Proposal for an extension to the current SED-ML Level 1 Version 2 to cover experiments on logical models encoded in SBML Qual

Draft version 1 | February 9, 2016

Claudine Chaouiya, Instituto Gulbenkian de Ciência, Oeiras, Portugal (chaouiya@igc.gulbenkian.pt)
Pedro T. Monteiro, INESC-ID, Lisbon, Portugal (Pedro.Monteiro@inesc-id.pt)
Aurélien Naldi, DIMNP, University of Montpellier, France (aurelien.naldi@umontpellier.fr)
Dagmar Waltemath, University of Rostock, Germany (dagmar.waltemath@uni-rostock.de)

Summary

Quantitative modeling is used in computational biology to study interaction networks (predominantly regulatory and signalling networks). The SBML Qual package is an extension to the SBML Level 3 standard.
SBML Qual supports the encoding of qualitative models. It defines the structure and syntax for qualitative models that associate discrete levels of activities with entity pools and the transitions between states that describe the processes involved (Chaouiya et al 2013).
SED-ML Level 1 Version 2 already supports different types of simulation experiments on SBML models. However, a few concepts are missing in order to fully support the encoding of models based on the SBML Qual extension. Here is a first proposal for how the missing concepts could be added to the current SED-ML Level 1 Version 2.

Document conventions

We use the same conventions that were already used in the SBML Qual package description (Section 1.3).

Specifically, we use UML 1.0 (Unified Modeling Language) class diagram notation to define the constructs provided by this package. We also use colour in the diagrams to carry additional information for the benefit of those viewing the document on media that can display color. The following are the colors we use and what they represent:
Black: Items colored black in the UML diagrams are components taken unchanged from their definition in the SED-ML Level 1 Version 2 specification document.
Green: Items colored green are components that exist in SED-ML Level 1 Version 2, but are extended by this package. (We do not use dashed lines to further distinguish them, as dashed lines indicate abstract classes in SED-ML notation).
Blue: Items colored blue are new components introduced in this package specification. They have no equivalent in the SED-ML Level 1 Version 2 specification.

Context

With this extension, we aim to encode simulation experiments on qualitative models. The current SED-ML does already provide the frame for the encoding of simulation experiments in computational biology. However, the nature of qualitative models is different to ODE-based models, which have been in the focus so far: Qualitative models are a class of discrete event systems, whose dynamics are represented by means of state transition graphs (STGs). These STGs represent, in the form of a graph, which nodes are the reachable states; the edges represent the state transitions.

The proposal at hand is the result of discussions at Instituto Gulbenkian de Ciência (IGC), Oeiras, Portugal, February 2016.

Scope of the extension

For qualitative models, we consider the generation of transition graphs, which could be represented by the following outputs:

(1) Generation of state transition graphs
(2) Generation of time courses with specific, defined sliding windows
(3) Generation of so-called dynamic profiles showing the state changes of a system

The transition graphs can be generated on the initial model (in SBML Qual format), or on model variants. Thus SED-ML should cover the definition of sets of initial states, mutants, and priority classes.

Initial state(s): This is the initial condition of the simulation, that is the (set of) initial state(s) from which the construction of the state transition graph (done by generating successor states) is performed.

Mutant: one important asset of qualitative models is that model perturbations are easy to define by adequately modifying the transitions of the perturbed species ; these amounts to

knockout (KO, blocking the variable to 0), ectopic activity (E*m*, blocking the variable to its maximal value *m*) or, for multi-valued components, intermediate range ([i,j], maintaining the variable within the specified interval). Another type of perturbation singles out the regulators of a component: selecting a regulator, this is subject to the previous perturbations (*e.g.* a KO regulator specifies that the component is insensitive to this regulator).

Priority class: relying on biological knowledge about the mechanisms at play, one can classify processes (*i.e.* concurrent transitions of the asynchronous dynamics) in terms of rates. A typical case is to define 2 priority classes, distinguishing fast and slow processes. More sophisticated settings then associate to each priority class an update mode (*e.g.* fast transitions (events) occur synchronously and slow ones are performed asynchronously), or yet distinguish between transitions that increase or decrease the level of a species.

Proposed extensions to the current SED-ML standard

To cover the description of the above simulation experiments, we propose the following extensions to SED-ML classes.

1. Introducing a new Simulation Class (Section 2.4.3)

The Simulation class serves as the container for the different types of experiments to be performed on a given model.
We propose to introduce a new type, called *DiscreteSimulation* (Fig.1). *DiscreteSimulation* would be derived from Simulation.

