are loads that consume energy (white goods, ventilation etc.), there are units that produce energy (PV panels) and finally there are devices that store excess energy produced (batteries).

In order to better prepare smart houses of the future to cope with demand response requests as well as being more financially viable and energy autonomous, forecasting techniques should be applied in the areas of power generation and consumption. By knowing in advance the PV generation and power consumption for the next day, the EMS could better respond to demand response requests by performing a variety of actions in different scenarios:

- storing renewable/cheap energy in the batteries if next day PV generation is forecast to be low,
- consuming battery energy if next day PV generation is forecast to be high, thus creating battery capacity for future energy storage,
- storing energy in the batteries to be used during periods when electricity is costlier,
- storing excess energy to battery, if fines are imposed by electrical utilities when exporting electricity back to the grid and

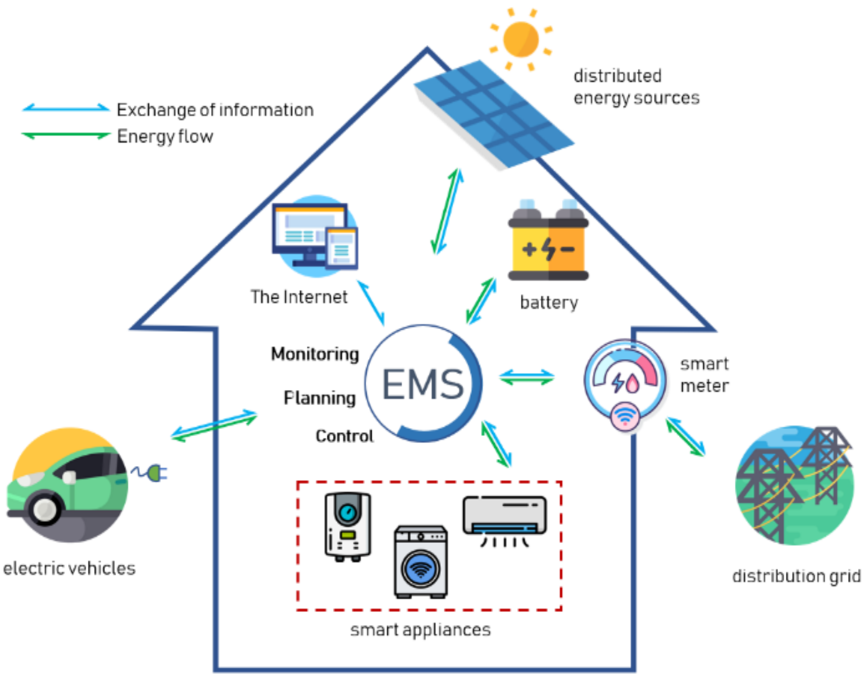
- storing energy to be sold back to the grid during periods when electricity demand is higher, thus creating a financially profitable smart house operation environment.

3.1 Energy Management System definition

In the context of Smart Houses and Buildings, an Energy Management System is a computer-aided program that make decisions or recommendations on how to operate different devices that influence the energy cost and consumption of a building, based on information about the building, the environment and the user preferences [4]. A literature review on the topic can be seen in [30].

Example of devices that can be controlled are the heating, ventilation and air conditioning (HVAC) units, the state of charge (SOC) of the battery or the schedule of smart appliances. The capability of control devices (loads) and batteries (storage systems) to follow external signals such as the electricity price [8][23], grants flexibility to the operation of smart buildings. Therefore, smart houses equipped with Energy Management Systems can take part in auxiliary markets providing demand response (DR) service [13][31]. This also allows better integration of renewable energy [1] in the buildings. The main focus on the decision adopted by the EMS, depends on what the user intends to achieve namely, minimizing total energy cost, maximizing self-consumption of the energy produced or minimizing total energy consumption, among others.

Fig. 5. Diagram of the energy elements of a Smart House



To achieve these goals, the EMS uses historical records, future prognosis of weather conditions and user preferences in the decision-making process. Furthermore, the EMS must be equipped with a more or less detailed model of the architecture and elements of the building. Based on all this information, the goal is to try to forecast the building behaviour and to plan the smart appliances and active elements operation accordingly. The detailed features and tasks of an EMS may vary, but a full EMS will normally be able to monitor, control and plan the devices. Home Automation programs belong to a similar interesting EMS category that focus on sensor monitoring and device operation and allow action triggering after an event occurs. Although being software-powered as well, they lack the holistic view of EMS and the mathematical tools to schedule and operate devices based on expected future events.

3.2 Introduction to PyEMS [20]

PyEMS is an in-house developed Python package designed to help an EMS implementation for a building or house tailored to user needs. The scope of the package is to extend the capabilities of existent Home Automation programs with the incorporation of mathematical and data pre-processing tools required for smart energy planning.

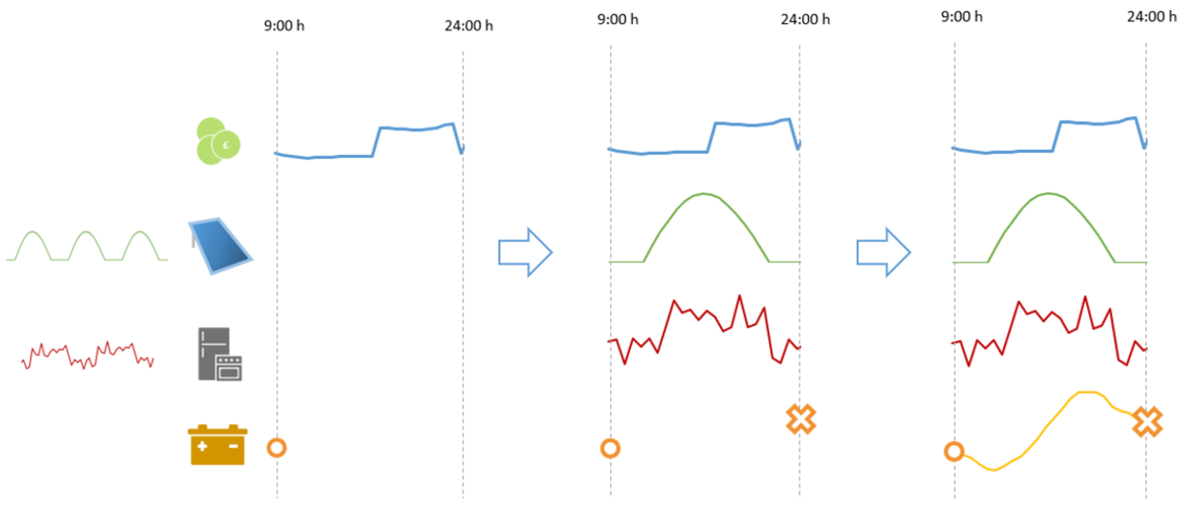
Although the HA software is able to send instructions to the devices part of a smart house, in most cases the user defines the rules and automations that are used to issue the instructions. However, setting this general logic requires expert knowledge that not all users hold. Furthermore, the number of possible device states together with the uncertainty of future events (e.g. weather, energy consumption) makes it that even an expert user can't efficiently define these rules. PyEMS is designed to help in such task providing data-driven optimization computations to determine the optimal state of the devices, using historical and forecast information. Based on the planned target configuration and the current state, an efficient set of commands is determined by the HA software for the house control. In the following case, we exemplify the general idea of implementing an EMS with the use of PyEMS. As such, we have considered a smart building equipped with a photovoltaic installation, some devices (loads) and a storage battery.

The house is equipped with a local machine (hub, single board computer etc.) that runs a Home Automation (HA) software (e.g. Home Assistant [14] or OpenHAB [15]) and a database that stores historical measurements of power consumption, power generation, battery state of charge etc. The automation software is in charge of integrating devices, collecting measurements and events, and sending instructions to operate the smart appliances based on automation rules. The measurements collected by the different sensors and meters are stored in the database.

In addition to this, there are many online sources for meteorological data that can be accessed to retrieve the weather forecast (e.g. temperature, solar irradiance) for the following days. The HA software can retrieve and store this information to be used by the energy management algorithm. The same rule applies when dealing with data related to electricity pricing (tariffs or dynamic).

Considering this scenario, an EMS can determine the optimal target state of charge (SOC) of the battery using the weather and consumption prognosis. The resulting commands to charge or discharge the battery are obtained after comparing the target SOC with its current SOC. The main goal is to accommodate as much renewable energy as possible, making use of it when the electricity is costlier. However, the user can choose to emphasise different algorithm strategies such as cost reduction, equipment life span preservation or increasing the share of renewable energy used. The computations are periodically updated with the new measurements and forecasts, at user-defined regular intervals - called time steps.

Fig. 6. Input and output information in an EMS execution for the planning of the SOC of a battery



The following paragraph will guide through the steps followed to determine the state of charge of the battery. The first important consideration is to determine how many periods ahead we want to plan, in other words, we have to decide the time horizon. In most cases, this decision is limited by third-party data availability (such as limited information on electricity prices). We assume that, electricity prices for the next day are published late in the evening and we only know the prices until next midnight. As an example, consider running the energy plan algorithm every hour and executing it just before e.g. 9.00 AM. Therefore, actions can be only planned until the end of the day. Under this scenario, the time horizon has different lengths at every hour.

Following this example, when the algorithm of our EMS is called, it starts by gathering all available data: historical data prior to 9.00 stored in the database, prices from 9.00 to 24.00 obtained through an Application

Programming Interface (API) and a weather forecast for the same period of time, obtained from a meteorological online source through another API. PyEMS automatically creates the queries for the different APIs and databases based on the execution time and time horizon in order to collect the required data.

The next step is to issue a forecast for the load of the building and the solar generation (there are many forecasting tools that can be used). In this sense, PyEMS also provides an abstraction layer in charge of formatting the data for the specific tool. In the SGIL, the open source Facebook Prophet [11] is used as a forecasting package combined with Ax [3], an adaptive experimentation platform. PyEMS also includes a routine to determine an adequate SOC at the end of the simulation horizon based on medium-term weather prognosis.

Finally, EMS uses mathematical programming techniques to determine the optimal path to be followed by the battery SOC under that foreseen landscape. In this regard, PyEMS automatically translates the system definition (a building with loads, PV panels and a battery) into an optimization problem modelled with Pyomo (Python-based optimization modelling language) [15] and triggers the process for its solution by GLPK [12], an open Mixed Integer Linear Programming (MILP) optimization problem solver. Based on that information, the HA can accomplish the necessary actions to reach the target SOC for the next step.

4 PyEMS insight

The toolkit intends to help in the development and automation of an energy management system for buildings and smart houses. The package, written in Python, provides the mathematical and data pre-processing tools to leverage existent Home Automation software into fully-fledged EMS. It includes methods to gather historical and auxiliary information and issue deterministic forecasts. Based on those forecasts and the system description, an optimization model is generated to help determine the optimal values of the decision variables. Examples of decision variables are the optimal state of charge of the battery or the schedule for a flexible smart appliance.

For instance, the EMS can compute results for every hour, from a certain hour in the morning until midnight. On the other hand, the toolkit is also a valuable tool for researchers who seek to investigate the possible benefits of different energy management strategies.

