



open energy modelling framework

Cord Kaldemeyer, ZNES Flensburg

Open Energy Modelling Workshop, Frankfurt

April 19th, 2017

Outline

- Idea of an open framework
- Status quo and quick overview
 - Current projects
 - Possible applications
- How to build an application
 - Basic components
 - Extended applications
- Getting involved
- Conclusion

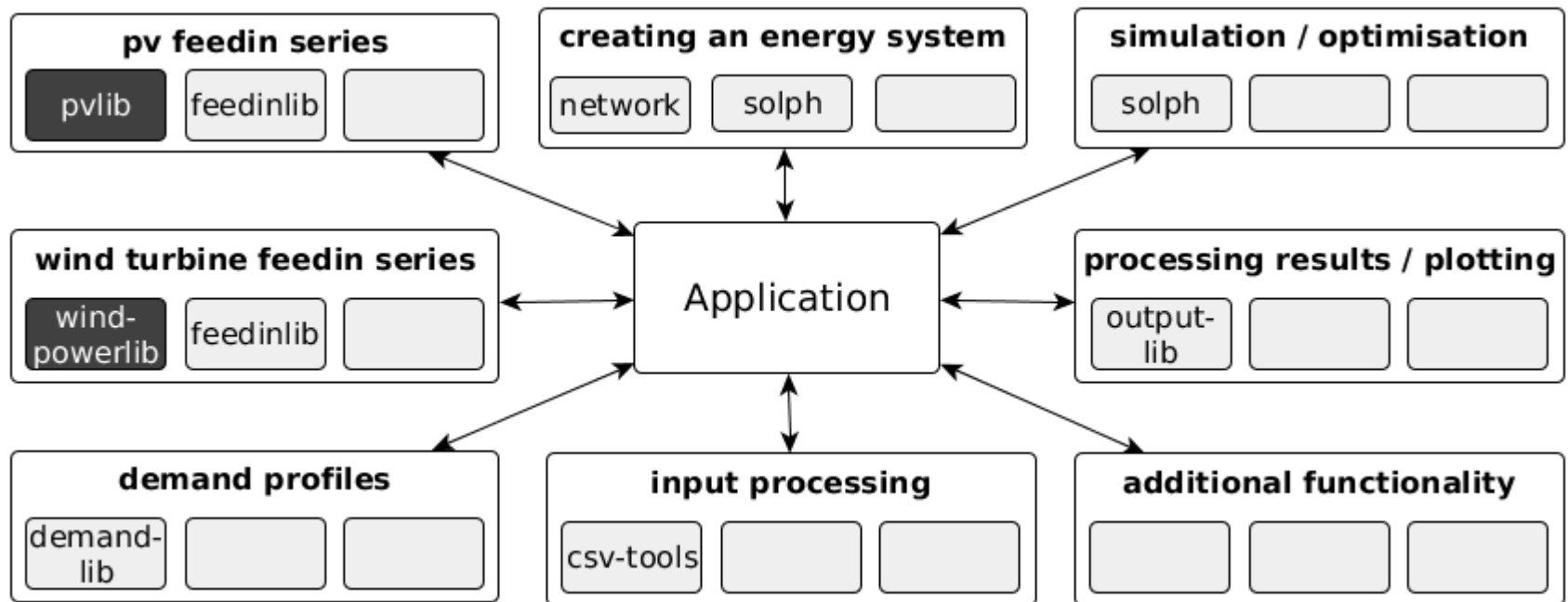
oemof – Initial idea

- People from different institutions
 - Center for Sustainable Energy Systems (ZNES), Flensburg
 - Reiner Lemoine Institut, Berlin
 - Otto-von-Guericke-Universität, Magdeburg
- Individual energy modelling requirements in
 - Research projects
 - Dissertations
 - Student projects
- Tired of repeatedly redundant work concerning data and model development → Join forces

oemof – Initial idea

- Collaborative development of an open and effective framework for energy system analysis
- Generic graph based foundation $G := (N, E)$
- The Set of Nodes N consists of
 - Busses
 - Components
 - Sinks
 - Sources
 - Transformer
- Object-oriented implementation in python
- Usage of existing packages for scientific computing