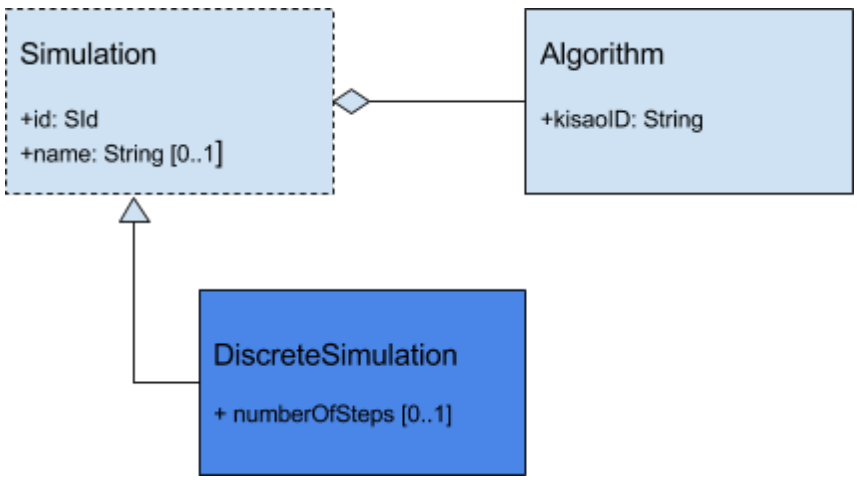

Figure 1: DiscreteSimulation as a specialisation of the abstract Simulation class.

The *DiscreteSimulation* class has derived attributes *id* and *name*. It has one additional attribute *numberOfSteps*.

Number of steps: this is the upper bound of the number of iterations (defined as the number of successors generated), which also corresponds to the depth of the exploration

from the initial state(s). Note that when no new states are encountered, the simulation stops before reaching this number. In the case of synchronous simulation (or any updating policy considering the selection of a unique successor state at each step), this number is equivalent to the number of states.

Example

```
<listOfSimulations>
   <discreteSimulation id=”sim1” name=”example for generation of STG with restricted number of states”
numberOfSteps=”5”>
      <algorithm kisaoID=”kisao:KISAO_0000363 />
   </discreteSimulation>
</listOfSimulations>
```

2. Introducing a new Task Class (Section 2.4.5)

The *AbstractTask* class serves as the container for the different types of tasks to be performed on a given model and using a specific simulation setup.
Logical models can be simulated using various update modes. These modes can be defined for different sets of species, using so-called priority classes. We propose to store the information about priority classes in relation to a specific task, thus extending the *Task* class by a new subclass called *PriorityClass* (Fig. 2).

