AMPL for MILP

Alessio Santecchia

Ermanno Lo Cascio

Jordan Holweger

Julia Granacher

Luise Middelhauve

Rafael Amorim Leandro de Castro Amoedo

Xiang Li

Prof. François Maréchal

EPFL

IPESE
Industrial Process and Energy Systems Engineering

MILP problem

A **Mixed-Integer Linear Programming** problem is a mathematical optimisation program in which some of the variables are constrained to be integers, while others are allowed to be continuous. The necessary condition for a mathematical problem to be part of the MILP class is that both the objective functions and the constraints have to be linear

$$
\begin{array}{rl}
\min & \mathbf{c}^T\mathbf{x} \\
\text{subject to} & \mathbf{A}\mathbf{x}\,\{\geq,=,\leq\}\,\mathbf{b} \\
 & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \\
 & \mathbf{x}_i \in \mathbb{Z} \quad \forall i \in \mathscr{S}
\end{array}
$$

where

$\mathbf{x}$	$\in \mathbb{R}^n$	is the vector of structural variables
$\mathbf{A}$	$\in \mathbb{Q}^{m\times n}$	is the matrix of technological coefficients
$\mathbf{c}$	$\in \mathbb{Q}^n$	is the vector of objective function coefficients
$\mathbf{b}$	$\in \mathbb{Q}^m$	is the vector of constraints right-hand sides (RHS)
$\mathbf{l}$	$\in \mathbb{Q}^n$	is the vector of lower bounds on variables
$\mathbf{u}$	$\in \mathbb{Q}^n$	is the vector of upper bounds on variables
$\mathscr{S}$		is a nonempty subset of the set $\{1\ldots,n\}$ of indices

A Mathematical Programming Language (AMPL)

High-level algebraic modelling language to build and solve complex problems for large-scale mathematical computing

1. Natural syntax (concise and easy to read)
2. Good scripting language to process data
    - Looping
    - If…then…else… command
3. Can handle continuous and integer variables
4. Many available solvers
    - Linear and convex quadratic (e.g. CPLEX, Gurobi,…)
    - Nonlinear (e.g. MINOS, SNOPT, CONOPT,…)

https://ampl.com/resources/the-ampl-book/

AMPL: modelling

Modelling steps

Mathematical formulation of MILP problems

Identify the parameters and variables

Formulate the objective function

Add constraints

AMPL

Solver
e.g. Gurobi

Results

AMPL: types of file

1. .mod
   Main file where the entire model is implemented. It contains the definition of the problem variables, constraints and objective function. All the laded data are processed in this file.

2. .dat
   It contains all the input data to the problem.

3. .run
   Commands file where the model and the data files are loaded. The major run options are defined here (e.g. selection of the solver) together with the *solve* command and the variables that have to be shown after the convergence is reached.

AMPL: useful commands

Parameter declaration

A parameter is a quantity not subjected to changes during the optimisation. It can be used only after it is declared.

```
param t;
param t := 1;
```

The operator *default* initialises the parameter while allowing its value to be overridden by a data statement or changed by subsequent assignments.

```
param a default 5;
param b := 3;
param c := a+b;
```

A parameter can also be a vector:

```
param C;
param cost{1..C};  #numeric index from 1 to C
```

AMPL: useful commands

It is possible to refer to a specific element of the vector by doing:

```
...cost[x]....     #where x is a numeric index
```

If a parameter has to be editable, instead of being introduced by :=, it has to be declared with the command *default* and later changed by using *let*

```
param c default 100;
....
let c := 200;
```

Sets

This is a fundamental component of an AMPL model. Almost all of the parameters, variables and constraints are indexed over sets.

```
set Technologies default {};
```

AMPL: useful commands

A set can be an unordered collection of character strings.

```
set Technologies = {“boiler”, “PV”, “HP”};
```

or

```
set Technologies := boiler PV HP;
```

Set members can also be numbers or even a mixture of numbers and strings. The *by* clause can be used to specify the spacing:

```
set years = 1990..2020 by 5;
```

It is also possible to construct new sets from existing ones (A and B) by using the following operators:

```
A union B      # union: in either A or B
A inter B      # intersection: in both A and B
A diff B       # difference: in A but not B
A symdiff B    # symmetric difference: in A or B but not both
```

AMPL: useful commands

Two other AMPL operators, *in* and *within*, are used to test the membership of sets:

```
"boiler" in Technologies
# true only if "boiler" is a member of the set Technologies

{"boiler", "PV"} within Technologies
# true only if {"boiler", "PV"} is a subset of the set Technologies
```

A parameter can be indexed over multiple sets.

```
param Qheating{b in Buildings, t in Time} default 0;
```

It is possible to assign values to the parameters indexed over the two sets Buildings and Time. To do so, one should specify the indexes followed by the corresponding values. To improve **readability** it is convenient to define the parameter as follows:

AMPL: useful commands

IPESE
Industrial Process
and Energy Systems
Engineering

```
param Qheating :=
                    "BC"  1   value1
                    "BC"  2   value2
                    "CE"  1   value3
                    "CE"  2   value4;
```

An element of the multi-dimensional parameter can be accessed by using the square brackets:

```
... Qheating["BC", 1] ...
```

It is also possible to read values from a .txt file and assign them to a parameter. The file must be unformatted in the sense that it contains nothing except the values to be read. The values in the file are assigned to the entries in the list in the order that they appear.

```
param time_steps := 5;
read {t in 1..time_steps} t_op[t] < values.txt;
```

From a file values.txt containing only:

```
100 200 450 135 700
```

AMPL: useful commands

Variables declaration

Variables represent unknown quantities whose values is calculated by the solver. As the parameter they have to be declared before being used.

```
var x;
var mult{Utilities};
```

Zero-one (or **binary**) variables for modelling logical conditions:

```
var use{Utilities} binary;
```

The lower bound has to be specified in order to avoid unfeasible evaluations and facilitate the resolution of the optimisation problem:

```
var mult{Utilities}>=0;
```

Or even the higher bound:

```
var mult{Utilities}>=0, <=10000;
```

AMPL: useful commands

IPESE
Industrial Process
and Energy Systems
Engineering

Objective function

It is the function that has to be maximise or minimise. It is declared with the command *maximize* or *minimize* followed by the name of the objective and by a mathematical expression of sets, parameters and variables previously defined.

```
minimize Total_cost : Inv_cost + Op_cost;
```

The expression can be even more complicated:

```
minimize Total_cost : sum{tc in Technologies} (cinv1[tc] * use[tc] + cinv2[tc] * mult[tc]) +
                      sum {u in Utilities, t in Time} (cop1[u] * use_t[u,t] + cop2[u] * mult_t[u,t]) * top[t];
```

However, it is highly recommended to avoid a complicated expression of the objective function since it considerably slows down the resolution of the problem!!

