SoftwareX 16 (2021) 100839

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

pycity_scheduling—A Python framework for the development and assessment of optimisation-based power scheduling algorithms for multi-energy systems in city districts

Sebastian Schwarz *, Sebastian Alexander Uerlich, Antonello Monti

Institute for Automation of Complex Power Systems, E.ON Energy Research Center, RWTH Aachen University, Mathieustrasse 10, Aachen, Germany

ARTICLE INFO

Article history:
Received 30 November 2020
Accepted 1 October 2021

Keywords:
Optimisation framework
Power scheduling algorithm
Multi-energy systems
Smart grid

ABSTRACT

We introduce the open-source Python software framework pycity_scheduling for the effective development, testing, and assessment of optimisation-based power scheduling algorithms for local multi-energy systems in city districts. The framework primarily targets the elaboration of coordination concepts that can efficiently solve the power dispatch problem on the city district level. Its target users are researchers in the field of smart grid applications and the deployment of operational flexibility for local energy systems. Illustrative code examples demonstrate the capabilities of the pycity_scheduling framework and its use cases. The design principles established in pycity_scheduling allows users to access, extend, and modify the Python package without any need for commercial software or licensing concerns.

Code metadata

Current code version	v1.0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-20-00087
Code Ocean compute capsule	https://doi.org/10.24433/CO.4147995.v1
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	Python 3
Compilation requirements, operating environments & dependencies	Python package pycity_scheduling can be used independently of the hardware. The Python site-package requirements are numpy, pandas, matplotlib, pyomo, Shapely, pycity_base, and pytest. Additionally, a mathematical programming solver, which is supported by the Pyomo optimisation modelling library, is required. A Dockerfile that installs all dependencies together with a Linux-based working environment is included in the repository.
If available Link to developer documentation/manual	https://acs.pages.rwth-aachen.de/public/simulation/pycity_scheduling/
Support email for questions	post_acs@eonerc.rwth-aachen.de

* Corresponding author.
E-mail addresses: sebastian.schwarz@eonerc.rwth-aachen.de (Sebastian Schwarz), sebastian.uerlich@eonerc.rwth-aachen.de (Sebastian Alexander Uerlich), post_acs@eonerc.rwth-aachen.de (Antonello Monti).

https://doi.org/10.1016/j.softx.2021.100839

Software metadata

Current software version	v1.0.1
Permanent link to executables of this version	https://git.rwth-aachen.de/acs/public/simulation/pycity_scheduling/-/releases/v1.0.1
Legal Software License	MIT
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows, Unix-like
Installation requirements & dependencies	Python package pycity_scheduling can be used independently of the hardware. The Python site-package requirements are numpy, pandas, matplotlib, pyomo, Shapely, pycity_base, and pytest. Additionally, a mathematical programming solver, which is supported by the Pyomo optimisation modelling library, is required. A Dockerfile that installs all dependencies together with a Linux-based working environment is included in the repository.
If available, link to user manual - if formally published include a reference to the publication in the reference list	https://acs.pages.rwth-aachen.de/public/simulation/pycity_scheduling/
Support email for questions	post_acs@eonerc.rwth-aachen.de

1. Motivation and significance

Modern city districts rely on various forms of energy, specifically electricity, natural gas, and district heating/cooling for residential, commercial, and industrial applications. In the past, these forms of energy and their infrastructures were usually considered separately and typically only analysed on a local scale, e.g., for large energy-demanding consumers or individual power plants solely. However, with the increasing focus on decarbonisation and energy efficiency aspects in city districts – including the effective integration of local renewable energy sources with respect to the idea of electrification and sector coupling – a holistic analysis covering the interactions and interdependencies between these multiple forms of energy becomes imperative. [1]

Scientific literature refers to such holistic approaches, which combine the different energy vectors of electricity, gas, thermal energy, and/or mobility, as the concept of so-called energy hubs or multi-energy systems [1,2]. A multi-energy system defines a local energy system, where multiple energy carriers are converted, conditioned, and stored to match the fluctuating local energy generation and consumption [2]. One of the major challenges for modern and innovative operators of multi-energy systems is in facing the new expectations from society, in which citizens have an increasing interest in sustainable and self-sufficient energy system solutions. Those solutions frequently involve highly customised energy system configurations, where the energy system operator must additionally maintain the system's reliability and profitability in accordance with the current legal regulations. This precondition, however, makes the operation of such specific energy system solutions to a certain extent unique and costly.

Thus, holistic multi-energy system operation approaches have been widely investigated in the recent literature. Because of the technical complexity that comes with advanced multi-energy system solutions, data-driven methods based on mathematical optimisation are considered a promising approach to effectively schedule/dispatch, coordinate, and control the assets inside multi-energy systems. For example, Martínez Ceseña et al. [3] use a mixed-integer linear programming co-optimisation approach embedded into a techno-economic framework to model and assess business cases for different energy services provided by the intelligent operation of local multi-energy microgrid systems. In contrast, Parisio et al. [4] propose a discrete-time linear robust optimisation model to operate heterogeneous assets inside energy hubs at minimal energy expenses under uncertainties. Similarly, Blaud et al. [5] develop an optimisation-based multi-prosumer economic model predictive control model for multi-energy systems, which accounts for load, weather, renewable power and energy grid cost predictions to minimise the overall operation costs.