oemof – Toolbox and applications



oemof – Toolbox

feedinlib

- Generates feedin timeseries of photovoltaic and wind power plants from weather data
- Provides interface to pvlib and windpowerlib

demandlib

- Generates power and heat load profiles for various sectors

windpowerlib

- Generates feedin time series of wind power plants
- Provides power (coefficient) curves for numerous wind turbine types

oemof – Toolbox

solph

- Creates and solves LP/MILP optimization models of energy systems
- Based on pyomo-package

db

- Provides tools for oemof related databases or data APIs (experimental)

outputlib

- Delivers data structures and plots results
- Based on pandas-package

oemof – Applications

renpassG!S

- Bottom up energy system model for Central Europe (LP, welfare maximization with inelastic demand and transshipment)
- More information: https://github.com/znes/renpass_gis

HESYSOPT

- Tool for modelling district heating systems (MILP, operational optimization with exogenous prices)
- More information: <https://github.com/znes/HESYSOPT>

oemof – Applications

reegis_HP

- Ecological and economic evaluation of district heating and combined heat and power in an energy system based on renewable sources (LP, welfare maximization with inelastic demand)
- More information: <http://reiner-lemoine-institut.de/en/ecological-and-economic-evaluation-of-district-heating-and-combined-heat-and-power-in-a-energy-system-based-on-renewable-sources/>

MicroPower

- Small micro grids with spinning reserve and minimal power

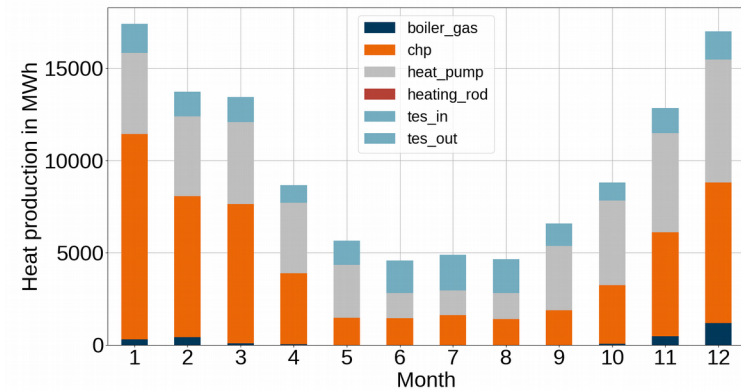
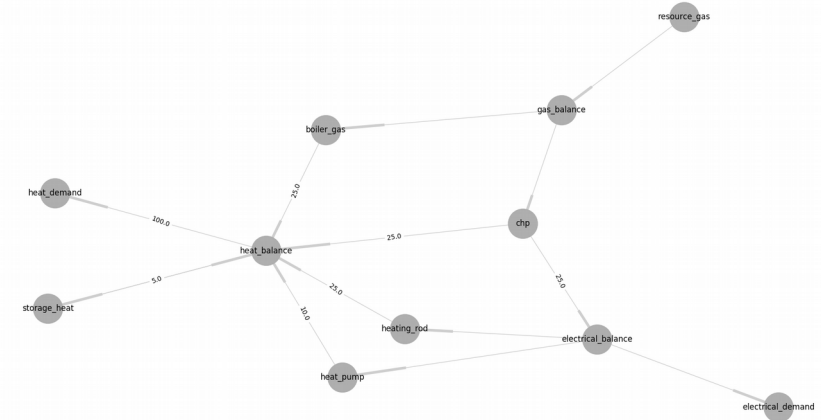
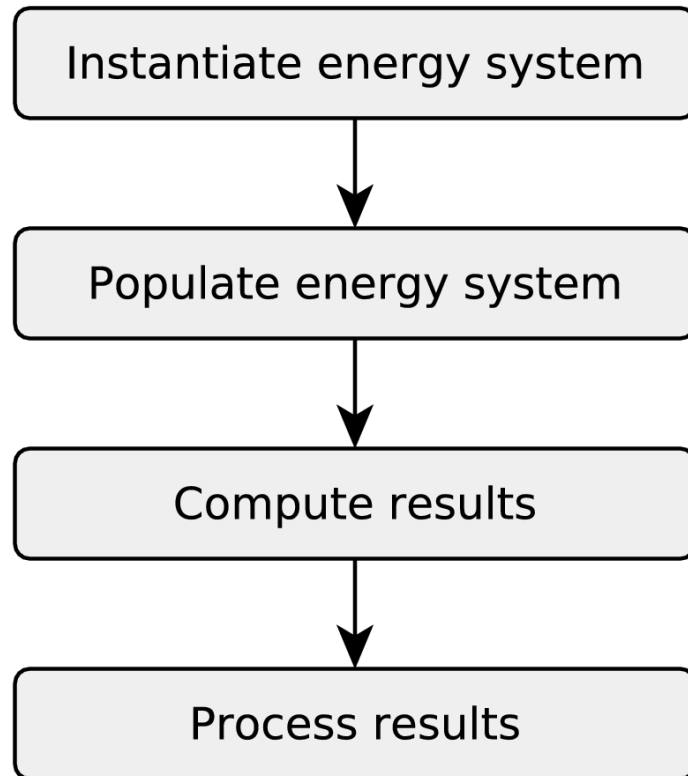
How to build an application