AMPL: useful commands

Mathematical constraints

The simplest kind of constraint declaration begins with the keywords *subject to*, a name, and a colon. Even the subject to is optional: AMPL assumes that any declaration not beginning with a keyword is a constraint. Following the colon is an algebraic description of the constraint, in terms of previously defined sets, parameters and variables.

```
subject to name:
    x <= 4;
```

Obs: The name of a constraint, like the name of an objective, is not used anywhere else in an algebraic model!!.

Most of the constraints in large programming models are defined as indexed collections, by giving an indexing expression after the constraint name.

```
subject to size_cstr1{u in Utilities, t in Time}:
    Fmin[u]*use_t[u,t] <= mult_t[u, t];
```

AMPL: useful commands

Iterated operators (sum, prod, min and max)

It is possible to iterate operations over sets. In particular, most large-scale linear programming models contain iterated summations (*sum*):

```
InvCost = sum{tc in Technologies} (cinv1[tc] * use[tc] + cinv2[tc] * mult[tc]);
```

The arithmetic expression is evaluated once for each member of the set *Technologies*, and all the resulting values are added.

Other iterated arithmetic operators are *prod* for multiplication, *min* for minimum, and *max* for maximum. For example one could write:

```
...max{t in Technologies} mult[t]...
```

to obtain the greatest mult among those attributed to each member *t* of the set *Technologies*

AMPL: useful commands

IPESE
Industrial Process
and Energy Systems
Engineering

Conditional operator if-then-else

It returns one of two different arithmetic values depending on whether the logical expression is true or false. For example:

```
param Qheatingdemand{h in HeatingLevel, t in Time} :=
    if h == 'MediumT' then
        sum{b in MediumTempBuildings} Qheating[b,t]
    else
        sum{b in LowTempBuildings} Qheating[b,t];
```

Qheatingdemand is a bi-dimensional parameter defined for each *HeatingLevel h* and *Time t*. If the *HeatingLevel h* is equal to '*MediumT*' then *Qheatingdemand* is calculated, for each *Time t* as the sum of all the demands *Qheating* of the buildings *b* belonging to the *MediumTempBuildings* set. The *LowTempBuildings* set is used otherwise.

AMPL: useful commands

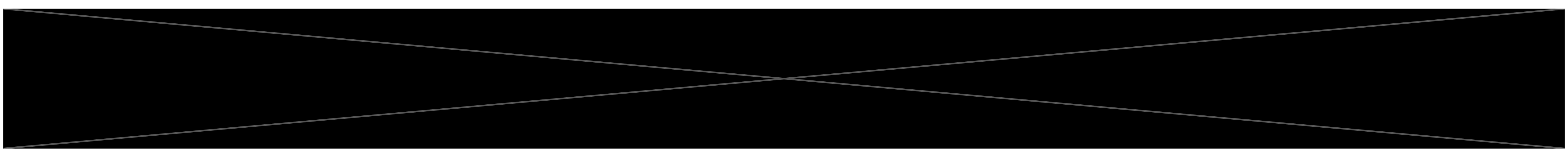
Commands file (.run)

The commands file with extension .run is often used to load both the model and the corresponding data. This is done by including the following lines:

```
model model_name.mod;
data datafile_name.dat;
```

Additional options, such as the used solver, are declared with the command *option:*

```
option solver gurobi;
```

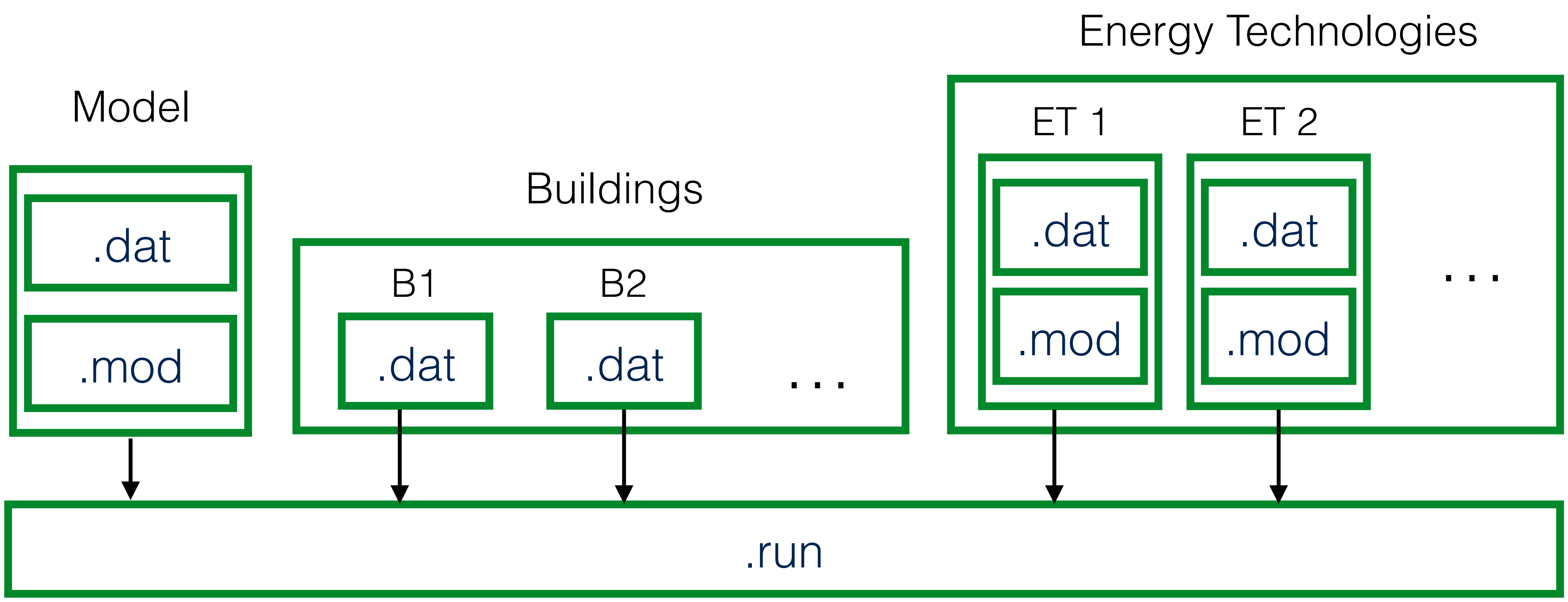


Finally, the commands *solve* and *display* are used to compute the model and output to the command window/terminal the desired quantities:

```
solve;
display Qheating;
```

