Spine Model: an Open-source and Flexible Energy-system Model Generator

Kris Poncelet (kris.poncelet@kuleuven.be)
Post-doctoral researcher, KU Leuven, Belgium

22nd Oct. 2019
INFORMS Annual Meeting 2019 Seattle
TE85 – Innovative approaches to Integrated Multisector Modeling
Spine project

• EU Horizon 2020 project
  • 10/2017 – 10/2021

• Main goals:
  1) Develop an open-source energy **modeling toolbox**
  2) Develop an open-source energy-system optimization model generator
Context/Motivation

- Increasing complexity of the energy system => increasing need to link or integrate different models:
  - Across time (e.g., investment planning -> unit commitment)
  - Across energy sectors (e.g., electricity – gas – heat – transport)

- Vast majority of models developed with a specific purpose and in isolation
  - Different models have different data structures
  - Different models use different (naming) conventions
  - Different models are implemented in different software languages (e.g., GAMS, python, Julia/JuMP)

- Consequences:
  - No economy of scale to build a good modeling toolbox
  - Difficult to link different models
  - Difficult to understand and to learn to work with different models
  - Difficult to share data
Problem independent data store (T2.2)

Problem independent GUIs (T2.4)

Original data sources

Data conversion tools (WP4)

Spine API (T2.2, 2.5)

Interfaces (T2.5)

Models

Spine generic energy system model (WP3)

Other models

Python (to do)

Julia (to do)

GAMS (to do)
Why a problem independent data store?

- Creates independence between problem structure, GUI, interfaces
  - => GUI does not need to be tailored to a specific problem
  - => Interfaces (e.g., to GAMS, Python, Julia) do not need to be tailored to a specific problem
  - => different models can make use of the same GUI and/or Interfaces
    - => economies of scale => nice interface with less time spent
    - => easier to learn to work with different models
    - => easier to link different models
Entity-Attribute-Value with Classes and Relationships
Design of a problem independent data store

- Database design should be able to contain data for different type of problems
  - E.g., unit commitment, heating systems, etc.
- Entity–attribute–value structure with classes and relationships (EAV/CR).
  - Generic database structure supporting different type of problems
  - flexibility to create new object and relationship classes without touching the design of the data store
  - => allows modelers to create new models or adapt models without the need to adapt GUI, interfaces, etc.
Problem agnostic toolbox (WP2)

Interfaces (T2.5)

Models

Spine generic energy system model (WP3)

Original data sources

Data conversion tools (WP4)

Problem independent data store (T2.2)

Problem independent GUIs (T2.4)

API

Spine API (T2.2,2.5)

Python

Julia

JUMP

GAMS

Other models
SpineInterface.jl

Modify data store structure:
- Add new table field
- Add data
- Modify interface

New Data Parameter
Required

Data available in model

Problem independent
data store

Interface (e.g., SpineInterface.jl)

Add data

New Data Parameter
Required

Data available in model

We don’t want this!
SpineInterface.jl

• What it does?
  • Converts Spine data store object classes, relationship classes and parameters into Julia functions
using SpineInterface

db_url_in = "sqlite:///C:/Users/u0093836/repos/git/toolbox"

using_spinedb(db_url_in, SpineInterface; upgrade=true)

for u in Thermal_power_plant()
    println("$(u)")
end

CoalPlantA
CoalPlantB
GasCCA
GasCCB
GasOC1
NuclearPlant
using SpineInterface

db_url_in = "sqlite:///C:/Users/u0093836/repos/git/toolbox"

using_spinedb(db_url_in, SpineInterface; upgrade=true)

for u in Thermal_power_plant()
    println("Efficiency of $(u) = $(Efficiency(Thermal_power_plant = u))%"")
end

Efficiency of CoalPlantA = 40%
Efficiency of CoalPlantB = 45%
Efficiency of GasCCA = 60%
Efficiency of GasCCB = 58%
Efficiency of GasOC1 = 45%
Efficiency of NuclearPlant = 36%
using SpineInterface

db_url_in = "sqlite:///C:/Users/0093836/repos/git/toolbox"
using_spinedb(db_url_in, SpineInterface; upgrade=true)

for (l,n) in Line__Nodes()
    println("$(l), $(n)")
end
SpineInterface.jl

• What it does?
  • Converts Spine data store object classes, relationship classes and parameters into Julia functions

• Benefits:
  • Convenience of developing/extending the model
  • Transparency/readability of model code
  • Useful for any Julia model that uses Spine data store

• Uses Spinedatabase API and Julia metaprogramming
Problem agnostic toolbox (WP2)

Original data sources

Data conversion tools (WP4)

Problem independent GUIs (T2.4)

Problem independent data store (T2.2)

API

Spine API (T2.2,2.5)