4.1 Main features

The main features of the package are:

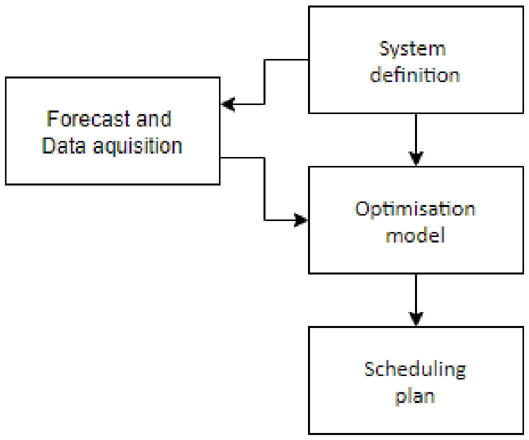
- Modular construction: the toolkit is not a final product but instead a toolkit that provides different entities (classes) and functions that can be combined together to create a custom EMS.
- I/O tools allowing a custom data flow while easing the input/output of data from/to different data sources like .csv files, databases, APIs etc.
- Integration of forecasting tools (e.g. Facebook Prophet [11] or scikit-learn [25]).
- Optimization - the planning is obtained by solving an optimization model. PyEMS solves a MILP problem modelled with Pyomo and solved with the GLPK open source solver.

The toolkit is implemented following an Object-oriented programming (OOP) approach.

4.2 Modes of operation

The simplified scheme of operation could be illustrated as follows:

Fig. 7. Summary steps of an EMS execution



This toolkit provides two well-differentiated modes of operation:

- Simulation - Reproduce the operation of the EMS based on a data set. This mode is useful for research purposes as well as to tune the parameters of the EMS and make economic evaluations. This can be ran without any third-party software, apart from the required python packages and optimization solver.
- Operation - Deployment of an EMS that controls a system. The system automatically collects data from different sources (databases, APIs etc.), runs the optimization model and returns the results. This mode requires the use of third-party tools to work.

4.3 Time in PyEMS

Managing time is a critical issue when working with time series - avoiding mismatches (e.g. when one API provides data in UTC and another one in local time) and ensuring correct calculations. The toolkit follows the good practice of internally performing operations in UTC time and only converting the timestamps to local time when producing user output, except when for example computing the local midnight time. For the EMS to work properly, the time zone must be specified to make the appropriate conversion of inputs and outputs from local time to UTC and vice versa.

Another important decision to be made, is defining the time horizon, i.e. the time interval between the start and end of the simulation. The toolkit provides two predefined behaviours, either a fixed window or a midnight horizon. The first approach consists of constantly planning for a fixed number of hours ahead irrespectively of the current time, whereas the second one always plans until the following midnight. Depending on data availability, i.e. due to the electricity price publication scheme, one approach could be more appropriate/beneficial than the other.

Due to the discrete and finite nature of digital simulation, a time step must be specified to determine the length of the periods of the simulation horizon and the way the historical and auxiliary data are grouped. Currently, the time step supported by the toolkit can vary from 1 minute to 1 hour. The lower limit is justified by scarce availability of data in a lower than one-minute resolution and the likelihood that the time invested in solving the simulation would be higher than this span. The upper limit is due to the resolution of electricity price and weather auxiliary data usually being equal to 1 hour. If a longer period is considered, two energy prices could be applied to the same time step, which is not supported by the toolkit at this stage. For the same reason, the time step must be the result of splitting one hour into equal parts – in this case an overlap between different periods and prices is not possible.

4.4 Units of physical magnitudes

Nowadays, most appliances and devices include electronic components. The normal operation of devices includes fast and substantial changes in their power consumption, sometimes exhibiting random consumption patterns. Since representing these fast changes would require a very small time step, the algorithm computes and returns the result of calculations in energy units. Furthermore, the equations expressed in terms of energy are more easily extendable between different physical fields.

Currently, available entities in the package describe electrical components only, thus the unit of measurement used is kWh. Nevertheless, in future releases, components from other units of measurement could be included. The convention adopted is that all energy inputs to the system, like electricity from the grid or PV generation, are positive quantities.

4.5 Main abstract entities

The main entities (classes) of the core package are:

- Simulation: controls the execution process and stores general and temporal parameters of the execution (like the number of time periods ahead to optimize or the time step resolution).
- System: represents a physical system like a house, an office building or a power system. The system is a container for other system components like electrical/thermal loads or generators.
- System Components: blocks to build the system. At the moment, the development is focused on electrical components. The intention is to include thermal components in the future.
- Forecaster: these entities act as interfaces that format the data collected by the PyEMS and third-party forecasting tools (e.g. Facebook Prophet and Facebook Ax). The Forecaster arranges the data into the specific format required by the tool and extracts the resulting forecast.
- Data handler: this class is in charge of the data input and output flow. It provides, on request, data to different components of the system.

- Optimizer: Translates the system definition into a mathematical optimization model (Pyomo model). After this, the Optimizer triggers the solution of the optimization problem, checks the validity and returns a Results object.

4.5.1 Simulation

The Simulation object controls the flow of execution, computes time variables, including the start and end of simulation and triggers actions coming from high-level components like the System and the Optimizer. At the moment, the simulation has two operational modes:

- Simulation: Reproduction of the EMS operation based on a data set. This mode is useful for research purposes, as well as for tuning the EMS parameters and making cost-effective evaluations. This can be run without any third-party software, apart from the required python packages and optimization solver.
- Real operation: Deployment of an EMS that actually controls a system. The system automatically collects data from different sources (databases, APIs, .csv files, etc.), runs an optimization model and returns the results. This mode requires the use of third-party tools to work.

The workflow in both modes can be summarized as follows:

- In real operation:
    - Retrieve current time and compute the start, end and number of periods of the simulation horizon.
    - Prepare the System (data gathering, formatting, processing - from local and online sources).
    - Execute the optimization model.
    - Process the results.
    - Return results to the HA software.
- In a rolling window simulation:
    - Iterate in a loop through the rolling window horizon.
    - Compute the start, end and number of periods based on simulation time.
    - Prepare the system (load and format all data required for the stored dataset).
    - Execute the optimization model.
    - Process and store the results.
    - Clean the system and optimizer.
    - Update current simulation time and start a new iteration loop.

This entity will arrange and direct the simulation, according to the time step and simulation horizon defined by the user. At this stage, the time step must be set between 1 min and 1 hour (both included). More details regarding the time step can be found in section 4.3.

Depending on the mode of operation, the Simulation gets the time from the local machine where it is being executed (real operation) or performs a rolling window to roll over a defined data set starting at the time set by the user (simulation mode). The start of the simulation horizon is calculated so that it can only occur on multiples of the time step, counting from the start of the hour. In this way, we ensure that no period prolongs more than between two hours with different prices.

Currently, the Simulation object provides two strategies to determine the end of the simulation. The simplest approach consists of setting a simulation horizon with a fixed number of periods. In this case, once the starting point is known, the end-point is calculated by just adding the desired number of periods to the start. The second approach consists of setting midnight to be the end of the simulation regardless of the current time. That leads to a varying simulation horizon length and is meant to address prices publication schemes, like the dynamic Spanish household prices.

4.5.2 System

The System represents an abstraction of a physical entity like a dwelling or an office building. This entity functions as a storage of other physical components that have an impact on energy consumption like appliances, PV panels or HVAC systems, which are themselves modelled as loads, and generators. The System is empty by default and the desired elements can be added programmatically. The System together with its attached Components, describes the building or a part of it. This definition is used by the Optimizer object to elaborate a custom optimization model. The next steps in the execution process are:

- Check system composition to ensure that some source of energy is provided, as well as algorithm limitations.
- Compute the total fixed electrical load. That will trigger the computation of the forecasts for the components requiring it, including uncertain and non-flexible loads.
- Compute the total stochastic electrical generation that will trigger the issue of forecasts of non-dispatchable generators like PV panels.
- Obtain battery initial and final state of charge. If a storage system is installed, it gets the initial battery SOC as the last value stored in the database. It also estimates a target SOC at the end of the simulation horizon according to mid-term (days) weather prognosis.
- Retrieve purchase and sale electricity prices: loads in the system the prices obtained through the toolkit’s Data Handler functions. These prices can also be estimated, in case they are not fixed or available in advance.

4.5.3 System Components

System components are any kind of device to be modelled in the calculation, be it a physical element like an appliance or a battery or instead an abstract entity like a group of loads. The system components should be declared programmatically as an attribute (internal variable) of the System instance.

We can classify system components by physical fields like electrical or thermal. At the moment, the toolkit is focused on electrical components, including Electrical Generators, Electrical Loads, Electrical Batteries or Electrical Generators with different subclasses.

Some of these components are not controllable or the EMS is simply not informed by the user when they are going to be used. This is the case with PV panels, a TV or the hold building load. The behaviour, then, becomes uncertain, and must be somehow foreseen. For this purpose, uncertain system components may include a Forecaster object to anticipate its behaviour.

4.5.4 Forecaster

Planning actions to anticipate future events requires an accurate estimation of the uncertainties of the system – e.g. PV production or the total energy consumption. PyEMS implements the Forecaster entity as an abstraction layer that interacts with third-party Machine Learning tools (e.g. Facebook Prophet [9] or scikit-learn [32]) to deliver precise forecasts. There are different Forecaster class implementations for different forecasting tools. The function of this entity is to arrange the data gathered by other components in the specific format understood by the tool. The Pandas toolkit [28] was used for the data processing. Then, retrieves the resulting forecast and load it into the linked component. A previous data analysis is required to determine the best parameters of the forecasting tool. This analysis was performed with Facebook Ax [10]. More details of the forecasting process are provided in section 4.6.

4.5.5 Data handler

Interoperability is a known issue among IoT devices. Every Smart Building is likely to have different components and use different technologies to monitor and store data. Therefore, a technology dependant implementation to retrieve data is very likely to be restrictive and inflexible. The Data Handler is a class meant to overcome these difficulties acting, as an intermediary between the EMS and data sources. That way the user is granted with total flexibility on how to retrieve this information and the source. Furthermore, the toolkit provides functions to ease the process of retrieving data from some popular sources.

The user should create firstly its own custom Data Handler that inherits the attributes and methods of the base class and their own functionality to retrieve a particular data series or data point in a standardised way. Next, the user must label the function and add it to the list of sources.

4.5.6 Optimizer

The Optimizer translates a System object populated with different System Components into a mathematical optimization model. The next step involves triggering the model solution by one out-of-the-shelf Solvers. The results are extracted and translated into a Results object.

The Optimization model is written in Pyomo, a Python-based, open-source optimization modelling language. Depending on the system components, different Pyomo blocks and expressions can be added and filled with data stored in each System Component. The default Solver used by the Optimizer is GLPK, that stands for GNU Linear Programming Kit, but thanks to the Pyomo abstraction layer, other solvers can be used without modifying the code.

The main steps followed by the Optimizer are:

- Creating an optimization model.
- Solving the optimization model.
- Extracting results from the model.
- Checking results.
- Returning the Results object.

4.6 Forecasting PV generation and building load

Forecasting uncertain magnitudes is an essential task to anticipate future events. The Forecaster entity, described in section 4.5.4, provides an interface to connect PyEMS with open-source tools. However, achieving state-of-the-art forecasts requires to carry-out a previous human-supervised analysis to tune the parameters of the forecasting tools. The required steps to obtain the forecast and the previous data analysis are described in detail in this section.

4.6.1 Introduction to the problem

Forecasting models are developed for both PV generation and laboratory energy consumption, being crucial parts of the EMS plan to adjust the smart house energy flow.

Regarding forecasting models, three types can be distinguished: black, grey and white box models. Their differences lie in the knowledge that the model has on the physical component.