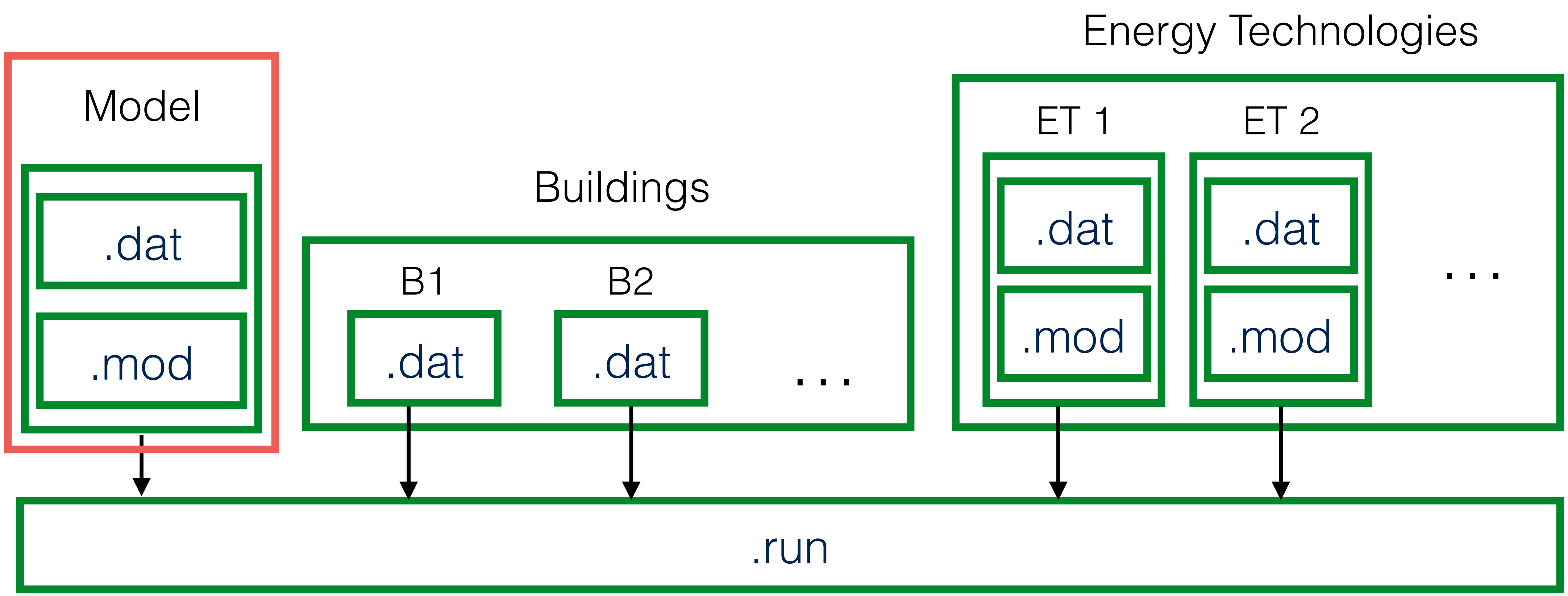
Project structure

- All the .mod and .dat file are finally loaded into the commands file (.run)
- Additional running options can be specified here:
  1. Solver
  2. Maximum number of iterations
- The display command is used to output the optimised variables



Project structure

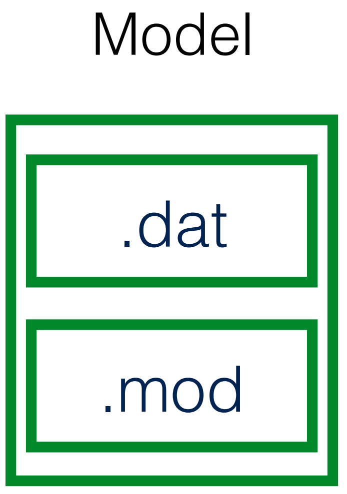
- All the .mod and .dat file are finally loaded into the commands file (.run)
- Additional running options can be specified here:
  1. Solver
  2. Maximum number of iterations
- The display command is used to output the optimised variables



Project files: .mod and .dat

Main model file **moes2020.mod**

- Sets
- Parameters
- Variables
- Constraints
  - Utility sizing constraints
  - Heating balances (MT and LT)
  - Resource balances
  - Electricity balances
- Objective function