You can build an application...

- ... from scratch using the python classes
- ... using a specific layouted spreadsheet (libreoffice, openoffice, MS Excel, ...)
- ... using scripts creating objects from a database or data file.

The following examples will show how to create a welfare maximization example from scratch using **oemof.network** and **oemof.solph** as currently prominent packages.

How to build an application - Workflow



How to build an application - Logger

- Use the oemof default logger
- All messages are stored in a file
- You can switch between different levels
- Returns oemof version or branch

```
# Default logger of oemof
import logging
from oemof.tools import logger
logger.define_logging()

logging.info("The program started")
logging.debug("This message is only for debugging")
logging.warning("Something odd happened.")
logging.error("That shouldn't happen.")
```

How to build an application - EnergySystem

- The EnergySystem class is the container for your energy model and holds the network, time series, ...
- Pass a time index to initialise it
- Nodes will be added automatically

```
import pandas as pd
from oemof import solph

date_time_index = pd.date_range('1/1/2017', periods=8760,
                                freq='H')

energysystem = solph.EnergySystem(timeindex=date_time_index)
```

How to build an application - Bus

- A Bus class can be seen as a balance for connected components with its respective inputs and outputs
- Every component has to be connected to a bus
- Every connection (edge) is directed

```
from oemof import solph

# create a natural gas bus
bgas = solph.Bus(label="natural_gas")

# create an electricity bus
bel = solph.Bus(label="electricity")

# create a thermal bus
bth = solph.Bus(label="heat")
```

How to build an application - Flow

- Can be interpreted as weight of directed edge between two nodes (Bus and Component)
- No obligatory parameters are needed
- A Flow object has various optional parameters like costs or minimum/maximum values

```
from oemof import solph  
  
my_first_flow = solph.Flow()
```

How to build an application - Source

- A component with no input and one output
- A source has different additional parameters
- **data['pv']** is a normalised feedin series

```
from oemof import solph

# renewable sources (windpower, pv)
solph.Source(label='pv', outputs={bel: solph.Flow(fixed=True,
    actual_value=data['pv'], nominal_value=5400)}))

# commodity source (gas, oil, lignite,...)
solph.Source(label='rgas', outputs={bgas: solph.Flow(
    nominal_value=194397000, summed_max=1000000)}))

# shortage source
solph.Source(label='shortage', outputs={bel: solph.Flow(
    variable_costs=5000)}))
```