Interfaces (T2.5)

Spine generic energy system model (WP3)

Models

Other models

GAMS (to do)
Spine Model: design goals and approach

- **Open**
  - Github
  - Use open-source software: Julia

- **Flexible**
  - One model generator for wide range of applications:
    - Long-term energy system optimization ~TIMES
    - Detailed UC models ~PLEXOS
    - Hydro scheduling
    - Heat system optimization with building heating physics
      - etc.
  - Easy addition of new parameters/constraints(/entities)

- **Fast**
  - Julia
  - Efficient formulations
  - Parallelization/decomposition techniques

- Problem independent formulation
  - Generically defined constraints + specific constraints
  - Flexible temporal and geographical structure

- Generic data store + SpineInterface.jl
Problem agnostic system representation

Generic object classes

Generic constraints

Example: Describing the operation of an arbitrary unit (arbitrary number of inputs and outputs) calls for generic equations and parameter names

parameter “efficiency” => parameter “fix_ratio_out_in_{u,c_{out},c_{in}}”

Specific constraints:

Specific, but commodity agnostic

E.g., ramping constraints (parameter ramp_rate_up_{u,c})
Flexible temporal representation

The user can decide the temporal resolution for each individual flow going in/out of a unit/connection

- Example 1: focus region in high-level of detail, other regions in lower level of detail

- Example 2: higher temporal resolution close to the start of the dispatching period, lower resolution later in the modeled horizon (e.g., within a rolling horizon dispatch model)

- Example 3: coal-fired power plant with electricity out flow modeled at a high resolution and coal inflow modeled at a low resolution
A1 Irish dispatch study with power flows

- The model includes three control areas, several units, one storage, various fuels
- It is a rolling horizon unit commitment model
A3 District heating study of Stockholm

- The model includes a dozen units of different types for example CHP, electric boiler and back-pressure steam turbine

- Two of these units can work on different modes

- It is a rolling horizon unit commitment model
A5 Hydro power study with river systems

- The case study aims at modelling the Skellefte river which has 15 stations during a week.
- Each station consists of a power plant and a reservoir.
- Delays in between stations are modelled and vary depending both on the station and on whether the water is discharged or spilled.
Further reading/exploration

Spine Toolbox
https://github.com/Spine-project/Spine-Toolbox

Spine database API
https://github.com/Spine-project/Spine-Database-API

SpineInterface.jl
https://github.com/Spine-project/SpineInterface.jl

SpineModel
https://github.com/Spine-project/Spine-Model
This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement N. 774629.
Problem-agnostic system representation

- **Coal plant**
  - Capacity: 800 MW
  - Efficiency: 40%
  - (Technical constraints)

- **CCGT**
  - Capacity: 400 MW
  - Efficiency: 55%
  - (Technical constraints)

- **Import Coal**
  - 12 €/MWh
  - BelgiumCoal

- **Import Gas**
  - 25 €/MWh
  - BelgiumGas

- **LeuvenElectricity**
  - Electricity

- **AntwerpElectricity**
  - Electricity

- **Electricity Line 1**
  - BelgiumHeat

- **Electricity Line 2**
  - Heat

- **Electricity Line 3**
  - BrusselsElectricity
Generic constraints to represent (energy) conversion and (policy) constraints

- Example for units:
  - ratios between commodity flows:
    - Between output and input flows, multiple outflows or multiple inflows
    - Upper, lower and fixed bounds
  - absolute bounds on the commodity flows
    - Instantaneous
    - Cumulative
    - Upper, lower and fixed bounds
  - relative bounds on commodity flows:
    - Instantaneous
    - Cumulative
    - Upper, lower and fixed bounds
Spine model formulation – flexibility of the grouping concept

- Users can define node/unit/connection/commodity groups
- Parameters and equations are set-up to be used with groups
  - E.g., absolute bound on commodity flow (parameter $\text{max\_out\_flow\_bound}_{ug,cg,t}$) -> Can specify bound for one/multiple commodities, single or multiple units.

\[
\sum_{(c,n,u)\in \text{OutputCommoditiesNodesUnits: } c\in cg, u\in ug} v^\text{Flow}_{c,n,u,\text{out},t} \leq p^\text{MaxOutFlowBound}_{ug,cg,t}
\]

\[
\forall (cg \in \text{CommodityGroups, } ug \in \text{UnitGroups, } t \in \text{Timesteps}) : \\
p^\text{MaxOutFlowBound}_{ug,cg,t} \text{ is defined}
\]

- => Generic constraints offer a lot of functionality without needing to add or modify the equations
  - E.g., a single constraint/parameter in the Julia code can be used for zonal reserve requirement, national RES targets, EU emission targets, nodal inertia constraints, etc.