Despite their valuable contributions to multi-energy system analysis and operation in general, nevertheless, these exemplary references have in common that the presented optimisation-based applications require data, analyses, algorithms and/or workflows, which are hardly to produce or to realise. Although many open-source as well as commercial simulation and optimisation tools for the design, sizing, and operation of multi-energy systems are already available today, compare the comprehensive review studies in [6–8], we can still identify a set of limitations when it comes to the usage of the existing tools. This comprises at least one of the following aspects:

- Adequate methodologies and routines to process input data (e.g., time series data required for the calculation of multi-energy system component load and generation profiles) are not or only partly available.
- The user cannot easily modify, extend or adapt the different optimisation models and scenarios.
- The user can neither influence nor investigate in the asset coordination principles for multi-energy system optimisation (e.g., such as centralised vs. distributed optimisation approaches). This also involves the definition of the multi-energy system's hierarchy and architecture, which the user cannot properly analyse or assess.
- The roles and responsibilities of the energy system operator and other market players and actors are usually not well integrated into the optimisation-based software application.
- The underlying optimisation-based approach and/or multi-energy system scenario setup is not highly scalable.

In the light of these gaps, this paper presents a novel, object-oriented, and open-source Python software framework named pycity_scheduling, which processes data related to the intelligent operation and sophisticated asset coordination of multi-energy systems with a strong focus on the development, testing, and assessment of optimisation-based power scheduling algorithms. The framework tackles the aforementioned drawbacks and allows its users to implement and assess optimisation-based power scheduling applications in a straightforward way. In doing so, it primarily addresses the research in the field of smart grid applications and the deployment of operational flexibility for multi-energy systems. The overall motivation behind the pycity_scheduling framework is therefore to support scientists and engineering professionals in their research activities on the development of effective and scalable power scheduling applications. Modern energy system operators necessitate such power scheduling applications to determine the optimal operation set-points for the system-level control of assets inside a multi-energy system, compare the work in [3–5]. In this context, the proposed framework facilitates that its users can implement and analyse different power scheduling algorithms and methods easily and

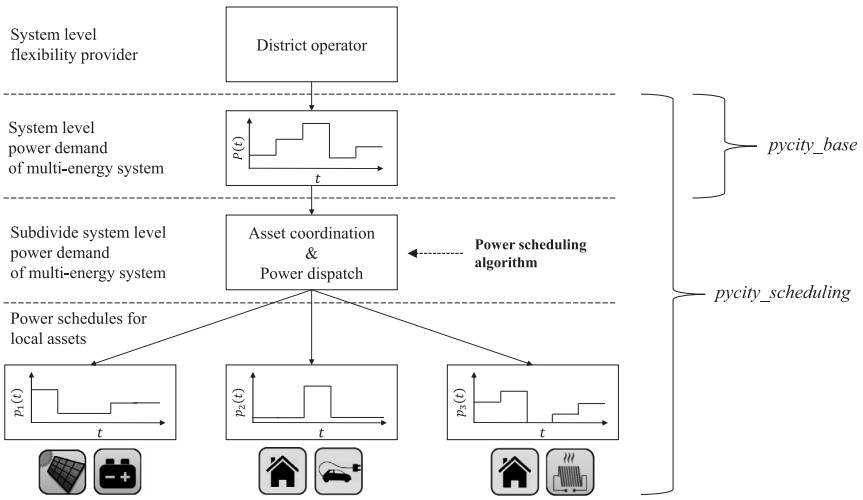

Fig. 1. Power dispatch coordination approach adapted from [9].

flexibly. Thus, we see in the framework a solid basis for a standardised Python programming environment for the development of such power scheduling algorithms applied to multi-energy system power scheduling applications, where several useful pre- and post-processing functionalities support the user with its implementations. One major advantage in using our framework is in its ease of the access and in the extensibility, modifiability, and adaptability of the framework components as well as in its "out-of-the-box" application without long training efforts for software developers and scientists that are already familiar with the Python programming language.

2. Software description

Software package pycity_scheduling constitutes a programming framework for the effective development, testing, and assessment of optimisation-based power scheduling algorithms for multi-energy system applications on the city district level. It is implemented in the Python 3 programming language and licensed under the MIT license. The framework builds on the pycity_base package [10] and is available from the Python Project Index (PyPI), see [11]. We chose the Python programming language for the framework, because it is open-source, platform-independent, widely used in academia, and allows scientists to use and contribute to it easily. Use of the pycity_scheduling framework requires git, Python 3, several free Python 3 site-packages such as numpy [12] and pandas [13], and a mathematical programming solver, which is supported by the Pyomo optimisation modelling library [14]. It is important to emphasise that the usability of the pycity_scheduling framework grounds on simple Python instructions and, hence, Python scripts or Jupyter notebooks have proven an easy and flexible way to codify the entire workflow. The pycity_scheduling package already includes a set of sample scripts, examples, and unit tests. Moreover, the pycity_scheduling framework package comes with a web-based documentation hosted at [15], which describes its components in detail as well as with a continuous integration (CI) GitLab development environment at [16].

