CESI Multidisciplinary Model Integration Environment

Andreas Grothey*, Ken McKinnon, Wei Sun

School of Mathematics
The University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ
United Kingdom

August 1, 2019

Abstract

1 Introduction

A Model consists of

- **Sets**
  These are the variable (or non-variable) dimensions of the entities in the model. For example: Set of Buses, Set of Generators, Set of Timesteps, Set of Lines.
  Sets are free for the modeller to choose.

- **Entities**
  These are model components such as Lines, Buses, Generators, Demands. They can occur singly (rare) or index over one or more sets (i.e. Demands[BUSES, TIMES]).
  Entities must be a subset of the predefined entities that are declared in the Modelling System. Note that a model may have other internal (state) variables/paramaters/etc that do not need to be classified as one of the predefined entities. However these can then not be part of the Model interface (i.e. not exposed to the outside world)

- **Fields/Properties**
  Every entity in the model has a set of properties/fields. For example a Line has *Real Flow From Source, Real Flow from Target, Reactive Flow, Reactance, Resistance, Flow Limit.* The properties must be a subset of a predefined list of properties of the given entity. The properties can have various functions

*Email: A.Grothey@ed.ac.uk, Homepage: http://maths.ed.ac.uk/~agr/

in the model: they can be data (that have to be specified with a dataset), they can be input parameters (i.e set by the caller of the model, possibly with the aim of obtaining sensitivities with respect to this parameter), they can be output of the model or they can be variables that the model should optimize over. The model needs to specify:

- which Sets, Entities, Fields it knows about
- What is the intended (or possible) purpose of each field (data, parameter, ouput, variable).

The model needs to specify a method that checks consistency of the given context (i.e. enough data is given for the model to make sense). The correct number of degrees of freedom is left.

Data for the model will be specified by 2d csv-tables. They have AMPL type format: the rows correspond to sets and the columns to fields. The first column is the members of the set for which data is given, the first row is the names of the fields/properties that are given in this table.

The first row can be taken to be the definition of a set for the model. Or it can just be a subset of the set (that is specified elsewhere in the data). The fields must be a subset of all the fields that are defined for the given entity.

2 Interface Layout

2.1 Model Interface API

`solve()`	run the model (assuming all data/input set)
`bool checkConsistency()`	check that all input data is specified
`Set[] getSets()`	sets are anything that input/ouput can be indexed over. Sets themselves have to be input.
`int getNrSets()`	Not sure if number of sets is a different method
`Components getInput()`	Returns a hierarchy of entities that are used as input to the model. These can be data or parameters
`Components getOutput()`	Returns a hierarchy of entities that can be used as output of the model. These can be direct output or (exposed) state variables.
`double getValue()`	
`getDerivative/getSensitivity()`	
`double *getValue(Entity)`	how is the value returned (just an array of double?) → see below for possible method signatures

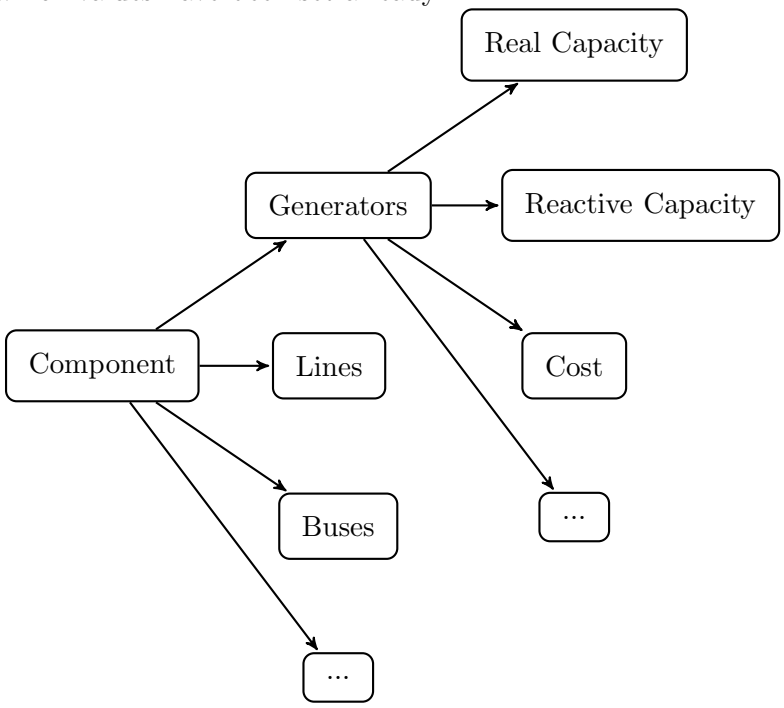
`Components` is an object that consist of

- Entities (and for each entity, which set(s) it is iterated over)

- Fields of each entity

The idea is that both *Entities* and *Fields of Entities* are a subset of a (hardcoded) list of possible entities (and their fields) that we think could be part of an energy model. These names can be used for example as identifiers/annotation of data which would enable automatic reading in of data tables into models (without needing to do a translation like "read in column 6 of data into field 8 of model").

It would be good if there is a `bool` and `double` version of Components. The `bool` version could be used to indicate which Entities/Fields are present in a Model, the `double` version could carry values. The `bool` version could also indicate which values should be set (from data), should be returned from the Model, or keep track of which values have been set already.



`readValuesFromData(BoolComponent& blcomp, FileID id)`
: Fills in the values indicated by `blcomp` from the data table `id`

`setValues(BoolComponent& blcomp, DoubleComponent& dblcomp)`
: Sets the values indicated by `blcomp` to the values given in `dblcomp`

`getValues(BoolComponent& blcomp, DoubleComponent& dblcomp)`
: Obtains the (output)-values indicated by `blcomp` in `dblcomp` (after running the mode

`getSensitivity(FieldID fid, BoolComponent& blcomp, DoubleComponent& dblcomp)`
: gets derivative of the (one) entry identified by `fid` with respect to the variables/data