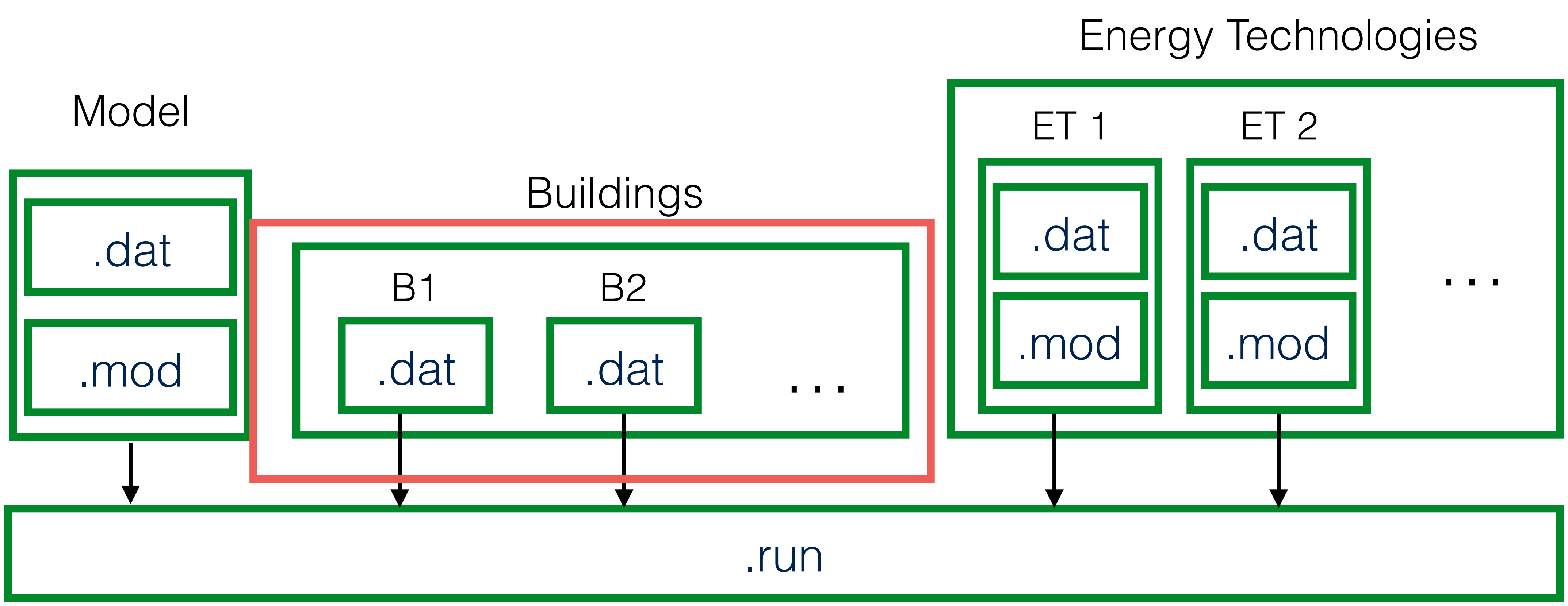


Main data file **moes2020.dat**

- Define the composition of each set
- Define the generic parameters of the model (top, irradiation, Text, c_spec, …)

Project structure

- All the .mod and .dat file are finally loaded into the commands file (.run)
- Additional running options can be specified here:
  1. Solver
  2. Maximum number of iterations
- The display command is used to output the optimised variables

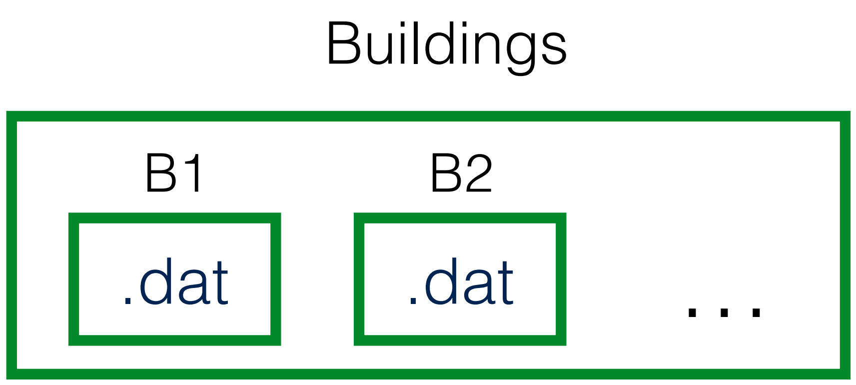


Project files: Buildings

Each building is loaded into the model using a .dat file only

1. Add the building to the corresponding set (*MediumTempBuild* or *LowTempBuild*)

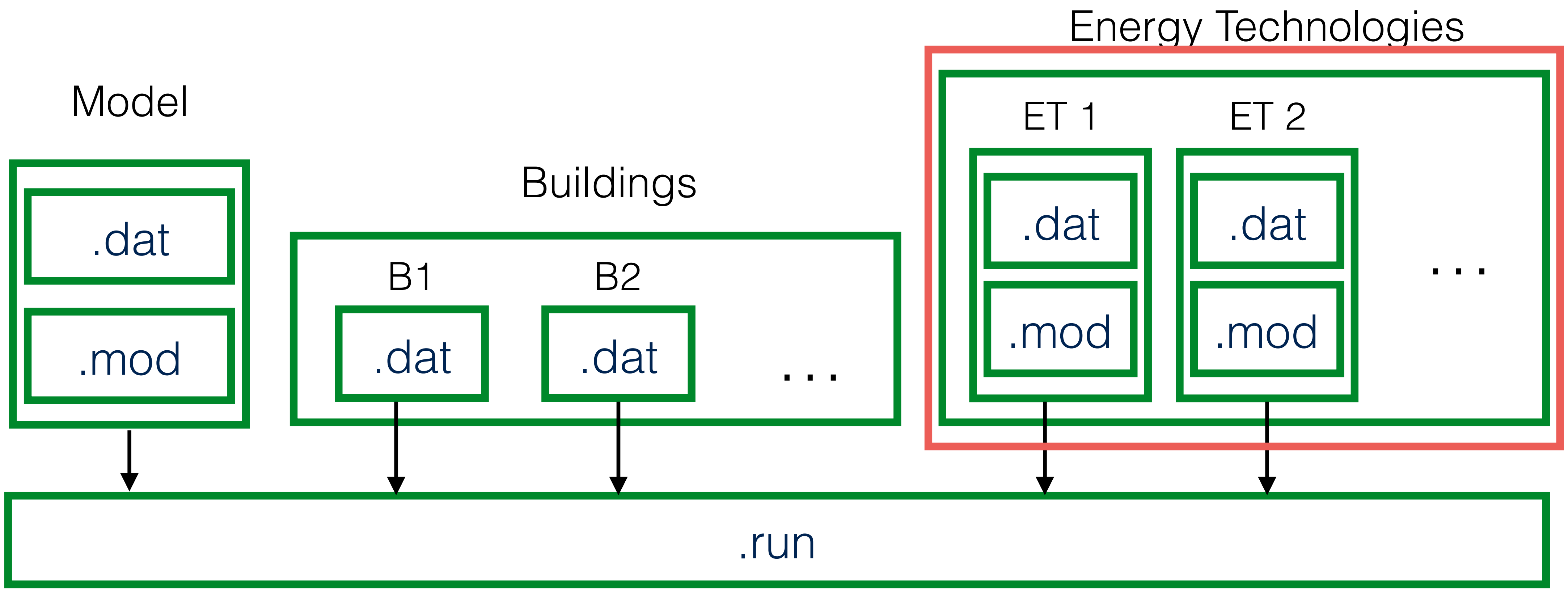
2. Define some building-specific parameters:
   - Floor area
   - kth and ksun
   - …