The main contribution of the pycity_scheduling framework is in its provision of useful tools and functionalities that address recent challenges and requirements on the application of optimisation-based power scheduling algorithms for city district multi-energy systems. Derived from the overall goal to support scientists and engineering professionals in their research activities on the development of such power scheduling algorithms, we can identify, in particular, three key contributions of our framework as follows:

- The framework provides a set of routines and functionalities, which focus on the definition and solution towards the optimisation-based power dispatch problem for user-defined multi-energy system setups.
- The framework predefines adequate optimisation models for different physical assets and devices inside a multi-energy system in a holistic way including all required inputs, optimisation variables, constraints, and objective functions.
- The framework specifies a sophisticated hierarchy and data model for complex multi-energy system setups comprising the variety and heterogeneity of the physical assets and devices present under varying characteristics, inputs, and operational constraints.

To satisfy and consolidate these three crucial capabilities within the pycity_scheduling framework, our implementations adapt the distributed power dispatch coordination approach including its mathematical formulation from the seminal work in [9] and map it to the field of multi-energy system applications with the support of base package pycity_base [10]. Fig. 1 illustrates this adaptation. In this context, the pycity_scheduling framework implements the principles established in [9], which one can summarise as the intelligent subdivision of the power demand on the city district level into local asset demands through the application of an user-defined power dispatch coordination strategy. The fundamental idea behind this power dispatch strategy is in the short-term power scheduling, i.e., the day-ahead or intraday operation of all local flexible assets and devices inside the city district's multi-energy system by a district operator, which eventually ensures a global power demand and supply balance on the system level. However, the pycity_scheduling framework is not meant to perform detailed physical power system simulations close to the real-time domain, but instead targets the day-ahead time frame with a time discretisation in the range of minutes up to hours. Therefore, the interactions between the different pycity_scheduling framework components, similarly to the work in [9], ground on a simplified copperplate model, which nevertheless fully complies with the assumed time resolution characteristics of the intended target use cases.

2.1. Software architecture and functionalities

Fig. 2 shows the software architecture of the pycity_scheduling framework. In general, the framework architecture can be divided into three core components according to the overall software workflow as visualised in Fig. 2.

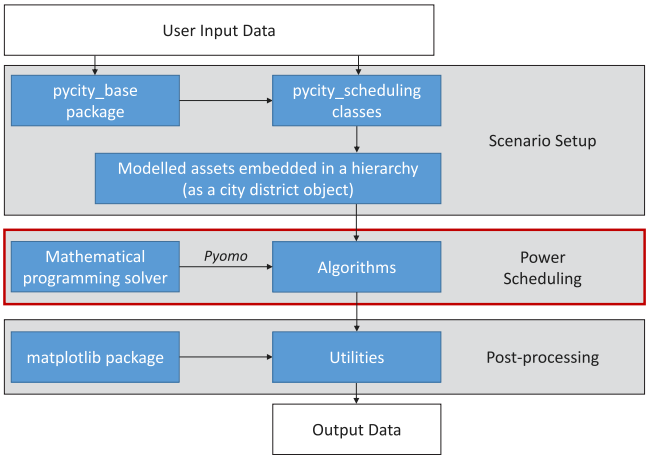

Fig. 2. Overview on the core components of package pycity_scheduling.

The scenario setup component provides the user with modelling functionalities to setup a specific multi-energy system scenario or application. In this initial step, the user specifies a set of input data, which describes the general parameters of the local multi-energy system under investigation. For example, the input data include information about the multi-energy system's geographical location, the local weather data and the considered simulation horizon (e.g., 24 h day-ahead). Moreover, the user also enters information about the number and types of consumers (or related stakeholders) as well as the number of physical assets/devices present in the scenario. For this purpose, every consumer is represented by a *Building* object in the pycity_scheduling framework, which may contain a finite number of (flexible) devices, which in turn are represented through corresponding device objects. In doing so, the pycity_scheduling framework provides the user with a set of hierarchical classes characterising the physical behaviours of the most common energy system devices and components one could find in modern residential, commercial, and industrial multi-energy system setups. This includes the following types of assets:

- inflexible/non-steerable loads
- curtailable loads
- deferrable loads
- gas boilers
- electrical heaters
- combined heat and power units
- heat pumps
- chillers
- thermal energy storage units
- electric vehicles
- battery storage units
- photovoltaic units
- wind turbine generators