- The white box model is based on physical equations that are known with certainty. As an example, in a house model, where thermodynamic equations are used to study the flow of heat, and information about the materials is available, the thermodynamic parameters can be easily computed.
- In the grey box model, parameters are not known in advance but can be estimated through temperature data analysis and weather conditions measurements.
- Black box models are general purpose mathematical algorithms having no knowledge about the system. They are fed with historical and auxiliary data in order to build a valid forecast for a specific variable. In fact, time series analysis is the basis for many black box models.

Even in white models case, a dependence on uncertain future environmental parameters is likely to exist, e.g. the indoor smart house temperature will depend, amongst others, on outdoor temperature, wind speed and solar irradiance. Because black box models are based on mathematical and machine learning techniques, they are more flexible and can yield better results when system parameters values are not known in advance. In that context, the forecasting tool developed belongs to the latter model family.

The forecasting code was developed in the Python programming language including (among other libraries) the use of the Pandas toolkit [18] for the data processing - an open source library for data manipulation and analysis offering data structures and operations for time series analysis and handling of numerical arrays. Moreover, two state-of-the-art tools were integrated to aid forecasting analysis, i.e. the Facebook Ax [3] and Facebook Prophet [11] software packages.

When confronted with problems involving a large number of possible ways to configure certain parameters or specific machine learning hyper parameters, developers and researchers spend time and resources to select, adjust and fine tune those configurations. Ax is an adaptive experimentation platform based on a machine learning system designed to automate this configuration process and output the best possible parameter values and combinations. It can optimize any kind of test, including machine learning experiments using multi-armed bandit optimization and integer or floating point configurations with the aid of Bayesian optimisation. The latter optimisations were implementing with the use of the built-in BoTorch model provided by Ax, which is a customisable reasonable default for modelling and optimisation.

On the other hand, Prophet is a procedure for forecasting time series data, based on an additive regression model including highly advanced forecasting methods like:

- changepoint analysis,
- model fitting with different seasonalities (daily, weekly, yearly or user preference based),
- seasonal components modelled on the basis of a fourier series,
- seasonal components incorporating user input variables, and
- linear or logistic growth curve trends, where Prophet automatically detects trend changes by selecting changepoints from the available data set. In the case of linear growth curve, data trend will keep on growing with time, whereas in the logistic case data values saturate at some point.

For better results, a large pool of historical data as well as some kind of strong seasonal behaviour of variables to be forecast are needed. Moreover, what makes Prophet suitable for our experiments is its tolerance in missing data (e.g. due to laboratory sensor malfunction or communication disconnection between sensors and the EMS) as well as efficient handling of data outliers. Prophet includes many possibilities for user adjustable and tweaked forecasts with the introduction of domain knowledge, like experiment regressors influencing the forecasted variables.

4.6.2 PV generation forecasting

As mentioned in chapter 3, by knowing in advance the PV generation and power consumption for the next days, the EMS could better respond to demand response requests by the electric utility, thus favoring the application of forecasting techniques in those two areas.

The first series of experiments include the forecasting of PV generation ideally for the next day, in a way that the EMS can adjust the energy plan accordingly. The novel methodology implemented consists of several steps, which will be explained in detail. At the end, all steps will be incorporated into an automated tool that can run without any user intervention and provide timely forecasts. The first step consists of constructing past and future data frames necessary as inputs to the Ax and Prophet forecasting tools.

Construction of past and future data frames

a) In order to forecast PV generation, historical data about sunshine and sunshine power are retrieved from the laboratory's database, where sunshine power represents sun intensity measured per square meter and sunshine the expected sunshine percentage at a given time. These data have been taken directly from the Dutch weather service Buienradar [5], as a daily weather report for a nearby station close to JRC Petten site (52.77N, 4.66E) using the provided API. In Fig. 8, there is a screenshot of the JSON formatted weather forecast data taken from Buienradar. Those variables will act as the regressors in the forecasting procedure, since solar irradiance directly influences PV generation. That cannot exclude the inclusion of other future regressors such as synthetic forecasts, other weather sources or other variables which might influence the forecast.

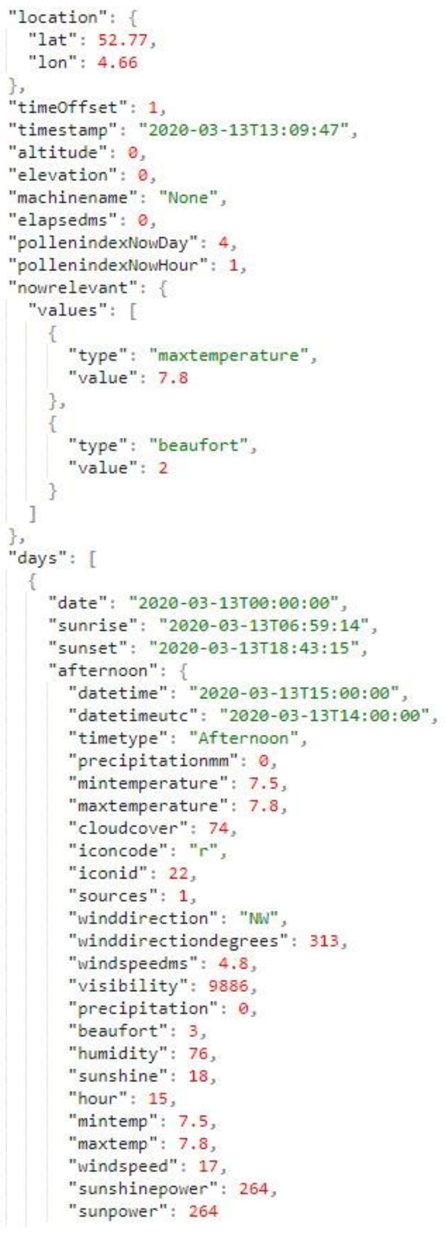
b) Hourly historical power data on PV generation can be retrieved from the lab's database. The data should be grouped and resampled (hourly granularity) by taking the integral of them in time space, the result being the total energy generated every hour.

c) By concatenating historical data about sunshine, sunshine power and energy generation, a past data array can be created, which will serve as an input for the forecasting tools.

d) The future interval of the regressors is retrieved from the same Buienradar web page used during step (a). Since the source weather forecast is provided in JSON format, some kind of data manipulation is needed to extract the required data.

e) The last step consists of constructing the future data frame, which includes information about the forecast regressor value and date stamp information.

Fig. 8. Sample JSON formatted Buienradar weather forecast data



```
"location": {
  "lat": 52.77,
  "lon": 4.66
},
"timeOffset": 1,
"timestamp": "2020-03-13T13:09:47",
"altitude": 0,
"elevation": 0,
"machinename": "None",
"elapsedms": 0,
"pollenindexNowDay": 4,
"pollenindexNowHour": 1,
"nowrelevant": {
  "values": [
    {
      "type": "maxtemperature",
      "value": 7.8
    },
    {
      "type": "beaufort",
      "value": 2
    }
  ]
},
"days": [
  {
    "date": "2020-03-13T00:00:00",
    "sunrise": "2020-03-13T06:59:14",
    "sunset": "2020-03-13T18:43:15",
    "afternoon": {
      "datetime": "2020-03-13T15:00:00",
      "datetimeutc": "2020-03-13T14:00:00",
      "timetype": "Afternoon",
      "precipitationmm": 0,
      "mintemperature": 7.5,
      "maxtemperature": 7.8,
      "cloudcover": 74,
      "iconcode": "r",
      "iconid": 22,
      "sources": 1,
      "winddirection": "NW",
      "winddirectiondegrees": 313,
      "windspeedms": 4.8,
      "visibility": 9886,
      "precipitation": 0,
      "beaufort": 3,
      "humidity": 76,
      "sunshine": 18,
      "hour": 15,
      "mintemp": 7.5,
      "maxtemp": 7.8,
      "windspeed": 17,
      "sunshinepower": 264,
      "sunpower": 264
```

Running the Facebook Ax tool

The next step involves running multiple iterative Prophet forecasts within the Ax toolkit in order to calculate the best possible parameter values. To help Ax provide more reliable and timely results, lower and upper bounds should be defined for every parameter based on the experiments' characteristics. For PV generation forecasting, these parameters include:

- initial period being the historical data time duration. Empirical evidence has shown that at least a month of historical data should be fed to Ax to give reliable results.
- fourier order, as seasonalities are estimated using a partial fourier sum. The number of terms in the partial sum, that is the fourier order, is a parameter determining the seasonality rate change. When introducing a larger value for the fourier order, more components are introduced in the trend leading to either picking up noise or/and obtaining more accurate values.
- two available seasonality modes, additive and multiplicative. In the case of additive seasonalities, the seasonality effect should be constant over the entire period, whereas in the multiplicative case, the importance of seasonalities increases over time.
- changepoint prior scale argument, where increasing it will make the trend more flexible. Allowing too much flexibility results into trend changes overfitting.
- sunshine regressor prior scale and sunshine power regressor prior scale arguments adjusting the overfitting of each regressor. By default, these arguments are equal to 10, so reducing them dampens the effect of each regressor.
- number of trials of the Ax tool which should be no less than 20, as recommended by the developers.
- cap and floor values for the past and future data frame variable to be forecast. The reasoning behind applying these limits, lie in the fact that PV panels have a certain PV generation capacity.

After setting the ranges for the above mentioned parameters, we set to run the Ax optimization loop using the Sobol sequence [28] with a specified number of trials. Each trial corresponds to a set of parameters and results in a Prophet forecast being issued using these parameters. These threads have been programmed to run in parallel for maximum processor utilization, speed and performance. There is no upper limit to the number of Sobol trials, having a larger pool improves the final result at the cost of a longer processing time.

Using the results from all the Sobol trials, Ax enters an iterative optimisation mode calculating the best possible parameter values by adapting them in every forecast trial using machine learning techniques like Bayesian and bandit optimizations. The objective function of Ax is to minimize MAE (Mean Absolute Error) between the forecast and the real data. Figure 9 present the process as a diagram.