A .mod file could be used to model the refurbishment
- Additional binary variable (refurbishment option)
- Equation defining the new heating demand of the building
- Equation for the cost of refurbishment

Project structure

- All the .mod and .dat file are finally loaded into the commands file (.run)
- Additional running options can be specified here:
  1. Solver
  2. Maximum number of iterations
- The display command is used to output the optimised variables

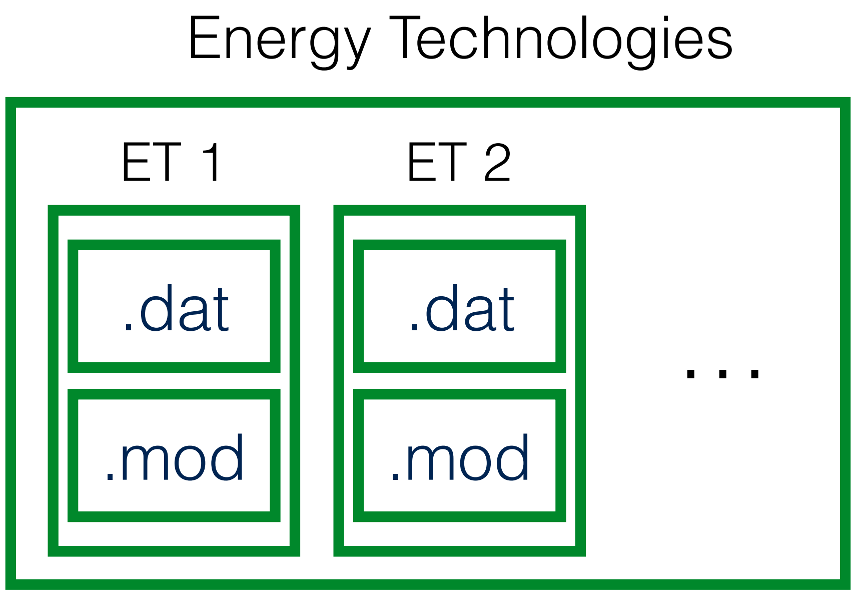


Project files: Energy Technologies

Each energy technology is introduced using the model-data structure

The data file contains:

- Definition of Maximum and minimum scaling factors
- Efficiencies
- Supply temperature
- …

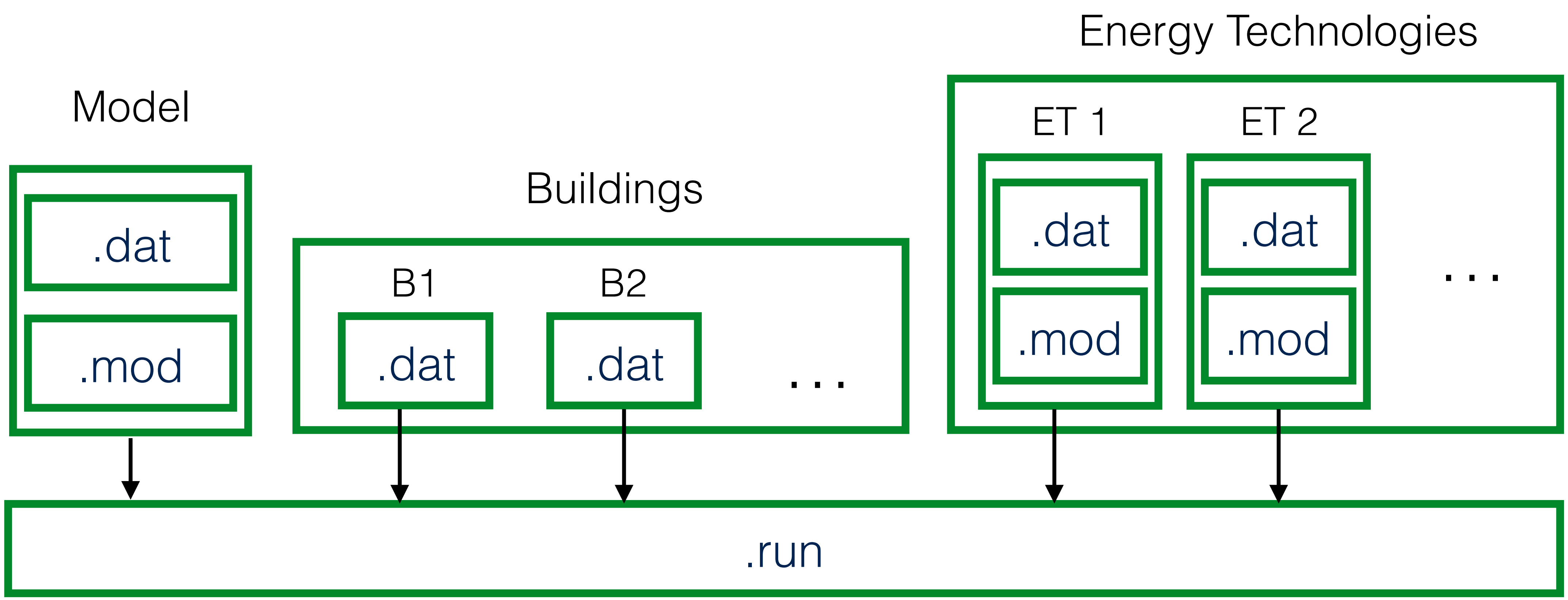


The mod file contains the modelling equations:

- Supplied heat
- Resource consumption (Natural gas, electricity, biogas, …)
- …

Project structure

- All the .mod and .dat file are finally loaded into the commands file (.run)
- Additional running options can be specified here:
  1. Solver
  2. Maximum number of iterations
- The display command is used to output the optimised variables



Overall picture of the model

Sets are fundamental to model large-scale optimisation problems

They are used to group elements of the same type and repeat operations

In the project, sets are defined for the main elements:

- Time steps
- Buildings
- Technologies → Utilities
- Grids (to buy resources) → Utilities
- Resource layers (Natural gas, electricity, biogas)