Every device's class already comes with a basic mathematical optimisation model, which inherits a set of mathematical constraints and objective functions describing the generic operating conditions of the specific device over the predefined simulation horizon. For example, in the case of an electric vehicle, the optimisation model defines the charging power rate over time subject to the vehicle's battery capacity, its charging efficiency, and the vehicle owner's driving patterns. For a more detailed overview on the modelling approach and optimisation problem formulation per device, we directly refer the reader to the documentation of the pycity_scheduling package at [15]. Moreover, for an adequate device modelling, the pycity_base package is used by the pycity_scheduling framework to obtain standard load time series for the majority of the electrical and thermal loads. It is important to mention that by design all mathematical models can be adapted and expanded easily by the user in order to reflect a more specific physical device behaviour, for instance, based on vendor-specific operational constraints. Furthermore, the framework provides the user with several modelling options on the different predefined optimisation models. For example, the user can select between a modelling approach based on convex or non-convex optimisation constraints. In a next step, the user must instantiate a so-called *CityDistrict* object that encapsulates all the local level devices in one single object according to the required hierarchy for the power dispatch coordination approach as defined in [17]. Fig. 3 visualises this hierarchy by means of the schematic class hierarchy diagram of package pycity_scheduling. Based on this schematic, it becomes evident that all framework objects inherit from the base class *OptimizationEntity*, which includes common attributes such as unique identifiers or optimisation model instances required by all framework objects. Further, the different types of assets stated above derive from the classes *ElectricalEntity* and/or *ThermalEntity* that define basic physical characteristics, i.e., whether an asset represents an electrical, thermal or electro-thermal device. Class *EntityContainer* finally groups and maintains a set of these assets, which the user can then assign to a *Building* object or another subobject such as an *Apartment* inside a *Building*.

The power scheduling component is considered the key element of the pycity_scheduling framework as highlighted by the red box in Fig. 2. Given the multi-energy system scenario setup in the form of a *CityDistrict* object from the previous step, the user can apply an arbitrary power scheduling algorithm to this specific scenario in order to solve the power dispatch problem. The power scheduling component of the pycity_scheduling framework already includes a few reference power scheduling algorithms, whereby the user has the freedom to use them as a starting point for its own implementations. It is important to mention that all new algorithm implementations and modifications must rely on the mathematical programming language Pyomo, compare [14]. Pyomo extends the modelling approach supported by modern algebraic modelling language tools and allows software engineers for a simplified implementation of mathematical expressions in a human-readable way [14]. In other words, Pyomo is an open-source Python library for formulating optimisation problems independently of the used optimisation solver, so that the user can use the pycity_scheduling framework with a third-party optimisation solver of its own choice. Examples for such optimisation solvers are the commercial tools Gurobi [18] and CPLEX [19] as well as the free ones SCIP [20] and BONMIN [21].

Finally, the post-processing component of the pycity_scheduling framework offers the user several functions, routines, and utilities for an effective evaluation and assessment of the previously performed power dispatch optimisation. This comprises the ability to evaluate technical metrics, to plot schedules (e.g., via the matplotlib package [22]) as well as to export data (e.g., to the .csv or .json file format) that is of particular interest to the user. Thus, the main idea of the post-processing component is in the versatile comparison of the characteristics and performance of the user's power scheduling algorithm implementation. More precisely, by providing meaningful output data, the post-processing component should assist the user to evaluate and assess its power scheduling algorithm implementations with minimum efforts.

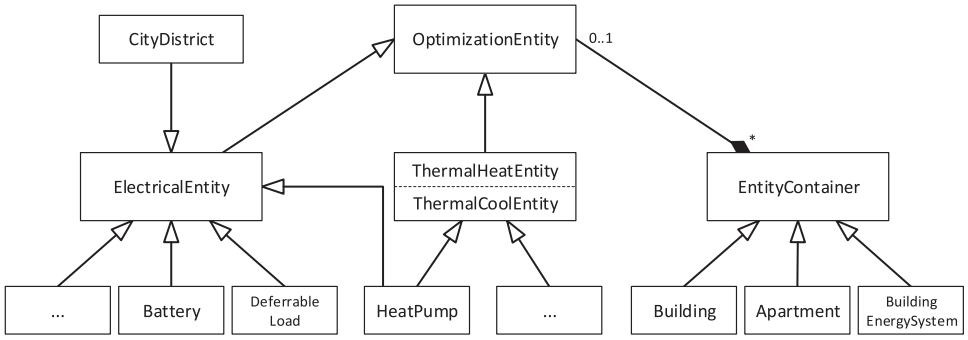

Fig. 3. Schematic hierarchy diagram of the main classes of package pycity_scheduling using Unified Modeling Language (UML) notation.

3. Illustrative examples

To complement the previous overview on the pycity_scheduling package's architecture and functionalities, the following subsections demonstrate basic capabilities and features of the framework. To this aim, we separate the core workflow of our software into three illustrative code examples which are in line with the different framework components as described in Section 2.1. In this context, the following three illustrative code examples define a simple optimisation-based energy cost minimisation use case for the day-ahead planning of assets inside a multi-energy system. Based on this, more complex applications and use cases that take advantage of the framework can be found in the additional example scripts provided by the pycity_scheduling source package.

3.1. Scenario setup code example

The Python code in Listing 1 illustrates the typical workflow for the initial scenario setup step of the intended use case. According to Listing 1, the user must always import all required modules from the pycity_scheduling framework and other third-party modules first, compare lines 1–3. Next and according to lines 5–8, the user must define an (pycity_base) *Environment* object to be used by the subsequent multi-energy system setup and modelling steps. The *Environment* object maintains general data, which is valid for all framework objects and which contains time, weather, and/or energy market price data information. For this reason, all objects in pycity_base/pycity_scheduling usually point to an *Environment*. In this example, we define our *Timer* object to maintain historical time data for one particular day, which is the 15th March of 2018, and hence we choose a time horizon of 24 h with a hourly time discretisation, i.e., 3600 s. For the location of our multi-energy system, we instantiate the *Weather* object with the given coordinates for the city of Aachen, Germany. The *Price* object is instantiated without optional arguments, which makes the pycity_scheduling framework to automatically load historical day-ahead market price data for Germany.