How to build an application - Sink

- A component with no input and one output
- A sink has different additional parameters
- **data['demand']** is a normalised demand series

```
from oemof import solph

# demand sink
solph.Sink(label='demand', outputs={bel: solph.Flow(
    fixed=True, actual_value=data['demand'],
    nominal_value=5460)})

# excess sink
solph.Sink(label='excess', inputs={bel: solph.Flow()})
```

How to build an application - Transformer

- A component representing different possibilities for the number of inputs/outputs (currently renamed)
- There are 1xN and Nx1 transformers
- The easiest transformer only has a constant efficiency

```
from oemof import solph

# gas fired power plant (output related definition)
solph.LinearTransformer(
    label='pp_gas',
    inputs={bgas: solph.Flow()},
    outputs={bel: solph.Flow(nominal_value=4711,
                             variable_costs=50,
                             fixed_costs=1000)},
    conversion_factors={bel: 0.58})
```

How to build an application - CHP

- Use the 1xN transformer
- The VariableFractionTransformer can be used to model an extraction turbine
- Heat pumps can be modelled similarly using the Nx1 transformer

```
from oemof import solph

# gas fired chp (input related definition)
LinearTransformer(
    label='pp_chp',
    inputs={bgas: solph.Flow(nominal_value=110,
                             variable_costs=42)},
    outputs={bel: solph.Flow(), bth: Flow()},
    conversion_factors={bel: 0.3, bth: 0.4})
```

How to build an application - Storage

- capacity_loss: relative loss per time step
- inflow_/outflow_conversion_factor: efficiency of charging and discharging
- nominal_capacity: Maximum effective storage capacity

```
from oemof import solph

solph.Storage(
    label='storage', nominal_capacity=6000,
    inputs={bel: solph.Flow(nominal_value=1000)},
    outputs={bel: solph.Flow(nominal_value=1000)},
    capacity_loss=0.01,
    inflow_conversion_factor=1,
    outflow_conversion_factor=0.8)
```

How to build an application – Optimisation

- Pass your EnergySystem instance to the OperationalModel class
- Solve the problem using your favoured solver (open: cbc, glpk, ... / proprietary: gurobi, cplex, ...)

```
from oemof import solph

# initialise the operational model (create problem)
om = solph.OperationalModel(energysystem)

# optionally write lp file to disc (debugging)
om.write(filename, io_options={'symbolic_solver_labels': True})

# set tee to True to get the solver output
om.solve(solver='cbc', solve_kwargs={'tee': True})

results = ResultsDataFrame(energy_system=es)
```

How to build an application – Options

- The **investment object** can be used to have a variable `nominal_value`. Periodical costs per installed capacity have to be defined. Useful to compare alternative capacities (e.g. storage vs. grid capacity expansion)
- **BinaryFlows** can be used to represent load ranges or up- and downtime restrictions (MILP)
- **DiscreteFlows** can be used to force flows to integer e.g. for discrete power blocks (MILP)

How to build an application – Own extensions

- You can **add your own components** with some fancy internal behaviour in your application
- You can **add additional constraints** e.g. connecting two flows
- There are existing examples how to add constraints and components
- We are currently trying to simplify the process of adding constraints and components