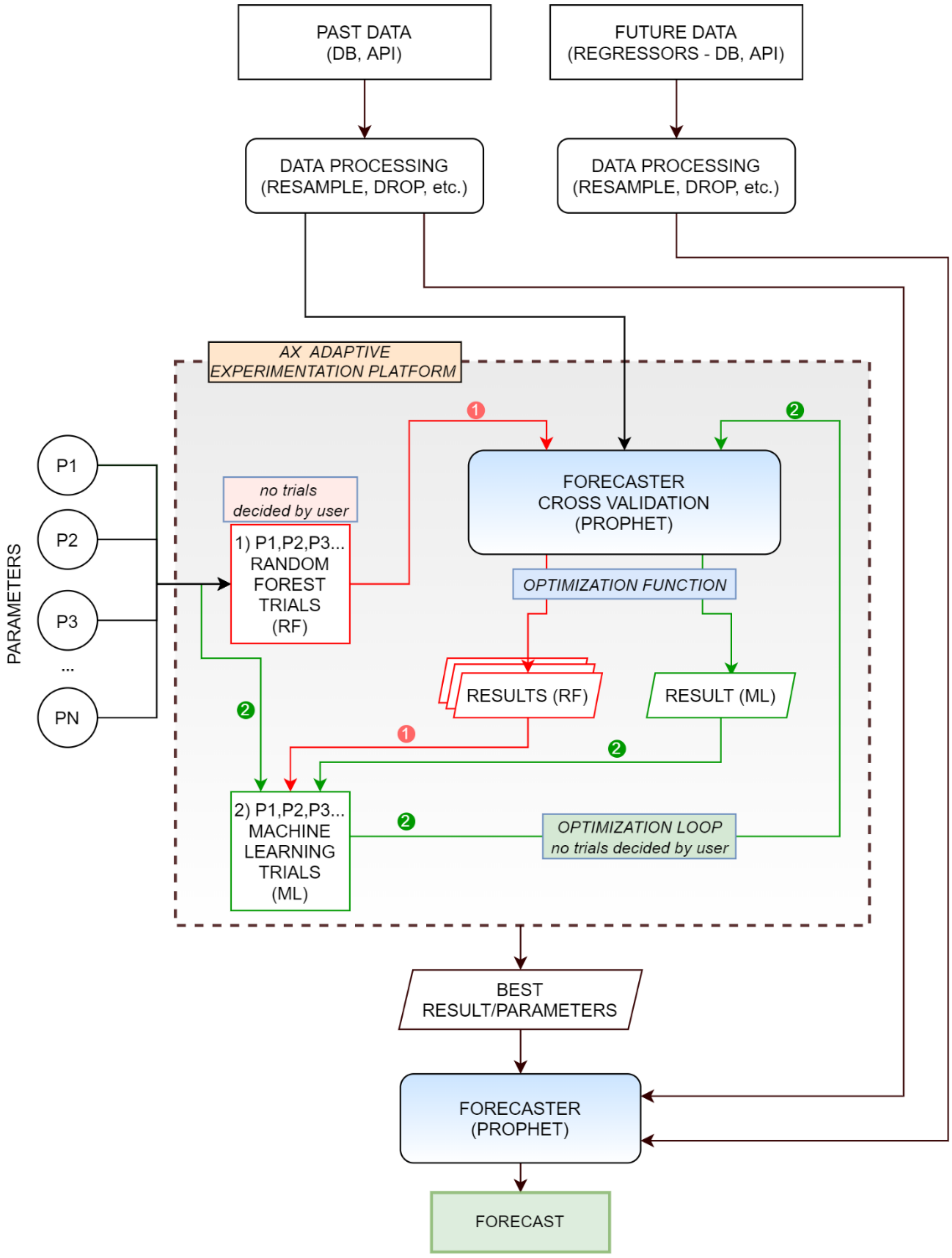
The Ax tool output consists of a list of the best possible parameter values, being the initial period, the fourier order, the seasonality mode, the changepoint prior scale, the sunshine regressor prior scale and the sunshine power regressor prior scale, determined by the RMSE (Root-Mean-Square Error) and MAE (Mean Absolute Error) values. As each error influences MAE in direct proportion to the absolute value of the error in contrast to RMSE, we will use MAE as an important characteristic of the forecasting model performance. Lower MAE values equal better performance, from a statistical point of view.

In setting up the experiments, the Ax tool was provided with the following parameter definitions and bounds:

- initial training period for laboratory PV generation historical data is set to 60 days.
- Buienradar forecast data period set to 2 days, which can be increased in case of a longer forecast needed.
- fourier order ranges between 5 and 40 in order to avoid overfitting the trend.
- changepoint prior scale, sunshine regressor prior scale and sunshine power regressor prior scale arguments can fluctuate between 0.01 and 40.
- the number of trials of the Ax tool is set to 30 or in another number in case a specific MAE/RMSE VALUE is targeted.

- the PV generation cap and floor are set to 80kWh and 0kWh, respectively.
- weekly seasonality is set to false because PV generation does not vary with the days of the week.

Fig. 9. Ax-Prophet process diagram


PAST DATA (DB, API)
FUTURE DATA (REGRESSORS - DB, API)
DATA PROCESSING (RESAMPLE, DROP, etc.)
DATA PROCESSING (RESAMPLE, DROP, etc.)
AX ADAPTIVE EXPERIMENTATION PLATFORM
PARAMETERS
P1
P2
P3
...
PN
no trials decided by user
1) P1,P2,P3... RANDOM FOREST TRIALS (RF)
FORECASTER CROSS VALIDATION (PROPHET)
OPTIMIZATION FUNCTION
RESULTS (RF)
RESULT (ML)
2) P1,P2,P3... MACHINE LEARNING TRIALS (ML)
OPTIMIZATION LOOP no trials decided by user
BEST RESULT/PARAMETERS
FORECASTER (PROPHET)
FORECAST

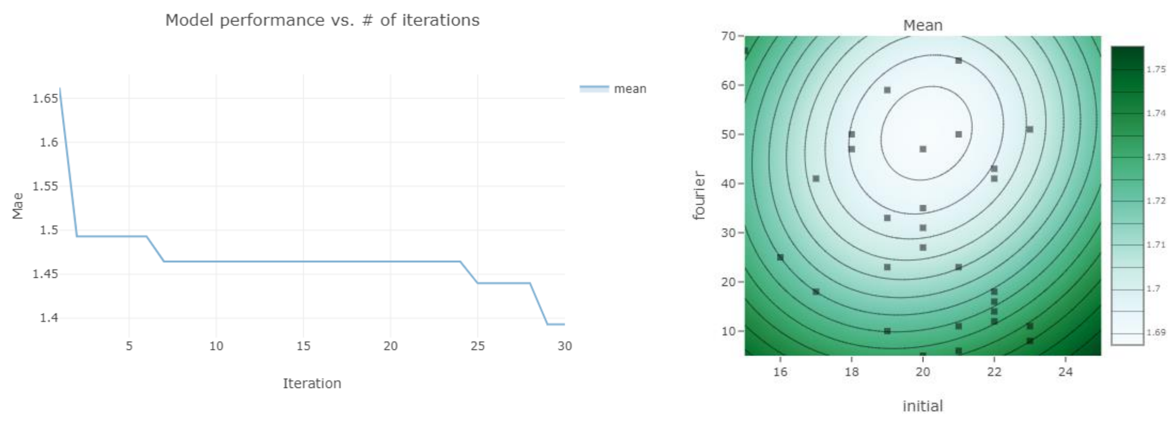
Running the Ax tool provided us with the following results as depicted in Table 1.

Table 1. Ax provided best parameter values

Parameter	**Value**
Initial period	20
Fourier order	5
Changepoint prior scale	2.21
Sunshine regressor prior scale	24.5
Sunshine power regressor prior scale	28.6
Sunshine regressor mode	additive
Sunshine power regressor mode	multiplicative
RMSE	1.87
MAE	1.39

Ax also provides the user with interesting graphs and contour plots like MAE vs number of Ax iterations and initial period vs fourier order (Fig. 10).

Fig. 10. Ax parameter graphs and contour plots



Running the Facebook Prophet tool

The last step of the forecasting methodology involves running Prophet using the best parameters obtained with the Ax tool. The preparation includes gathering the best parameters calculated by Ax to issue a forecast for the next days.

To measure forecast error using historical data, Prophet includes time series cross validation functionality. This is achieved by selecting cut-off points in the historical data and for each of the points fitting the model, use data only up to that point, with the intent of comparing the forecast values with the real data. The forecast horizon (horizon) and optionally the initial period of historical data (initial) and the spacing between cut-off dates (period) should be defined as cross validation arguments in Prophet.

Figures 11-12 show the forecast prediction graphs generated with the Prophet and Ax combination for two consecutive months (March and April, transition to spring season). The red line illustrates the real measurements, whereas the green line is the forecast for the solar generation. It is obvious that the forecast model works well as it follows the general trend of the real data while also showing a low MAE. Some errors are present either due to the weight brought by the past data (e.g. 18-21 March) or by an unreliable weather forecast (e.g. 24 March), however the algorithm corrects the error over the following days. The long term seasonality is captured by the forecast.

The forecast was issued on an hourly basis and stored in the database of the laboratory.

Figure 13 shows the daily mean absolute error (MAE) for the forecasted months. The total MAE for March is 3.8 and for April is 2.5 31 (the PV panels generated an average of 300kWh/day in March and 510kWh/day in April).

Fig. 11. Forecast prediction graphs for PV generation (March and April 2020)

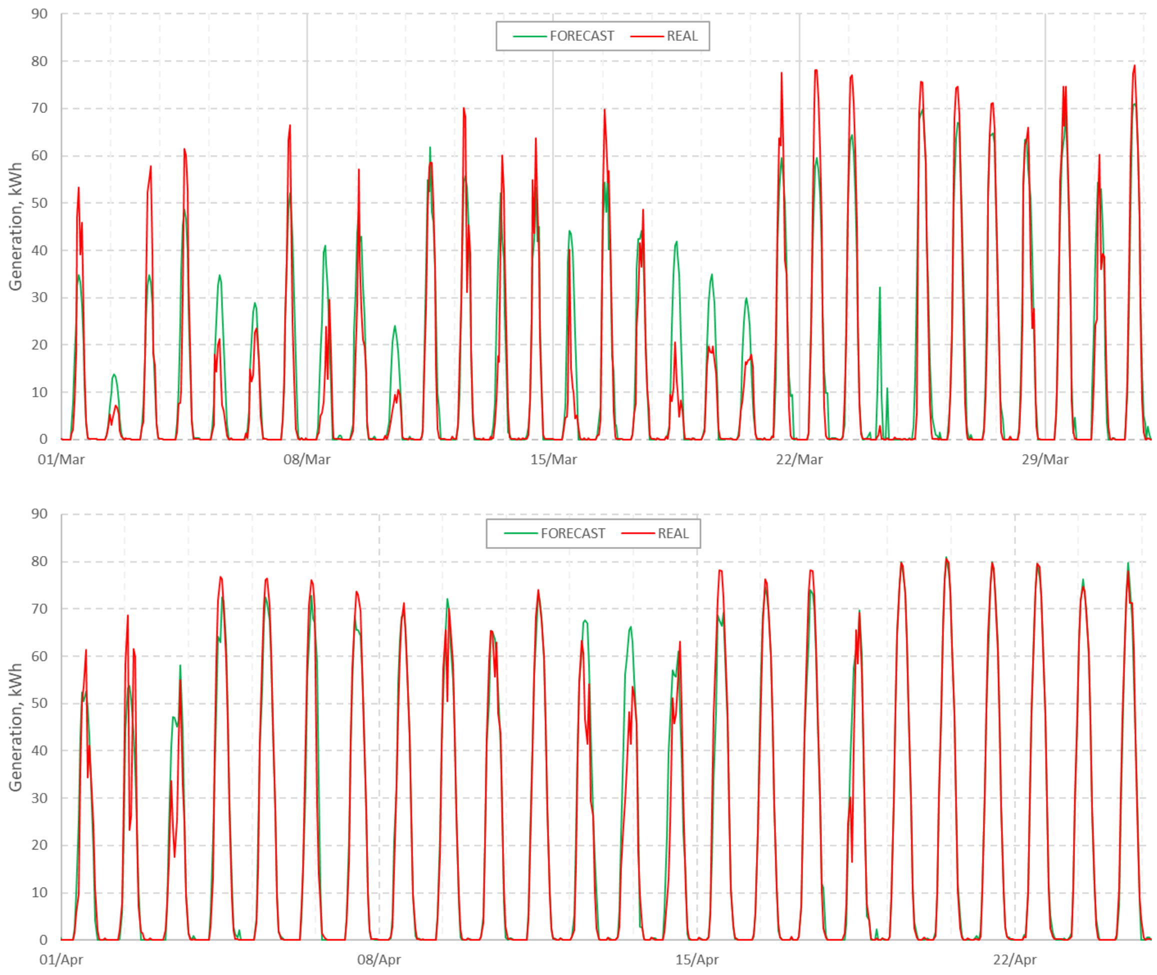


Fig. 12. Forecast prediction graphs for PV generation (detail 08-15 April)