In lines 10–12, now the user can instantiate and define the different assets and load components that are part of the multi-energy system under investigation. For the sake of exemplification, we define a *FixedLoad* object with an annual electrical energy demand of 3000 kWh/a. The parameter *profile_type* is set to "H0", which refers the fixed load (i.e., the inflexible load) to follow the standard load profile characteristics of a residential single-family house. Further, we instantiate a *Photovoltaic* object as well as a *Battery* object in this example, which represent a photovoltaic unit of peak power 6 kWp and a battery storage system of capacity 8.4 kWh with a charging/discharging power rate of 3.6 kW, respectively. In the same way, other assets and loads present in the considered multi-energy system setup could be instantiated and added by the programmer.

For demonstration purposes, we visualise some of the time series data obtained by the instantiated objects for the 15th March of 2018 in lines 14–25. The corresponding plots are shown in Fig. 4. As it can be seen, we can easily access these time series data by using predefined attributes of the different objects, such as *p.da_prices* representing the energy spot market day-ahead prices, *fi.p_el_schedule* representing the fixed load's power demand, and *pv.p_el_supply* representing the photovoltaic unit's power generation over time.

3.2. Power scheduling code example

To illustrate the power scheduling workflow step, we extend the scenario setup code example from the previous section as shown in Listing 2. For this purpose, at first we define the hierarchy of our multi-energy setup according to the code stated in lines 27–35. This can be done in a straightforward way, in which we start with the instantiation of a *Building* object to which one we assign two different *EntityContainer* subobjects, namely an *Apartment* object and a *BuildingEnergySystem* object. The *Apartment* object takes and maintains energy devices that residents may own and operate on the individual apartment level such as the electrical load and the battery unit in our case, whereas the *BuildingEnergySystem* object takes and maintains energy devices that are usually installed on the global building level such as the photovoltaic unit. In the following step, we instantiate a *CityDistrict* object that can bundle a set of different buildings, but which is only one building in this code example for the sake of exemplification (compare lines 34–35). We further define the *CityDistrict* object to possess a price-driven optimisation objective, as we want to perform an energy cost minimisation in this example. However, we could also define the *CityDistrict* object (and if desired the *Building* object, too) to aim for an optimisation objective other than energy cost minimisation instead, such as a peak-shaving or a low CO_2 emission objective. In the following step, the actual day-ahead power dispatch is performed in lines 37–39. In this step, we pass our *CityDistrict* object to the pre-available *CentralOptimization* optimisation algorithm in line 37 and then call the Pyomo's underlying third-party optimisation solver in line 38. As it can be seen, the *mode* parameter is set to "integer" in line 37, which makes the pycity_scheduling framework to use a modelling approach based on mixed-integer programming. Lastly, we can (temporally) store the optimal power schedules obtained by the optimisation solver by calling the *CityDistrict*'s *copy_schedule* function in line 39. As it can be seen, we tag those power schedules with the identifier "optim_schedule" here.

```python
import matplotlib.pyplot as plt
from pycity_scheduling.classes import *
from pycity_scheduling.algorithms import *

t = Timer(op_horizon=24, step_size=3600, initial_date=(2018, 3, 15), initial_time=(0, 0, 0))
w = Weather(timer=t, location=(50.76, 6.07))
p = Prices(timer=t)
e = Environment(timer=t, weather=w, prices=p)

fi = FixedLoad(environment=e, method=1, annual_demand=3000.0, profile_type="H0")
pv = Photovoltaic(environment=e, method=1, peak_power=6.0)
ba = Battery(environment=e, e_el_max=8.4, p_el_max_charge=3.6, p_el_max_discharge=3.6)

plot_time = list(range(t.timesteps_used_horizon))
fig, axs = plt.subplots(1, 3)
axs[0].plot(plot_time, p.da_prices, color="black")
axs[0].set_title("Day-ahead energy market prices [ct/kWh]")
axs[1].plot(plot_time, fi.p_el_schedule, color="black")
axs[1].set_title("Single-family house electrical load demand [kW]")
axs[2].plot(plot_time, pv.p_el_supply, color="black")
axs[2].set_title("Residential photovoltaics generation [kW]")
for ax in axs.flat:
    ax.set(xlabel="Time [h]", xlim=[0, t.timesteps_used_horizon-1])
plt.grid()
plt.show()
```

Listing 1: Illustrative code example on the scenario setup component.

```python
bd = Building(environment=e, objective="none")
bes = BuildingEnergySystem(environment=e)
ap = Apartment(environment=e)
bd.addMultipleEntities(entities=[bes, ap])
bes.addDevice(objectInstance=pv)
ap.addMultipleEntities(entities=[fi, ba])

cd = CityDistrict(environment=e, objective="price")
cd.addEntity(bd, position=(0, 0))

opt = CentralOptimization(city_district=cd, mode="integer")
res = opt.solve()
cd.copy_schedule(dst="optim_schedule")
```

Listing 2: Illustrative code example (continued) on the power scheduling component.

