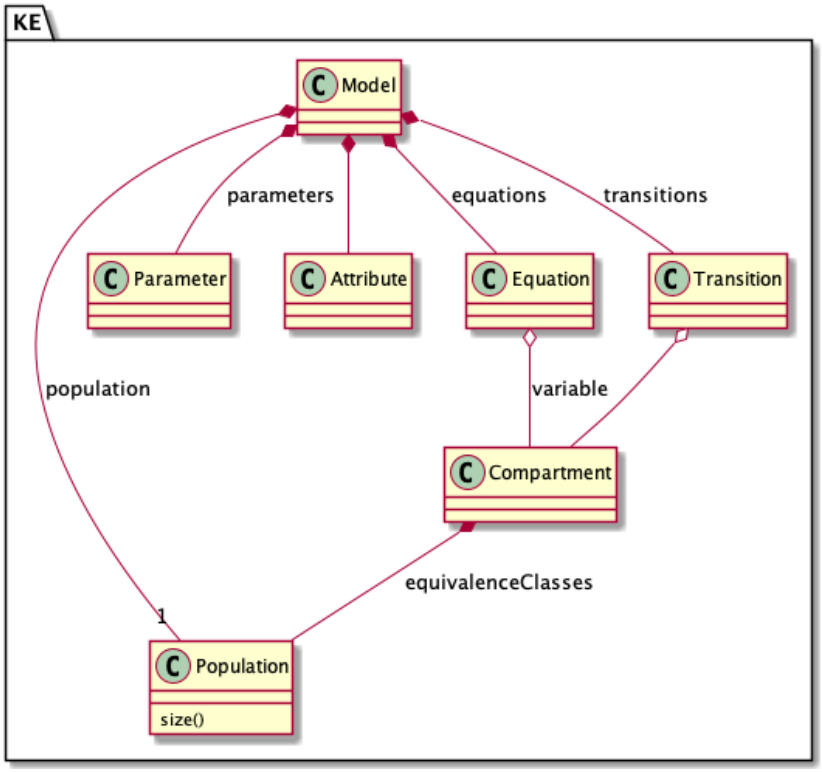
Kendrick Meta-model v3

Serge Stinckwich

January 4, 2021

1 Kendrick Meta-Model Diagram



2 Kendrick Meta-model description

The abstract syntax of Kendrick is based on the mathematical definitions of Kendrick and relies on an object-oriented meta-model. We use the UML graphical modelling language to describe the Kendrick meta-model.

2.1 Model

A Kendrick **model** (instance of KEModel) is defined by a collection of **transitions**, a collection of **parameters**, a collection of **attributes**, a collection of **equations** and one **population**.

In the implementation, equivalence relations are restricted to equalities on individuals' attributes. This guarantees that each equivalence class (the individuals with the same values for a given set of attributes) has a known name: the ordered set of attribute names used in the equality. An attribute is simply a partial function on the population to any domain with equality. The function is assumed total on the compartments it defines. When models are combined, attribute names must stay unique in the whole population (name clashes must be avoided).

2.2 Concern

Do we need to introduce **concern** concept in the meta-model ?

2.3 Population

A Kendrick **population** (instance of KEPopulation) is defined by a collection of **compartments** i.e. of equivalence classes. A population has a method size (number of individuals of that population). The size of the population is the sum of the sizes of each compartment. A new population is empty (size = 0).

2.3.1 Issues to be solved:

- Why compartments are represented as a set of dictionaries with a specific structure (linked to the fact that there is no compartment class).
- Why we can add/remove individuals or have access to the individuals from population ?
- What is diedInList ?

2.4 Compartment

At the moment, we have no class **compartment**, this is bit weird because compartment are represented as Dictionary.

2.4.1 Issues to be solved:

- Represent compartment as object and not Dictionary (issue 89).

2.5 Attribute

An attribute is not defined by a class. Examples of attributes are: #species, #status. A Kendrick **attribute** has a name and a domain.

2.6 Functional rates

Apparently there is no representation of functional rates in the K meta-model. Do we need to introduce it ?

The following concepts are not fondamental to Kendrick, but allow to represents ODEs.

2.7 Equation

A Kendrick **equation** (instance of KEEquation) represent an ODE (Ordinary Differential Equation). It is defined by a **variable** that depends on time

2.7.1 Issues to be solved:

- Do we need equation and transition, because normally they can be generated from each others ?

2.8 Variable

A Kendrick **variable** (instance of KEVariable) represent a mathematical variable in an equation. A variable is defined with a name (symbol).

2.8.1 Issues to be solved:

- variable name is called symbol in the implementation that is not really informative.
- there is no dependency between variables. In order to know the dependency we have to look at the **equation** that contains the variable. A variable should have a list of dependencies.

`Kendrick_Meta-model_description/2020-07-16_17-49-49_ReHab_Pharo.st`

2.9 Parameter

A Kendrick **parameter** (instance of KEParameter) represent parameters in an epidemiological model. A parameter is defined with a name (symbol) and an expression. By default, a Kendrick model got a **N** parameter initialized with the cardinality of the whole population of a model.

2.9.1 Issues to be solved:

- `https://github.com/UMMISCO/kendrick/issues/99`
- This is not really clear why we separated variables and parameters. Apparently parameter are not depending on another variable.

3 Kendrick Workflow Meta-model description

4 Smalltalk implementation

Functional rates are represented as lexical closure in Smalltalk.

4.1 Attribute

Attributes are Smalltalk symbol (immutable String). Domain associated to attributes are defined as Smalltalk symbol also. You add all the attributes with attributes: method

```
model := KEModel new.
model attributes: {(#status -> #(#S #I #R). (#species -> #(#human #bird))}.
```

or add attributes one by one with: addAttribute:value:

```
model := KEModel new.
model addAttribute: #status value: #(#S #I #R).
model addAttribute: #species value: #(#human #bird).
```