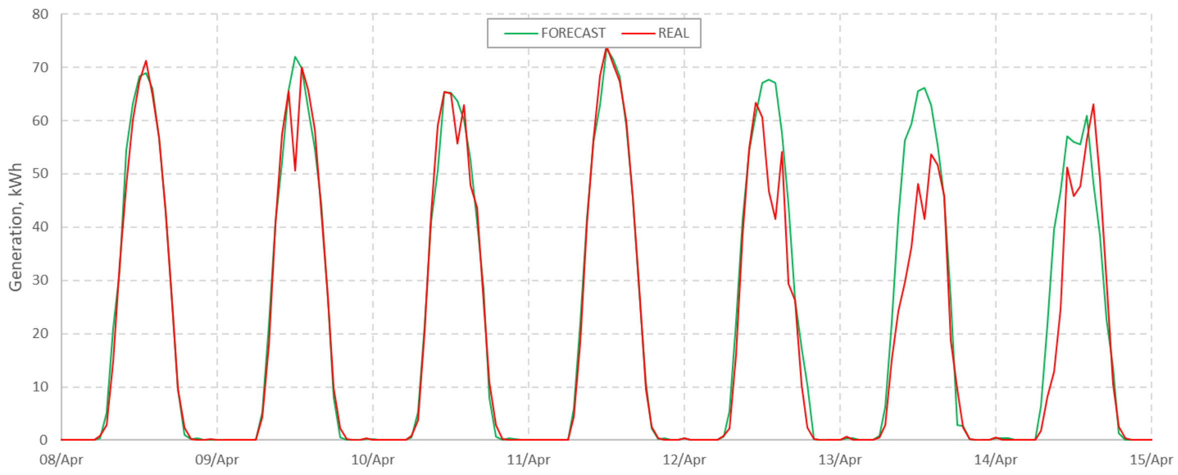
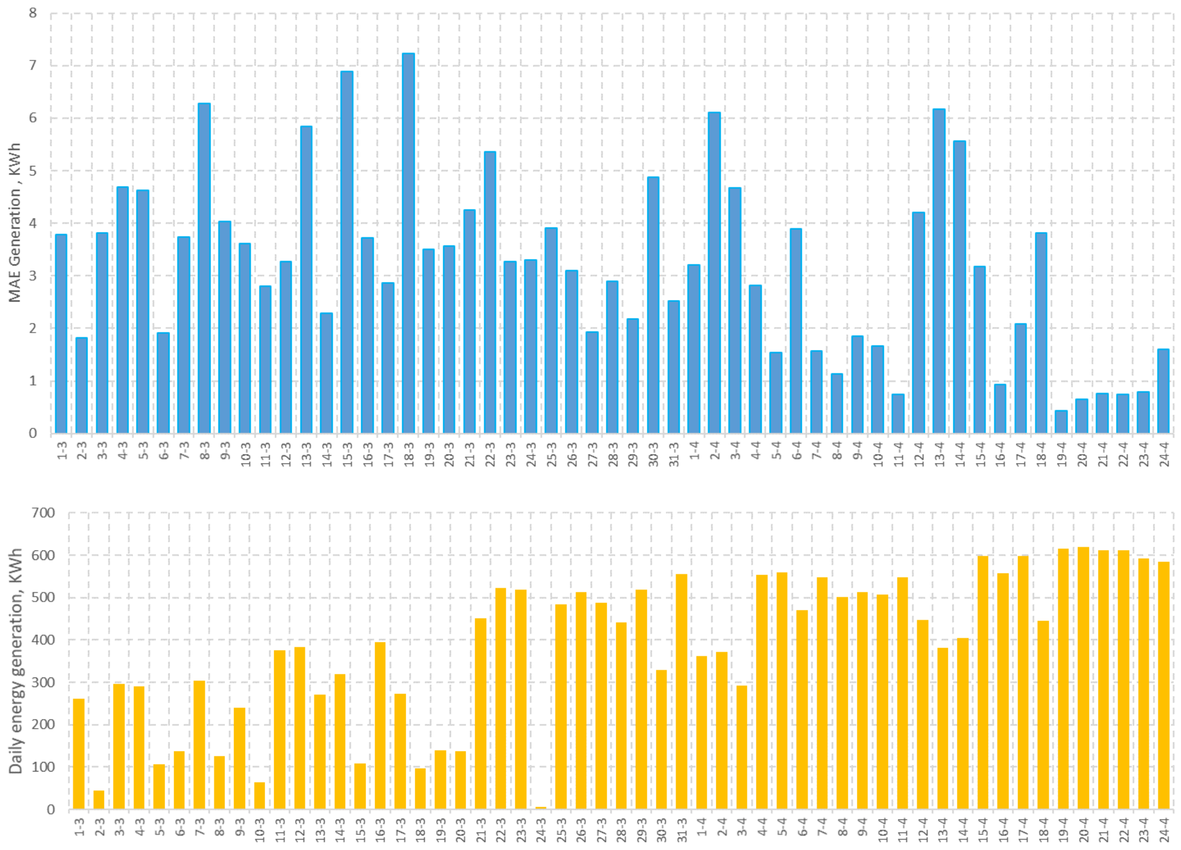


Fig. 13 Daily mean absolute error for PV generation compared to daily generation (March and April 2020)



4.6.3 Building consumption forecasting

The second series of experiments include the forecasting of building consumption, ideally for the next day. The steps involved are identical to the ones presented during PV generation forecasting with differences in parameter definition, which will be further explained below.

Construction of past and future data frame

a) Since outdoor temperature affects the power consumption of the building e.g. the HVAC, to forecast building consumption, historical data about power consumption and outdoor temperature should be retrieved from the laboratory's database. Power consumption values should also be grouped and hour sampled in order to calculate their integral in time space. The result will be the total energy consumed every hour.

b) Two regressors will be considered in this forecasting sequence, outdoor temperature and building ventilation. In the case of SGIL, the HVAC system is fully functional every working day from 07:00 to 23:00, and off the rest of the time and the weekends.

c) Buienradar daily weather forecast from a nearby station will provide us with the outdoor temperature regressor column for the future data frame.

d) The past data frame includes building consumption, temperature and ventilation columns.

e) The future data frame consists of past and Buienradar forecast temperature data as well as a ventilation regressor column.

f) The future data frame as in the in the case of PV generation will include information about the forecast regressor value and date stamp information.

Running the Facebook Ax tool

Building consumption exhibits daily, weekly and yearly seasonalities, but the focus will be on daily and weekly, due to the data load involved when considering yearly seasonalities. Two different fourier order variables linked to daily and weekly seasonality will be defined as inputs to Ax, compared to just one fourier order variable in the case of PV generation forecasting.

Moreover, two regressor prior scale arguments will be used related to temperature and ventilation: temperature regressor prior scale and ventilation regressor prior scale, respectively. As a final step, the best parameter values produced by Ax will then be fed to Prophet including the initial period, daily fourier order, weekly fourier order, changepoint prior scale, temperature regressor prior scale and ventilation regressor prior scale. As a reminder, temperature and ventilation regressor prior scales adjust the overfitting of each regressor, whereas the changepoint prior scale influences the trend flexibility.

Running the Facebook Prophet tool

Figures 14-15 show the forecast prediction graphs generated with the Prophet and Ax combination for two consecutive months (March and April, transition to spring season). The red line illustrates the real measurements, whereas the green line is the forecast for the building consumption. It is obvious that the forecast model works well as it follows the general trend of the real data while also exhibiting a low MAE. Both the long term seasonality and the weekly seasonality (Figure 15 - during the weekend the ventilation of the building is reduced) are captured by the forecasting algorithm.

The forecast was issued on an hourly basis and stored in the database of the laboratory.

Fig. 14. Forecast prediction graphs for the building consumption (March and April 2020)

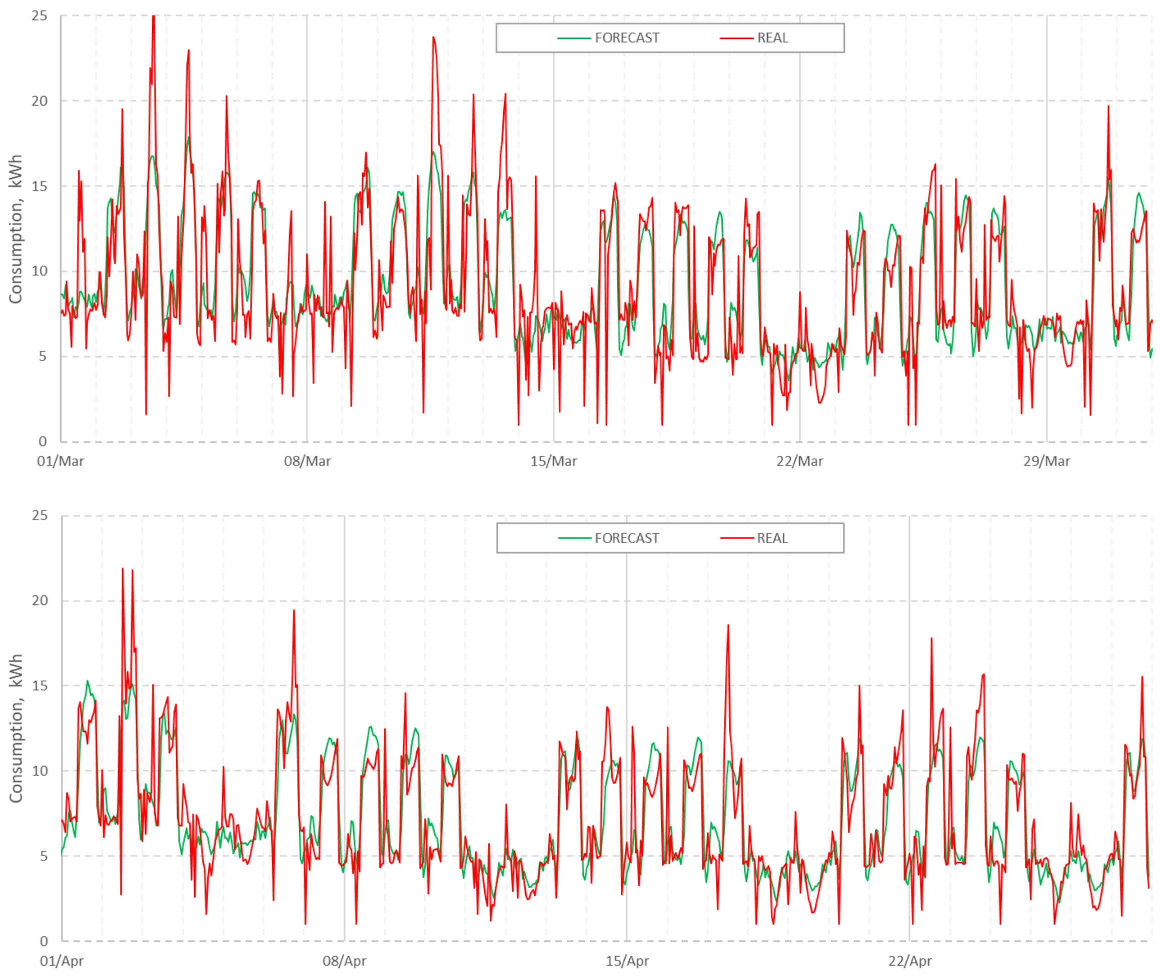


Fig. 15. Forecast prediction graphs for the building consumption (detail 08-15 April)