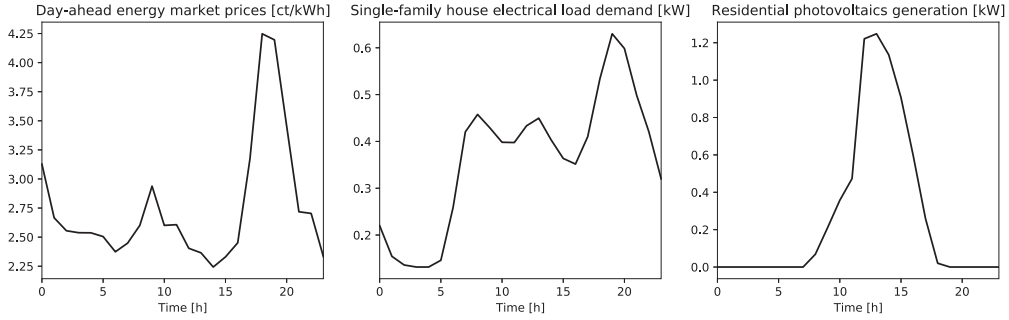

Fig. 4. Time series data plots according to the code in Listing 1.

3.3. Post-processing code example

The code in Listing 3 finalises the overall workflow of this illustrative code example by demonstrating different post-processing functionalities of the pycity_scheduling framework. For this purpose, in a first step we import the framework's post-processing utilities "metric", "plot_schedules", and "write_schedules" as shown in lines 41–43 in Listing 3. In a second step, we call the *CityDistrict*'s *load_schedule* function in line 45 to load the previously stored power schedules as tagged with the identifier "optim_schedule". In a third step, we use the framework's "plot_entity" functionality to plot the schedules of all optimisation variables for the two objects of instance *CityDistrict* and *Battery*. These plots are shown in Fig. 5, where the schedules with the suffix "p_el" denote electrical power and "e_el" denote electrical energy, respectively. From Fig. 5, it becomes evident that the flexible battery device is scheduled in a way such that power is primarily imported from the energy spot market by the district operator during cheap spot market periods, compare the left plot in Fig. 4. This means that the battery unit is incentivised to charge itself during these periods based on the given energy cost minimisation objective. Because of this behaviour, low-cost electrical energy is temporarily stored inside the battery unit. Vice versa, the battery unit is incentivised to discharge itself during expensive energy spot market tariff periods to supply the non-flexible building's electrical load locally during these periods. Moreover, one can see that the battery unit is also charged during time slots of high power penetration by the photovoltaic unit, compare the right plot in Fig. 4. This is because the locally generated photovoltaic energy is assumed to have zero energy costs, i.e., it can be perceived as free. The building's power self-consumption rate metric of approximately 67%, as evaluated in line 50, confirms this circumstance. The remaining 23% of photovoltaic power generation, however, cannot be consumed locally by the building, since the battery unit already operates at its physical charging power limit of 3.6 kW. Finally and for further studies, we export the obtained schedules of the different multi-energy system assets into a file named "cost_otpim.json" according to line 52.

```
from pycity_scheduling.util.metric import self_consumption
from pycity_scheduling.util.plot_schedules import plot_entity
from pycity_scheduling.util.write_schedules import schedule_to_json

cd.load_schedule(schedule="optim_schedule")

plot_entity(entity=cd, schedule=["optim_schedule"], title="City district - Cost-optimal schedules")
plot_entity(entity=ba, schedule=["optim_schedule"], title="Battery unit - Cost-optimal schedules")

print(self_consumption(entity=bd))

schedule_to_json(input_list=[fi, pv, ba], file_name="cost_optim.json", schedule=["optim_schedule"])
```

Listing 3: Illustrative code example (continued) on the post-processing component.

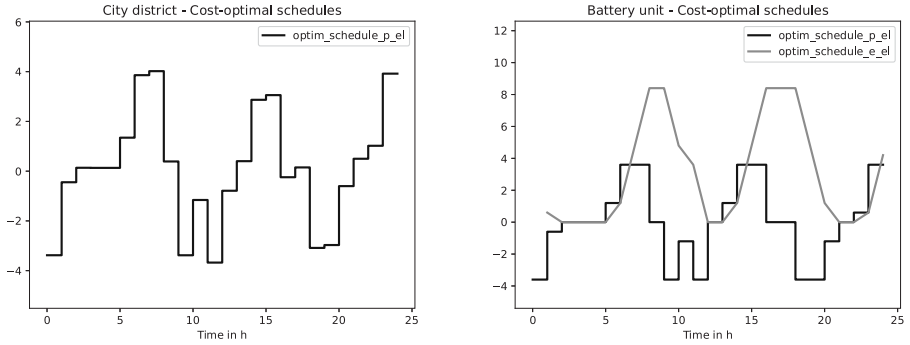

Fig. 5. Plots on the optimal schedules obtained for the city district (left) and the battery unit (right) according to the code in Listing 3. A positive sign represents power/energy import, whereas a negative sign represents power/energy export.