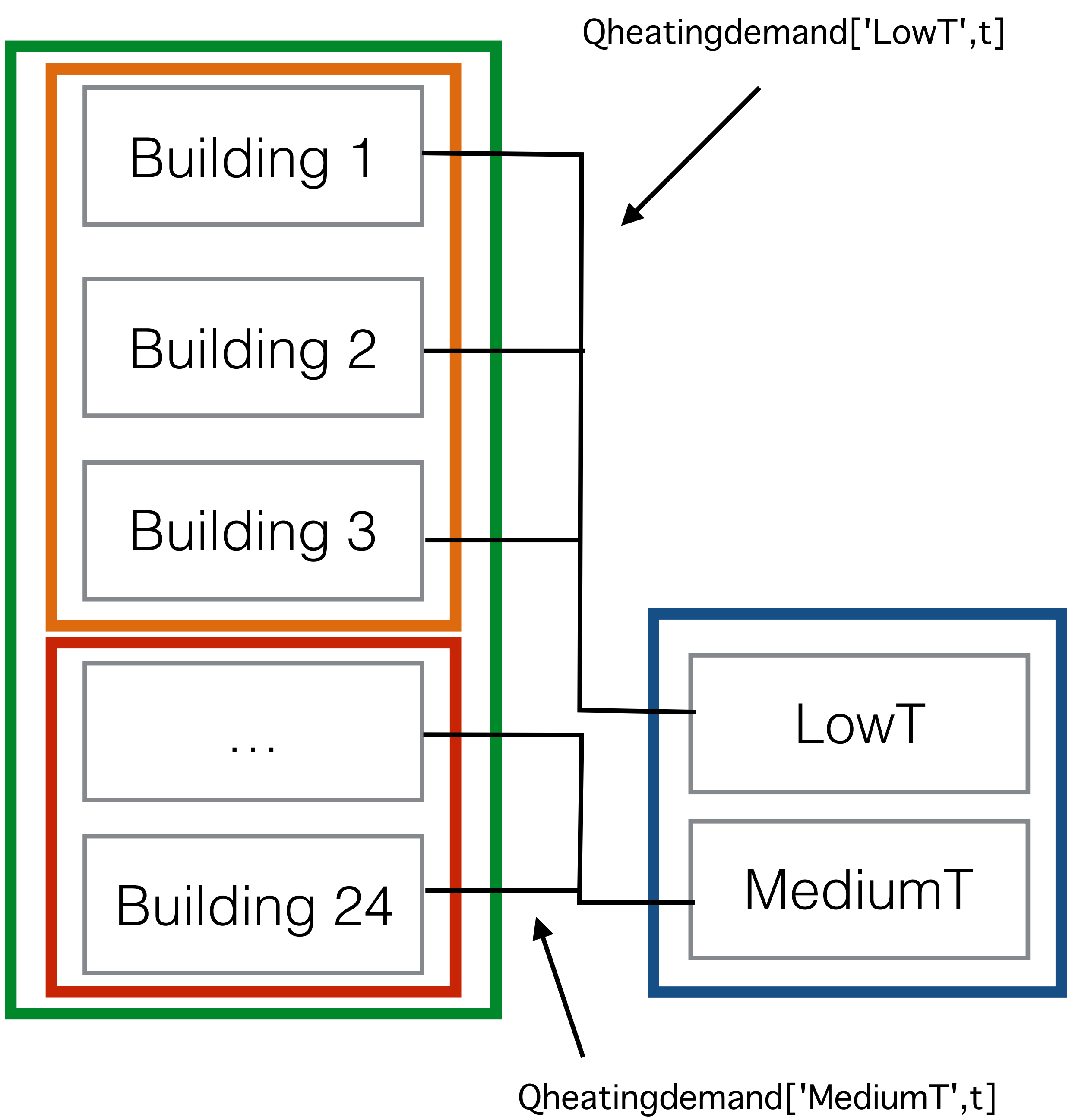
Additional sets are used to group elements because of common features

- Low and Medium temperature buildings
- Type of utility (heating, electricity producers and consumers)
- Layer of the utility

Some sets are strictly related to each other since they share same elements!

Demand-side sets

IPESE
Industrial Process
and Energy Systems
Engineering



Buildings
- List of buildings

LowTempBuildings
- Buildings heated by low T loop

MediumTempBuildings
- Buildings heated by medium T loop

HeatingLevel
- Two options (low/medium)
- Classify buildings
- Each utility can provide heat to both loops

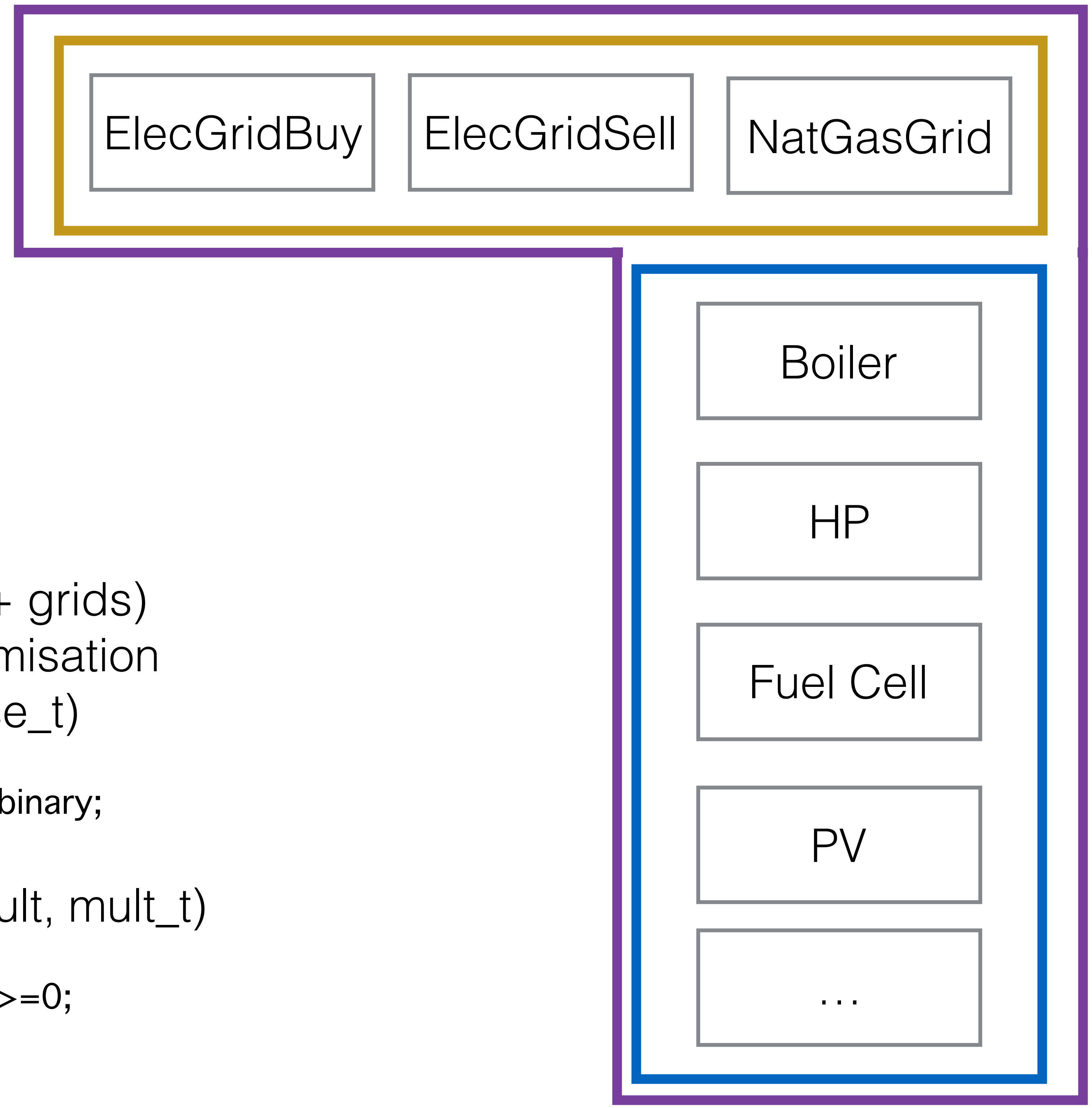
mult_heating_t[u,t,'LowT']
mult_heating_t[u,t,'MediumT']