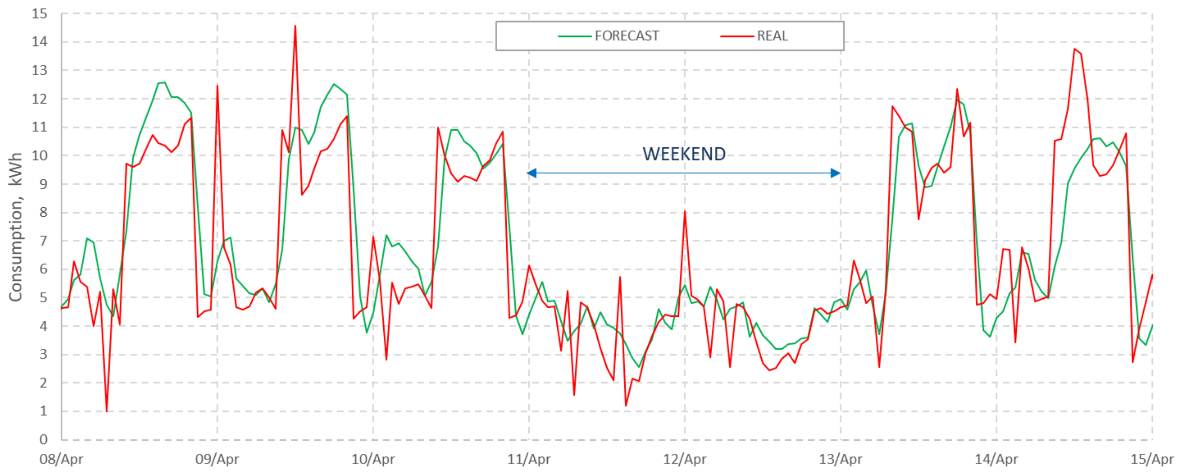
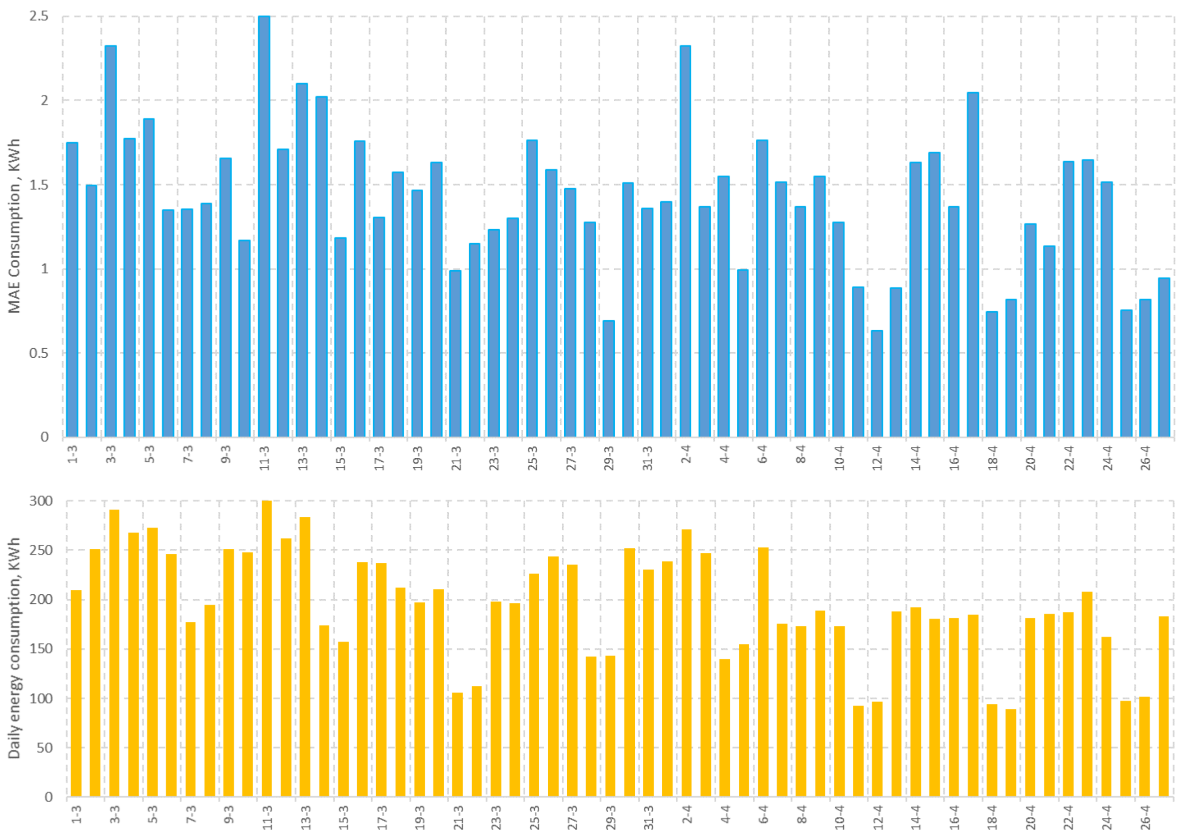


Figure 16 shows the daily mean absolute error (MAE) for the forecasted months. The total MAE for March is 1.56 and for April is 1.31 (the building consumed an average of 220kWh/day in March and 170kWh/day in April).

Fig. 16. Daily mean absolute error for the building consumption compared to daily consumption (March and April 2020)



4.7 Optimization model

There are a lot of different approaches on the scheduling of smart devices [17][16] [24]. The PyEMS approach is based on the resolution of a deterministic optimization model leveraging state-of-the-art forecast described in the previous section.

After the system is defined and all data is collected, an optimization model is automatically created based on the defined system components. As an example, an optimization model for a general system is presented in this section. The components of the system considered are:

- Stochastic generator: an electrical generator that cannot be controlled by the user, but instead its production is uncertain, i.e. photovoltaic panels.
- Stochastic load: any device, which is not flexibly operable and whose consumption is uncertain.
- Electrical battery: a device able to store electrical energy.
- External electrical grid: a connection of the building with the Distribution System Operator (DSO) grid allowing the purchase and sale of electricity on demand.

Before introducing the model, we define the notation used in this section:

Indexes and sets

- $\mathcal{T}$: Ordered set of the time periods in the simulation horizon.
- t: time period index of the $\mathcal{T}$ set.
- T: Last time period of the $\mathcal{T}$ set.

Variables

- E_t^{Chrg}: Gross energy used to charge the battery at time t including losses (kWh).
- E_t^{Dis}: Net energy discharged from the battery at time t excluding losses (kWh).
- E_t^{Buy}: Energy bought from the grid at time t (kWh).
- E_t^{Sell}: Energy sold to the grid at time t (kWh).
- SOC_t: State of charge of battery at time t (%).
- y_t^{Grid}: Binary auxiliary variable to ensure the grid is either selling or buying at time t.
- y_t^{Batt}: Binary auxiliary variable to ensure the battery is either charging or discharging at time t.

Parameters

- E_t^{PV}: Total forecast energy production of the photovoltaic array (kWh).
- E_t^{Load}: Total forecast energy load of the system (building) considered (kWh).
- p_t^{Buy}: Price of electricity at time t (€/kWh).
- p_t^{Sell}: Income obtained for selling electricity at time t (€/kWh).
- C^{Buy}: Maximum allowed energy to be bought in each period t (kWh).
- C^{Sell}: Maximum allowed energy to be sold in each period t (kWh)
- C^{Batt}: Battery capacity (kWh).
- C^{Chrg}: Maximum allowed energy to charge the battery in each period (kWh).
- C^{Dis}: Maximum allowed energy to discharge the battery in each period (kWh).
- η^{Chrg}: Battery charging efficiency (%).
- η^{Dis}: Battery discharging efficiency (%).
- SOC^{init}: Measured initial state of charge (SOC) of the battery (%).

- SOC^{reqr}: Minimum required SOC in the battery at the end of the simulation horizon (%).
- SOC^{min}: Minimum SOC allowed in the battery during the simulation horizon (%).
- SOC^{max}: Maximum SOC allowed in the battery during the simulation horizon (%).

The optimization model is defined in terms of energy due to an easier formulation and higher accuracy compared to being expressed in power terms. The model formulated below follows a deterministic approach where the forecast load and energy production are assumed to be the true values. This is, of course, an approximation, as real values usually deviate from the forecast. In future versions, other techniques can be employed to account for the intrinsic uncertainty such as considering scenarios [18] or robustification techniques.

Following the above notation, the PyEMS underlying optimization model can be written as follows:

$$min \sum_{t \in \mathcal{T}} p_t^{Buy} E_t^{Buy} + p_t^{Sell} E_t^{Sell} \tag{1}$$

subject to:

$$E_t^{PV} - E_t^{Load} + E_t^{Dis} - E_t^{Chrg} + E_t^{Buy} - E_t^{Sell} = 0 \quad \forall t \in \mathcal{T} \tag{2}$$

$$0 \leq E_t^{Buy} \leq y_t^{Grid} C^{Buy} \quad \forall t \in \mathcal{T} \tag{3}$$

$$0 \leq E_t^{Sell} \leq \left(1 - y_t^{Grid}\right) C^{Sell} \quad \forall t \in \mathcal{T} \tag{4}$$

$$y_t^{Grid} \in \{0, 1\} \quad \forall t \in \mathcal{T} \tag{5}$$

$$C^{Batt}(SOC_{t+1} - SOC_t) = \eta^{Chrg} E_t^{Chrg} - \frac{1}{\eta^{Dis}} E_t^{Dis} \quad \forall t = 1, \dots, T-1 \tag{6}$$

$$C^{Batt}(SOC^{reqr} - SOC_t) = \eta^{Chrg} E_t^{Chrg} - \frac{1}{\eta^{Dis}} E_t^{Dis} \quad t = T \tag{7}$$

$$SOC^{min} \leq SOC_t \leq SOC^{max} \quad \forall t \in \mathcal{T} \tag{8}$$

$$SOC_1 = SOC^{init} \tag{9}$$

$$0 \leq E_t^{Chrg} \leq y_t^{Batt} C^{Chrg} \quad \forall t \in \mathcal{T} \tag{10}$$

$$0 \leq E_t^{Dis} \leq (1 - y_t^{Batt}) C^{Dis} \quad \forall t \in \mathcal{T} \tag{11}$$