4. Impact

The development of the pycity_scheduling framework was primarily promoted by the energy targets formulated by the European Commission [17,23]. With a growing share of decentralised generators and renewable energy sources installed in distribution grids and local energy systems, the idea behind the framework, therefore, suits to the latest political intentions of the European Union and supports the investigation in smart grids technology and digitalisation in energy. Given the customer-centric focus of the framework's implementation design, it already satisfies the key requirement of the future European energy policy towards a more sustainable European energy supply system as postulated in [23].

Moreover, since the pycity_scheduling framework is open-source and as a Python package fully hardware independent, it can serve as a reference software tool for developing, modifying, testing, and benchmarking optimisation-based power scheduling algorithms for multi-energy systems and provide a common basis for scientists working on different conceptual methods and solutions in this area of research. Unfortunately, many presented power scheduling algorithm realisations in literature are limited in their meaningfulness and reproducibility, as there is no distinct standardisation of scenarios and mathematical programming models that allow one to adequately compare different approaches. Instead, many solutions depend on very specific use case applications and, hence, are hardly to assess and replicate because of the unavailability of inputs, data, and models. Many scientists in this area of research also still make use of their individual but prevalently non-accessible software solutions, which leads to a lack of software code transparency, a duplication of work on similar tasks, and related drawbacks. The pycity_scheduling framework tackles this dilemma. Having a common software framework basis additionally facilitates fruitful discussions as well as an improved exchange of expertise and new ideas among researchers and potential stakeholders involved.

Although the pre-available algorithms in the pycity_scheduling framework are implemented in a sequential way, the pycity_scheduling framework also allows its users to perform scalability analyses and large-scale energy system simulations. This is important for all power dispatch applications targeting complex multi-energy system setups that consist of hundreds of assets and devices. We want to encourage users to implement new power scheduling algorithms to support parallel computations on either tightly or loosely coupled machines. This would facilitate distributed and decentralised computations and therefore could reduce the computation time for complex optimisation problems significantly. In addition, this feature could also be interesting for simulations in which companies or third-party organisations are involved, but cannot share confidential data because of data privacy issues, e.g., due to the safeguard of real customer data or critical infrastructure topology data.

As a scientific tool, the pycity_scheduling framework is currently successfully applied in the European Union's Horizon 2020 research and innovation project IELECTRIX [24] to assess the flexibility potential for a rural local multi-energy system operated by a German distribution system operator. This particular multi-energy system is subject to a high share of renewable energy production, and the objective is to quantify the techno-economic measures in terms of flexibility provision that are of particular value for such systems in the long-term. This also includes day-ahead power scheduling applications as a basis for a novel demand side management concept and as already implemented by the pycity_scheduling framework.

Besides that, several scientific publications already took advantage of the pycity_scheduling framework implementations such as the ones in [25–27]. In addition, the pycity_scheduling package is promoted by the non-profit FEIN Aachen association [28] that provides permanent links to its source code and its documentation on its website.

5. Conclusions

We introduce the pycity_scheduling object-oriented and platform independent Python software package for the effective development, testing, and assessment of optimisation-based day-ahead power scheduling applications for local multi-energy systems. The framework consists of three core components, which

allow the user to code, optimise and evaluate such multi-energy system setups in a straightforward way.

The current package and its future releases are licensed under the MIT license and are publicly available via the Python Project Index (PyPI). The pycity_scheduling package comes with a complete online class documentation and several example scripts that illustrate the main capabilities and features of the framework. This also includes the illustrative code examples discussed in this article.

The framework's novelty and main contribution is in its focus on the elaboration of holistic coordination concepts that can efficiently solve the power dispatch problem on the city district level. It is meant to constitute a solid basis for a standardised Python programming environment for the development of such power scheduling concepts/algorithms, where several useful pre- and post-processing functionalities can support the user with its implementations. This includes the provision of a set of hierarchical classes which physically and mathematically model different assets inside multi-energy systems.

Since Python package pycity_scheduling is open-source, developers are highly encouraged to extend and/or modify its components, models, and features.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work has been undertaken within the framework of the European Union's Horizon 2020 research and innovation programme under IELECTRIX project grant agreement No. 824392.

References

[1] Dall'Anese E, Mancarella P, Monti A. Unlocking flexibility: Integrated optimization and control of multienergy systems. IEEE Power Energy Magaz 2017;15(1):43–52. http://dx.doi.org/10.1109/MPE.2016.2625218.

[2] Geidl M, Koeppel G, Favre-Perrod P, Klockl B, Andersson G, Frohlich K. Energy hubs for the future. IEEE Power Energy Magaz 2007;5(1):24–30. http://dx.doi.org/10.1109/MPAE.2007.264850.

[3] Martínez Ceseña EA, Good N, Syrri AL, Mancarella P. Techno-economic and business case assessment of multi-energy microgrids with co-optimization of energy, reserve and reliability services. Appl Energy 2018;210:896–913. http://dx.doi.org/10.1016/j.apenergy.2017.08.131.