Utilities set

Technologies
- List of ET

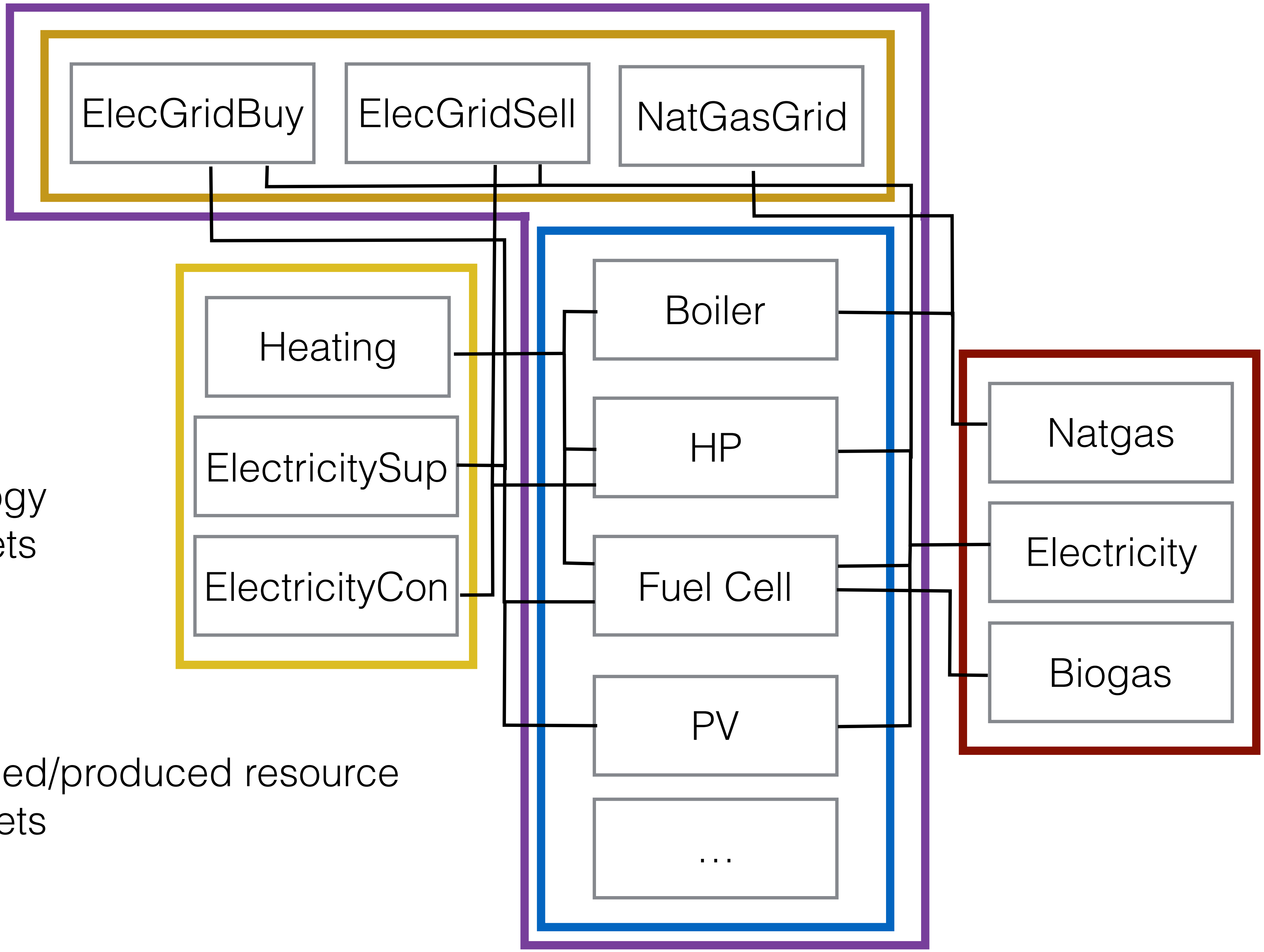
Grids
- Grid units to buy resources

Utilities
- List of utilities (technologies + grids)
- Utilities are the object of optimisation
  1. Binary variables (use, use_t)
     ```
     var use{Utilities} binary;
     var use_t{Utilities, Time} binary;
     ```
  2. Continuous variabels (mult, mult_t)
     ```
     var mult{Utilities}>=0;
     var mult_t{Utilities, Time}>=0;
     ```