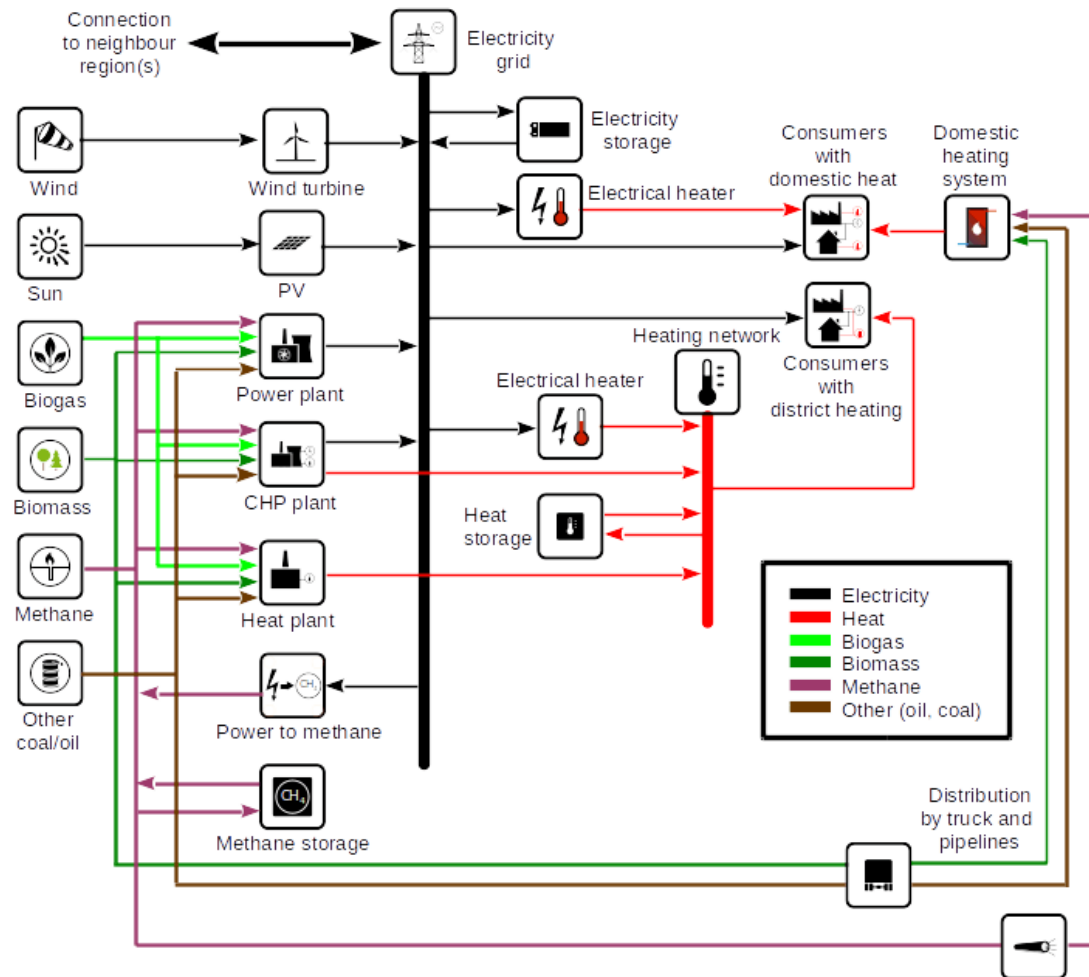
Getting started - Examples

- `test_installation`: Test solph and solver
- `storage_investment`: Basic usage, investment
- `simple_dispatch`: Basic usage, chp
- `csv_reader_investment`: csv-read, investment
- `csv_reader_dispatch`: csv-reader, basic
- `add_constraints`: adding constraints and components
- `variable_chp`: Modelling an extraction turbine

Getting started - Documentation

- Homepage
 - General information, Newsletter, ...
 - URL: <http://www.oemof.org>
- General Documentation
 - API (docstrings)
 - Installation guide
 - Overall documentation
 - URL: <http://oemof.readthedocs.io>
- Docstrings (source code)
 - Parameter
 - Attributes
 - Constraints, sets and variables

Flexible modelling of arbitrary systems within a single framework



Conclusion – oemof is..

- **Cross-Sectoral** – Include and link the heat, power and mobility sector
- **Multiregional** – Flexibly connect multiple regions
- **Time-step-flexible** – Choose the temporal resolution mostly suited for you application
- **Modular** – Choose from various python packages (libraries) with well defined interfaces for modelling and optimisation
- **Open Source and community driven** – It's free, transparent and well documented
- **Versatile** – Create applications and adapt components to your scope and purpose

Thanks for your attention. Questions?

Cord Kaldemeyer, M.Eng

Research Assistant, ZNES Flensburg

PhD Student, Europa-Universität Flensburg

cord.kaldemeyer@hs-flensburg.de

+49 (0)461 1260

Official website and contact to all developers:

<http://www.oemof.org/contact>

github repositories:

<http://github.com/oemof>