[4] Parisio A, Del Vecchio C, Vaccaro A. A robust optimization approach to energy hub management. Int J Electr Power Energy Syst 2012;42(1):98–104. http://dx.doi.org/10.1016/j.ijepes.2012.03.015.

[5] Blaud PC, Haurant P, Claveau F, Lacarrière B, Chevrel P, Mouraud A. Modelling and control of multi-energy systems through multi-prosumer node and economic model predictive control. Int J Electr Power Energy Syst 2020;118:105778. http://dx.doi.org/10.1016/j.ijepes.2019.105778.

[6] Connolly D, Lund H, Mathiesen B, Leahy M. A review of computer tools for analysing the integration of renewable energy into various energy systems. Appl Energy 2010;87(4):1059–82. http://dx.doi.org/10.1016/j.apenergy.2009.09.026.

[7] Allegrini J, Orehounig K, Mavromatidis G, Ruesch F, Dorer V, Evins R. A review of modelling approaches and tools for the simulation of district-scale energy systems. Renew Sustain Energy Rev 2015;52:1391–404. http://dx.doi.org/10.1016/j.rser.2015.07.123.

[8] Sola A, Corchero C, Salom J, Sanmarti M. Simulation tools to build urban-scale energy models: A review. Energies 2018;11(12):3269. http://dx.doi.org/10.3390/en11123269.

[9] Juelsgaard M. Utilizing distributed resources in smart grids: A coordination approach. (Ph.D. thesis), Aalborg: Automation & Control, Department of Electronic Systems, Aalborg University; 2014.

[10] Schiefelbein J, Rudnick J, Scholl A, Remmen P, Fuchs M, Müller D. Automated urban energy system modeling and thermal building simulation based on OpenStreetMap data sets. Build Environ 2019;149:630–9. http://dx.doi.org/10.1016/j.buildenv.2018.12.025.

[11] The Python Package Index. Pycity-scheduling 1.0.1. 2020, URL https://pypi.org/project/pycity-scheduling/.

[12] Harris CR, Millman KJ, van der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[13] Reback J, McKinney W, Jbrockmendel, van den Bossche J, Augspurger T, Cloud P, et al. Pandas-dev/pandas: Pandas 1.1.3. 2020, http://dx.doi.org/10.5281/zenodo.3509134.

[14] Hart WE. Pyomo - optimization modeling in Python. Springer optimization and its applications, 2nd ed.. Vol. 67, Cham, Switzerland: Springer; 2017.

[15] Institute for Automation of Complex Power Systems, EON Energy Research Center, RWTH Aachen University. The pycity_scheduling documentation: v1.0.1. 2020, URL https://acs.pages.rwth-aachen.de/public/simulation/pycity_scheduling/.

[16] GitLab of the RWTH Aachen University. Repository pycity_scheduling. 2020, URL https://git.rwth-aachen.de/acs/public/simulation/pycity_scheduling.

[17] Expert Group 3 of the Smart Grids Task Force (SGTF) of the European Commission. Regulatory recommendations for the deployment of flexibility: EG3 report. 2015, https://ec.europa.eu/energy/sites/ener/files/documents/EG3Final-January2015.pdf.

[18] Gurobi Optimization LLC. Gurobi optimizer reference manual. 2020, URL https://www.gurobi.com/.

[19] IBM Corp., IBM ILOG CPLEX Optimization Studio User's Manual: Version 12 Release 7. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.studio.help/pdf/usrcplex.pdf.

[20] Gamrath G, Anderson D, Bestuzheva K, Chen W-K, Eifler L, Gasse M, et al. The SCIP Optimization Suite 7.0: Technical Report. http://www.optimization-online.org/DB_HTML/2020/03/7705.html.

[21] Bonami P. Bonmin: Basic open-source nonlinear mixed integer programming. 2020, URL https://github.com/coin-or/Bonmin.

[22] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(3):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.

[23] The European Commission. Clean energy for all europeans. Luxembourg: Publications Office of the European Union; 2019, http://dx.doi.org/10.2833/9937.

[24] The IELECTRIX Consortium. IELECTRIX: European and Indian citizen energy communities for renewable integration and the energy transition: EU H2020 grant agreement no. 824392. 2020, URL https://ielectrix-h2020.eu/.

[25] Molitor C. Residential city districts as flexibility resource: analysis, simulation, and decentralized coordination algorithms. (Ph.D. thesis), E.ON Energy Research Center, Vol. 32, Aachen: Institute for Automation of Complex Power Systems, E.ON Energy Research Center, RWTH Aachen University; 2015.

[26] Diekerhof M, Monti A, Schwarz S. Demand-side management—Recent aspects and challenges of optimization for an efficient and robust demand-side management. In: Classical and recent aspects of power system optimization. Elsevier; 2018, p. 331–60. http://dx.doi.org/10.1016/B978-0-12-812441-3.00012-4.

[27] Diekerhof M, Peterssen F, Monti A. Hierarchical distributed robust optimization for demand response services. IEEE Trans Smart Grid 2018;9(6):6018–29. http://dx.doi.org/10.1109/TSG.2017.2701821.

[28] FEIN Aachen eV. Pycity_scheduling. 2020, URL https://fein-aachen.org/en/projects/pycity_scheduling/.