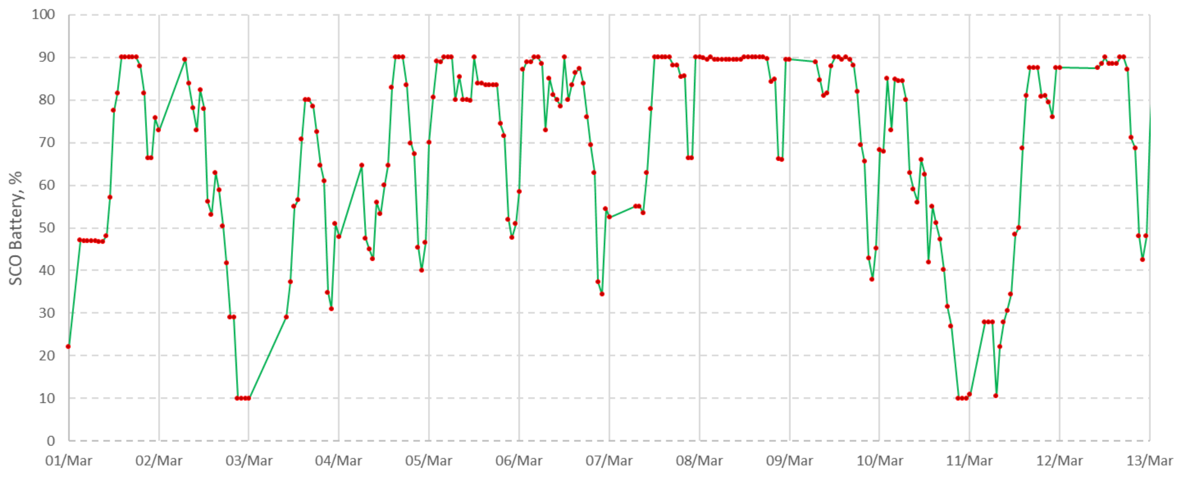
$$y_t^{Batt} \in \{0, 1\} \quad \forall t \in \mathcal{T} \tag{12}$$

The objective function (1) of this particular model, seeks to minimize the energy cost for the planned time horizon. Other objective functions can be used instead, to improve battery life time or to reduce CO2 emissions. The pursued objective is limited by a set of constrains that defines the relation between system components. Equation (2) ensures energy balance in the system (building). Equations (3)–(5) apply when considering the external electrical grid and limit the amount of energy that can be exported or imported, due to, i.e. contractual or technical limitations, in every period of time. The binary variable y_t^{Grid} ensures that the total energy transferred between the system and the grid at time t results in either importing or exporting energy, but not both at the same time. The presence of a battery requires the definition of equations (6)–(12). Equations (6) and (7) take into consideration the relation between the energy transferred to / from the battery and the variation in the state of charge (SOC) of the battery. In (7), a target SOC^{reqr} is introduced as a parameter to specify the desired SOC of the battery at the end of the simulation horizon. For example, the SOC^{reqr} can be lower if the following days are forecast to be sunny and an excess of PV is expected. Equation (8) ensures the SOC is bounded between the operational limits. Equation (9) uses the current (measured) SOC of the battery as the initial SOC for the optimisation. Finally, equations (10)–(12) are equivalent to (3)-(5) and ensure that the battery is only charging or discharging at any time t.

The resulting optimization model presented above can be categorised as a Mixed Integer Linear Programming (MILP) according to mathematical programming and operation research theory [7]. This means, that all restrictions are linear and all variables are either continuous or binaries. These models are computationally harder to solve than their standard Linear Programming (LP) counterparts, but with current computational power and modern off-the-shelf solvers, medium-size MILP can be solved very efficiently. Provided a solution exists, the results obtained by the solvers is guaranteed to be the best possible solution (within a tolerance) under the conditions specified in the model.

Figure 17 presents the behaviour of the controlled battery based on the PyEMS toolkit, considering the variable electricity prices, solar generation, building consumption and the current state of charge. The upper and lower limit of the SOC of charger where limited to 90% and respectively 10% in order to extend the lifespan of the battery.

Fig. 17. Optimisation model results detail (Battery SOC)



5 Technical aspects

In this section, we briefly walk through the initial steps required to develop an EMS based on the PyEMS Python package. This section covers how to install PyEMS, some tips regarding the deployment of the EMS using third-party tools, an introduction to the PyEMS folder structure and, lastly, an implementation example of the entry file (main.py) of a custom EMS.

5.1 Installation

At the moment, PyEMS is not included in the pip repository and a manual installation is required. A good practice, when starting a new project in Python, is to create a new environment through *pip* (installing Python packages) *venv* (virtual environment manager), or Anaconda. If the standard configuration is used, Facebook Prophet is used to issue the forecasts. In this case, creating an Anaconda [2] environment is strongly recommended, due to an issue in the installation of one of the required packages of Prophet on Windows. For more information, see Prophet installation docs PyStan [22] for Windows.

Download the source files in a folder and create your own implementation project at the same level:

```
root_folder/
├── PyEMS/
│   ├── PyEMS/
│   └── other_files.*
└── your_implementation/
    ├── main.py
    └── your_modules.py
```

Create an Anaconda environment and install the requirements from the requirements.txt file:

```
conda create --name ems_env
conda install --file /path/to/PyEMS/requirements.txt
```

In the main.py, include the following code:

```
import os
import sys
sys.path.append(os.path.abspath(r'..\PyEMS'))
import PyEMS.environ as ems
```

In the main.py you can create and use PyEMS entities like the following:

```
system = ems.System(name='your_system')
```

5.2 Deployment

An Automation System (AS) like Home Assistance (HASS) [14] or OpenHAB [15] could be used to monitor and operate a building. This means collecting data of different devices (PV panels, building load, temperature etc.) into a permanent infrastructure like a database and control them by switching them on and off or controlling a certain parameter. Task schedulers like the Windows task scheduler or Linux cron jobs could also be used in conjunction or not with the AS, to run the EMS at a certain frequency. The function of the latter group could be also assumed by the Automation System.

Deploying the Automation System through containers is a good practice to isolate the installation from side effects and ease the management, monitoring and removal of service. For that purpose, we recommend the use of Docker [9].

5.3 Code Structure

The main folder structure of the package is organized as follows:

```
PyEMS/
├── docs/
├── examples/
├── PyEMS/
│   ├── contrib/
│   ├── core/
│   ├── environ/
│   ├── tools/
│   └── config.py
└── tests/
```

There are files, like the README.md, in the outer most level of the package not included above, as they are common files in any Python package. We briefly comment on the purpose of each of the listed elements:

- docs: this folder is dedicated to documentation. Some guidelines regarding how to install and use the PyEMS are provided in the files inside.
- examples: the name is self-explanatory. Some examples on how to implement an EMS are provided here.
- PyEMS: This folder contains the elements of the package:
    - contrib: this folder is intended to hold the code related with the PyEMS developed by external contributors.
    - core: is the main folder, where the core or the most important part of the source code is located.
    - environ: this folder contains a single file, __init__.py. With this component, the user can import the Optimizer class through the PyEMS.environ instead of its real path. Through this environ shortcut, the user has access to Simulation, System, System Components, Forecaster, Data Handler, Optimizer, among others.
    - tools: some functions that can be used independently, not only for deploying an EMS but for other purposes can be found here.
    - config.py: this file stores the constants and configurations used throughout the package,
- tests ensuring well-functioning of the code.

```
core/
├── components/
├── entity/
├── forecasting/
├── graphs/
├── iodata/
├── optimization/
├── results/
├── simulation/
├── system/
└── utils/
```

The core folder, contains the main source code and its structure is commented below:

- components: the components are subclasses of the class Entity that represent a physical element, such as, an Electrical Load or an Electrical Generator. These components are intended to be declared as attributes of a System that stores them. The code can be found here.
- entity: the base class for the all entities in the PyEMS package is Entity and is located here. This class allows to implement properties for all entities, like a unique id number, and also allows to distinguish a PyEMS class from the rest Python class by checking if the object is an instance of Entity.
- forecasting: the purpose of the entities is to issue a forecast after collecting historical and auxiliary data. The code here acts as an abstraction layer to interactions with other third-party tools like Prophet or scikit-learn. This is done through classes called Oracles.
- graphs: this folder includes modules with specific graph styles to plot the results.
- iodata: this folder is dedicated to the input/output operation, with an emphasis on data collection from different sources. The user can obtain data through .csv files, InfluxDB or APIs, e.g. to retrieve the Spanish household electricity prices.
- optimization: this folder is devoted to determine the best schedule for the devices by solving the optimization model. It houses the code of the Optimization class and other auxiliary modules related to solving the optimization model.
- results: after the optimization model is solved, a Result object is created. Through this object, results can be depicted or stored.
- simulation: this folder is dedicated to the Simulation entity of the code. The Simulation allows the user to select the type of operational mode, a single step mode for real operation or rolling window to analyse different behaviours.
- system: the System represents the smart building or home. The code of this entity is here.
- utils: other important modules that are fundamental to EMS functioning, but not necessarily belonging to either of the previous code categories. The time.py code contains a critical function for time management across the package.

5.4 Implementation examples

This section provides an example of an EMS implementation through the PyEMS package. The code includes comments for clarification. We have tried to use self-explanatory names of variables and functions. The code below corresponds to the main.py that should be executed periodically by a third-party tool, like a task scheduler or a cron job, just before the next time step begins.

```
# IMPORTS


# Standard and third-party imports.
import os
import sys
import datetime
from pathlib import Path


# PyEMS imports.
```

```
sys.path.append(os.path.abspath(r'..\PyEMS'))  # Importing PyEMS from local folder.
import PyEMS.environ as ems  # Importing general tools.
from PyEMS.core.iodata.ioinflux import write_point_to_influxdb # Importing a specific function.

# Other custom user made imports.
from data_handler_single_step import CustomDataHandler

print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S: Starting EMS.'))

# DEFINE PATHS

# Defining paths and creating output folder.
script_path = os.path.dirname(__file__)
parent_path = os.path.dirname(script_path)
output_path = os.path.join(script_path, 'EMS_results')
Path(output_path).mkdir(parents=True, exist_ok=True)

# PARAMETER DEFINITION

DB_ROUTE = {'host': 'URL_or_IP', 'port': PORT}
DB_CREDENTIALS = {'database': 'database_name', 'username': 'user', 'password': 'user_password'}

battery_parameters = {
  'batt_C': 50, # (kWh) Capacity of the battery.
  'soc_lb': 0.10, # (%) Minimum battery SOC at any period.
  'soc_ub': 0.90, # (%) Max battery SOC at any period.
  'batt_chrg_speed': 10, # (kWh) Max energy the battery is allowed to be charged per period.
  'batt_dis_speed': 10, # (kWh) Max energy the battery is allowed to be discharged per period.
  'batt_chrg_per': 0.85, # (%) Charging efficiency.
  'batt_dis_per': 0.85 # (%) Discharging efficiency.
}

# SETTINGS

# List of Time Zones https://en.wikipedia.org/wiki/List_of_tz_database_time_zones.
ems.Setting.time_zone = 'Europe/Amsterdam' # Local time zone.
ems.Setting.timestep = '15m' # 1m <= time step <= 1h h: hours, m: minutes, s: seconds

# SYSTEM DEFINITION
# Initializing a Custom Data Handler object extended by the user.
custom_handler = CustomDataHandler()

# Initializing Data System object.
system = ems.System(name='Smart_Building')

# Adding a connection to the DSO external supply grid.
system.supply = ems.ElectricalExternalGrid(
  publication_time='20:30', data_handler=custom_handler,
  purchase_label='purchase_prices', sell_label='sale_prices',
)

# Adding one stochastic generator that represents some PV panels.
system.pv_panel = ems.StochasticElectricalGenerator(
  training_span=2*30*24, data_handler=custom_handler,
  historical_label='pv_panels', regressor_labels=['pysolar', 'sunpower'],
)

# Adding one non-flexible load that represents the total total load of the building.
system.total_load = ems.FixElectricalLoad(
  training_span=2*30*24, data_handler=custom_handler,
  historical_label='total_building_load', regressor_labels=['temperature', 'sunpower'],
)

# Adding a battery.
system.battery = ems.ElectricalBattery(
  **battery_parameters, data_handler=custom_handler,
  initial_soc_label='initial_battery_soc', final_soc_label='final_battery_soc',
)

# Initializing an Optimizer using GLPK Solver.
optimizer = ems.Optimizer(
  solver='glpk', full_solver_info=False, display_solver_info=False,
  write_solver_info=True, solver_factory_options={},
  solve_options={}, info_path=output_path, include_timestamp=False
)

# Initializing Simulation.
```

```
simulation = ems.Simulation()

# SIMULATION EXECUTION

# Executing a simulation for a single step, this is the case of real operation.
# The execution of this script should be programmed with a task manager or other third-party tool.
results = simulation.run_single_step(system=system, optimizer=optimizer)

# WRITING RESULTS

# Using default functions of the Results object to save them.
results.write_results_to_csv(
  file_path=output_path, include_timestamp=False
)

results.write_target_soc_to_influxdb(
  soc_entity_id='TARGET_SOC',
  db_connection_parameters={**DB_ROUTE, **DB_CREDENTIALS}
)

# Using handy functions of the PyEMS Toolkit to write additional info to DB,
# in this case, the initial SOC of the battery.
write_point_to_influxdb(
  entity_id='INITIAL_SOC',
  value=system.battery.soc_0 * 100, measurement='%',
  db_connection_parameters={**DB_ROUTE, **DB_CREDENTIALS}
)
```