Layers and other sets

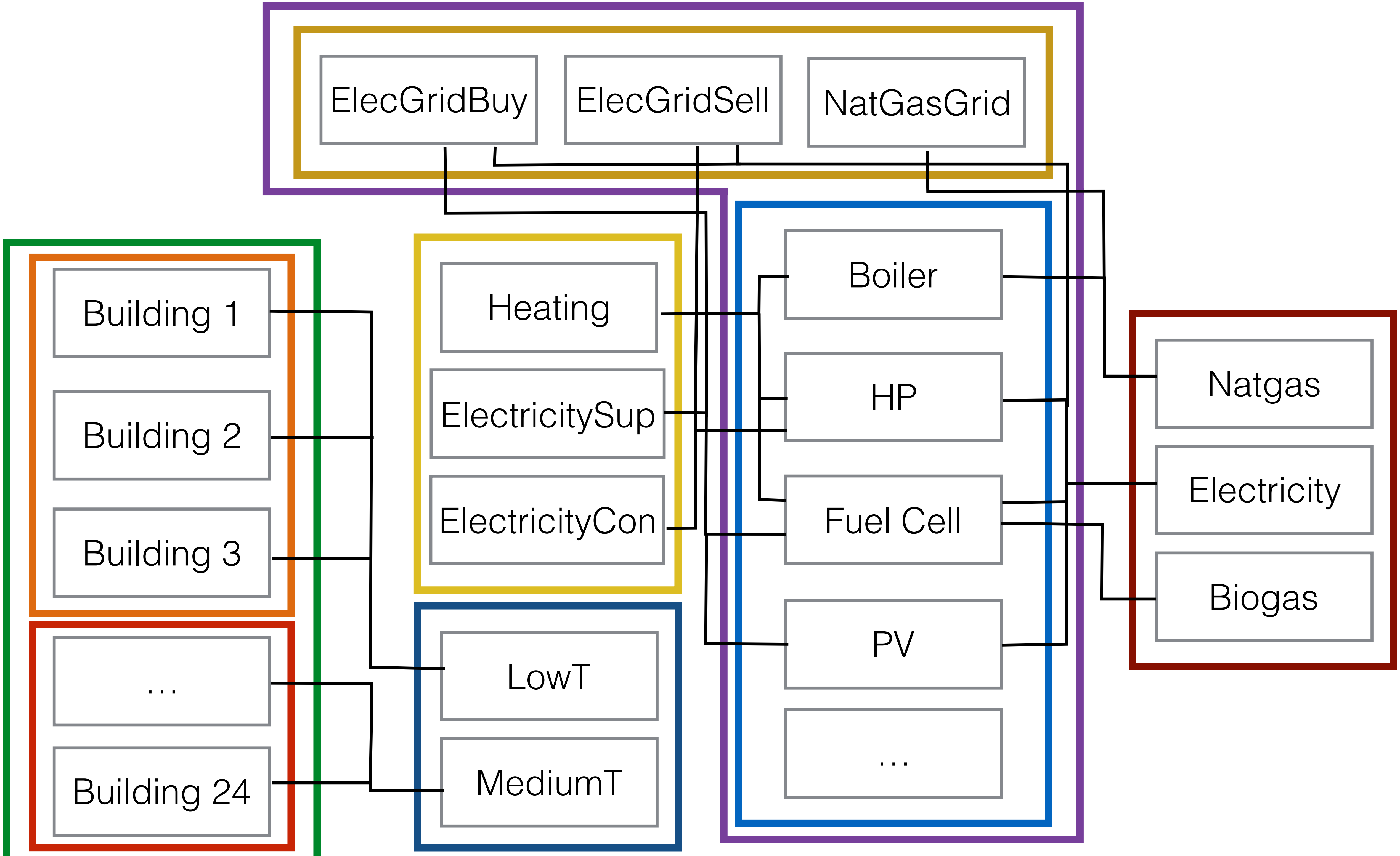
Other useful sets…



UtilityType
- Type of technology
- List of ET sub-sets

Layer
- Type of consumed/produced resource
- List of ET sub-sets

Overall structure of the sets



Constraints: heating balances

Heating balances:

1. LT_balance{t in Time}

   Low temperature demand = sum over the utility set of the heat supplied by each utility of type heating (**mult_heating_t[u,t,'LowT']**)

2. MT_balance{t in Time}

   Medium temperature demand = sum over the utility set of the heat supplied by each utility of type heating (**mult_heating_t[u,t,'MediumT']**)

Observe that for each utility u and time step t:

mult_t[u,t] = mult_heating_t[u,t,'LowT'] + mult_heating_t[u,t,'MediumT']

Constraints: resource balance

Resource balance:

1. Sum of the **FlowInUnit** of the utilities over each layer l = Sum of the **FlowOutUnit** of the utilities over the same layer l (except for electricity layer)

where…

```
FlowInUnit[l, u, t] = mult_t[u,t] * Flowin[l,u];
FlowOutUnit[l, u, t] = mult_t[u,t] * Flowout[l,u];
```

Observe that for each utility u and layer l:

Flowin[l,u] and Flowout[l,u] are calculated using the reference size

Constraints: electricity balance

Electricity is considered to be a resource as well…

Electricity balance constraint:

1. Total electricity demand + Sum of the **FlowInUnit** of the utilities over the sub-set ElectricityCons = Sum of the **FlowOutUnit** of the utilities over the sub-set ElectricityProd

Observe that:

UtilitiesOfType['ElectricityCons'] and UtilitiesOfType['ElectricityProd'] are two sets indexed over the elements of the set UtilityType

Objective function

The objective function is the **Total cost** (annual) defined as the same between:

1. Operating cost

```
OpCost = sum {u in Utilities, t in Time} (cop1[u] * use_t[u,t] + cop2[u] * mult_t[u,t]) * top[t];
```

2. Annualised investment cost

```
InvCost = sum{tc in Technologies} (cinv1[tc] * use[tc] + cinv2[tc] * mult[tc]);
```

Where:

cop1[u] is the fixed operating cost of the utility u [CHF/h]

cinv1[tc] is the fixed investment cost of the technology tc [CHF/y]

cop2[u] is the variable operating cost of the utility u [CHF/h]

cinv2[tc] is the variable investment cost of the technology tc [CHF/y]

Thanks for your attention and
ENJOY!

EPFL

IPESE
Industrial Process
and Energy Systems
Engineering