6 Conclusion

Energy management Systems are called to play an important role in the smart buildings of the future. Increasing complexity in the integration of smart devices and distributed energy sources requires smart controllers to address interoperability problems and make the most of them. In this report, we have presented PyEMS, a versatile toolkit implemented as a Python package that eases the development of a custom EMS. The toolkit is also a valuable tool for researchers and analysts who seek to simulate different control strategies and determine the best ones. The toolkit is open source [20] and free to use and contribute to, although still under development.

This package has been developed and tested in the premises of the Smart Grid Interoperability Lab of the Petten JRC site, in a real setup. The implemented EMS leverages historical, auxiliary and future information together with the Facebook Ax and Prophet packages to issue forecasts of the loads and photovoltaic production of the building. On top of these forecasts, the optimal schedule is determined through a tailored optimization model based equipment installed in the building. This model is defined in Pyomo and solved through an open source GLPK solver. A practical implementation is briefly discussed in this report for that particular building.

This initial version of the toolkit opens many other development possibilities. From an academic perspective, more sophisticated techniques could be used, such as, modelling scenarios or robust models, to account for the intrinsic uncertainty of photovoltaic production or loads. On the other hand, the development of thermal components like HVAC and their integration with electrical ones is also a challenge for the future. Technically speaking, integration with other data sources and databases together with enhanced documentation are part of a list of future priorities. Finally, a general objective of this tool is to contribute to the open energy community and improve the efficiency of smart buildings and houses.

References

[1] A. Al-Ali, A. El-Hag, M. Bahadiri, M. Harbaji, and Y. A. E. Haj, “Smart home renewable energy management system,” Energy Procedia, vol. 12, pp. 120 – 126, 2011, the Proceedings of International Conference on Smart Grid and Clean Energy Technologies (ICSGCE 2011. [Online]. Available: http://www.sciencedirect.com/science/article/ pii/S1876610211018431

[2] Anaconda. [Online]. Available: htps://www.anaconda.com/

[3] Ax - Adaptive Experimentation Platform. Available: https://ax.dev. Last accessed on March 15, 2020.

[4] B. Zhou, W. Li, K. W. Chan, Y. Cao, Y. Kuang, X. Liu, and X. Wang, “Smart home energy management systems: Concept, configurations, and scheduling strategies,” Renewable and Sustainable Energy Reviews, vol. 61, pp. 30–40, 2016.

[5] Buienradar weather data. [Online]. Available: https://www.buienradar.nl/overbuienradar/gratis-weerdata

[6] CEN-CENELEC-ETSI Smart Grid Coordination Group Smart Grid Reference Architecture, 11-2012

[7] D. Han and J. Lim, “Design and implementation of smart home energy management systems based on zigbee,” IEEE Transactions on Consumer Electronics, vol. 56, no. 3, pp. 1417–1425, Aug 2010.

[8] D. Mahmood, N. Javaid, N. Alrajeh, Z. Khan, U. Qasim, I. Ahmed, and M. Ilahi, “Realistic scheduling mechanism for smart homes,” Energies, vol. 9, no. 3, p. 202, 2016.

[9] Docker. [Online]. Available: https://www.docker.com/

[10] F. S. Hillier, Introduction to operations research. Tata McGraw-Hill Education, 2012.

[11] Facebook Prophet; https://facebook.github.io/prophet. Available: Last accessed on March 15, 2020.

[12] GLPK (GNU Linear Programming Kit). Available: https://www.gnu.org/software/glpk/. Last accessed on March 15, 2020.

[13] H. Shareef, M. S. Ahmed, A. Mohamed, and E. Al Hassan, “Review on home energy management system considering demand responses, smart technologies, and intelligent controllers,” IEEE Access, vol. 6, pp. 24 498–24 509, 2018.

[14] Home Assistant. [Online]. Available: https://www.home-assistant. io/. Last accessed on March 15, 2020.

[15] OpenHAB foundation. [Online]. Available: https://www. openhabfoundation.org/. Last accessed on March 15, 2020.

[16] P. Chavali, P. Yang, and A. Nehorai, “A distributed algorithm of appliance scheduling for home energy management system,” IEEE Transactions on Smart Grid, vol. 5, no. 1, pp. 282–290, Jan 2014.

[17] P. Du and N. Lu, “Appliance commitment for household load scheduling,” IEEE Transactions on Smart Grid, vol. 2, no. 2, pp. 411–419, June 2011.

[18] Pandas. [Online]. Available: https://pandas.pydata.org/

[19] Papaioannou I., Tarantola S., Lucas A., Kotsakis E., Marinopoulos A., Ginocchi M., Olariaga Guardiola M., Masera M., Smart grid interoperability testing methodology, EUR 29416 EN, Publications Office of the European Union, Luxembourg, 2018, ISBN 978-92-79-96855-6, doi:10.2760/08049, JRC110455

[20] PyEMS. [Online]. Available: https://github.com/covrig/PyEMS. Author: Munoz Diaz M.A., Covrig C.F.

[21] Pyomo open-source optimization modelling language. Available: http://www.pyomo.org/. Last accessed on March 15, 2020.

[22] PyStan. [Online]. Available: http://pystan.readthedocs.io/en/latest/

[23] R. Missaoui, H. Joumaa, S. Ploix, and S. Bacha, “Managing energy smart homes according to energy prices: Analysis of a building energy management system,” Energy and Buildings, vol. 71, pp. 155 – 167, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/ pii/S0378778813008335

[24] S. Zhou, Z. Wu, J. Li, and X.-p. Zhang, “Real-time energy control approach for smart home energy management system,” Electric Power Components and Systems, vol. 42, no. 3-4, pp. 315–326, 2014.

[25] Scikit-learn. [Online]. Available: https://scikit-learn.org/

[26] Smart Grid Interoperability Laboratory, [Online]. Available: https://ses.jrc.ec.europa.eu/sgil-petten

[27] Smart Grid Interoperability Laboratory, [Online]. Available: https://ses.jrc.ec.europa.eu/sgil-petten

[28] Sobol sequence. [Online]. Available: https://ax.dev/docs/models.html/

[29] X. Wu, X. Hu, S. Moura, X. Yin, and V. Pickert, "Stochastic control of smart home energy management with plug-in electric vehicle battery energy storage and photovoltaic array," Journal of Power Sources, vol. 333, pp. 203 – 212, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S037877531631357X

[30] Y. Liu, B. Qiu, X. Fan, H. Zhu, and B. Han, "Review of smart home energy management systems," Energy Procedia, vol. 104, pp. 504 – 508, 2016, clean Energy for Clean City: CUE 2016–Applied Energy Symposium and Forum: Low-Carbon Cities and Urban Energy Systems. [Online]. Available: http: //www.sciencedirect.com/science/article/pii/S1876610216316435

[31] Y. Ozturk, D. Senthilkumar, S. Kumar, and G. Lee, "An intelligent home energy management system to improve demand response," IEEE Transactions on Smart Grid, vol. 4, no. 2, pp. 694–701, June 2013.

[32] Y. Son, T. Pulkkinen, K. Moon, and C. Kim, "Home energy management system based on power line communication," IEEE Transactions on Consumer Electronics, vol. 56, no. 3, pp. 1380–1386, Aug 2010.

[33] Y. Wi, J. Lee, and S. Joo, "Electric vehicle charging method for smart homes/buildings with a photovoltaic system," IEEE Transactions on Consumer Electronics, vol. 59, no. 2, pp. 323–328, May 2013.

[34] Z. Zhao, W. C. Lee, Y. Shin, and K. Song, "An optimal power scheduling method for demand response in home energy management system," IEEE Transactions on Smart Grid, vol. 4, no. 3, pp. 1391–1400, Sep. 2013.

List of abbreviations and definitions

API	Application Programming Interface
AS	Automation System
DER	Distributed Energy Resources
DR	Demand Response
EMS	Energy Management System
GLPK	GNU Linear Programming Kit
HA	Home Automation
HASS	Home Assistant
HVAC	Heating, Ventilation and Air Conditioning
ICT	Information and Communications Technology
I/O	Input/Output
IoT	Internet of Things
JRC	Joint Research Centre
JSON	JavaScript Object Notation
LP	Linear Programming
MILP	Mixed Integer Linear Programming
ML	Machine Learning
NAS	Network-Attached Storage
PV	Photovoltaic (panels)
SGIL	Smart Grid Interoperability Lab
SOC	State of Charge (of a battery)
UTC	Coordinated Universal Time

List of figures

Fig. 1. Smart Grid Interoperability Lab.. 4
Fig. 2. Smart Grid Interoperability Lab facilities 5
Fig. 3. Electric vehicle and Vehicle to Grid (V2G) charger 5
Fig. 4. SGAM framework [6] 6
Fig. 5. Diagram of the energy elements of a Smart House 8
Fig. 6. Input and output information in an EMS execution for the planning of the SOC of a battery............. 9
Fig. 7. Summary steps of an EMS execution...11
Fig. 8. Sample JSON formatted Buienradar weather forecast data 17
Fig. 9. Ax-Prophet process diagram 19
Fig. 10. Ax parameter graphs and contour plots 20
Fig. 11. Forecast prediction graphs for PV generation (March and April 2020) 21
Fig. 12. Forecast prediction graphs for PV generation (detail 08-15 April) 21
Fig. 13 Daily mean absolute error for PV generation compared to daily generation (March and April 2020) ...22
Fig. 14. Forecast prediction graphs for the building consumption (March and April 2020).......................23
Fig. 15. Forecast prediction graphs for the building consumption (detail 08-15 April)............................24
Fig. 16. Daily mean absolute error for the building consumption compared to daily consumption (March and April 2020)...24
Fig. 17. Optimisation model results detail (Battery SOC) 27

List of tables

Table 1. Ax provided best parameter values 20

KJ-NA-30211-EN-N

doi:10.2760/822668

ISBN 978-92-76-18820-9