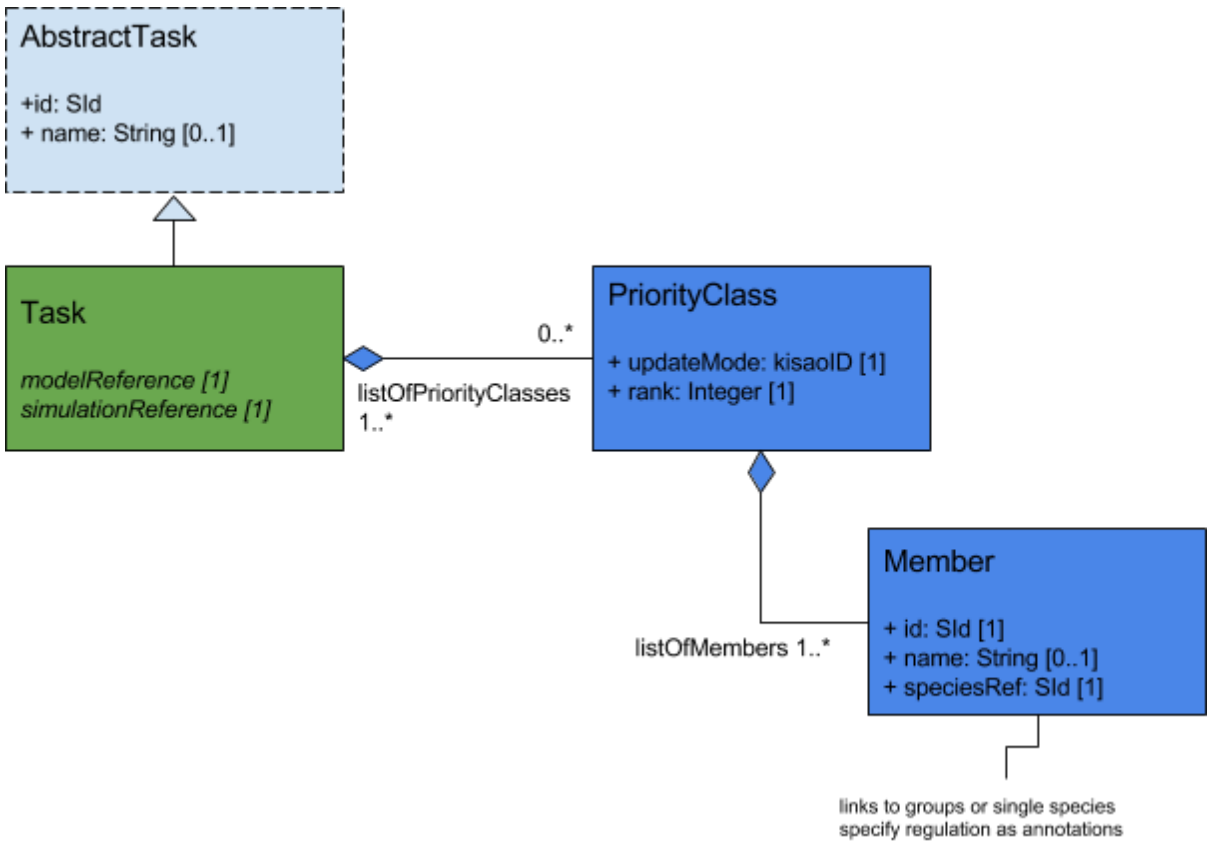

Figure 2: PriorityClass class and Member class

A *PriorityClass* is part of a *Task*. With this extension, it will be possible to define a number of priority classes for a given task. The *Task* already contains a *modelReference* (pointing to

the model of interest) and a *simulationReference* (pointing to the simulation of interest). Thus the defined priority classes are valid within this task only.
In the list of *PriorityClasses*, each such class receives a rank that determines when it will be considered for the update. In addition, an update model can be defined on the basis of *PriorityClasses*, using on of the following predefined modes (linked through a KiSAO ID):

- synchronous *(KiSAOID to be announced)*
- asynchronous *(KiSAOID to be announced)*
- fully asynchronous *(KiSAOID to be announced)*
- random asynchronous *(KiSAOID to be announced)*

Note: We recommend to define at least one *PriorityClass* per task -- this is necessary to provide information on the update mode. If one or more priority classes are defined, then all species must appear in at least one of these classes.
In the case of a unique priority class, all species should be contained (list= "/qual:listOfQualitativeSpecies") to define the update mode of the system.
If no such unique priority class is provided, the simulators will assume that all species are updated synchronously.

Example: Defining one priority class to declare the update mode of the model as synchronous.

```
<listOfTasks>
  <task id="task1" name="example for a task with one priority class covering all species in the model">
    <listOfPriorityClasses>
      <priorityClass id="pc1" name="fast" updateMode="someKiSAOID-synchronous" rank="1">
        <listOfMembers>
          <member id="mem1" name="all species"
                speciesRef="sbml:sbml/sbml:model/qual:listOfQualitativeSpecies" />
        </listofMembers>
      </priorityClass>
    </listOfPriorityClasses>
  </task>
</listOfTasks>
```

Each defined *PriorityClass* must contain a list of members. These members define the (SBML) species for which a specific update mode is given.

Each Member has an *id* and an optional *name*. Furthermore, a member contains a link to (1) a group of species (Using the SBML Group package) or to
(2) a single species, using the species' *SId*.
To each member, an annotation can be added to indicate the direction of the update (increase or decrease). Only in this case, a species could appear in two different priority classes, with different directions of update.

Example: defining two priority classes, to separate fast from slow reactions.

```
<listOfTasks>
  <task id="task2" name="example for a task with two defined priority classes">
    <listOfPriorityClasses>
      <priorityClass id="pc2" name="fast species" updateMode="someKiSAOID-asynchronous" rank="1">
        <listOfMembers>
```

```
            <member id="mem1" name="species A"

          speciesRef="sbml:sbml/sbml:model/qual:listOfQualitativeSpecies/qual:qualitativeSpecies[@id='A']">
            <member id="mem2" name="species B"

          speciesRef="sbml:sbml/sbml:model/qual:listOfQualitativeSpecies/qual:qualitativeSpecies[@id='B']">
           </listofMembers>
        </priorityClass>
        <priorityClass id="pc3" name="slow species" updateMode="someKiSAOID-asynchronous" rank="2">
          <listOfMembers>
            <member id="mem1" name="species C"
             speciesRef="sbml:sbml/sbml:model/qual:listOfQualitativeSpecies/qual:qualitativeSpecies[@id='C']">
          </listofMembers>
        </priorityClass>
      </listOfPriorityClasses>
   </task>
</listOfTasks>
```

3. Best practise: Annotation in the Data Generator

In the case of synchronous updates, and if the output type is a *plot2D*, then we need to define the species that are to be displayed. This can be done with the existing concept of a *DataGenerator*.

For the rather specific case that a sliding window must be defined for the simulation output (in a *plot2D*), we recommend to put this information into an annotation.

SlidingWindow: for the evolution of a variable along the simulation steps, it is possible to consider the average value over a sliding window of a given number of iterations (that is the size of the sliding window).

4. Introducing TransitionGraph as a new type of Output (Section 2.4.9)

The typical result of a simulation of an SBML Qual model is the state transition graph of the system. We hence propose to add the **TransitionGraph** class as a new type of output

(Figure 3).

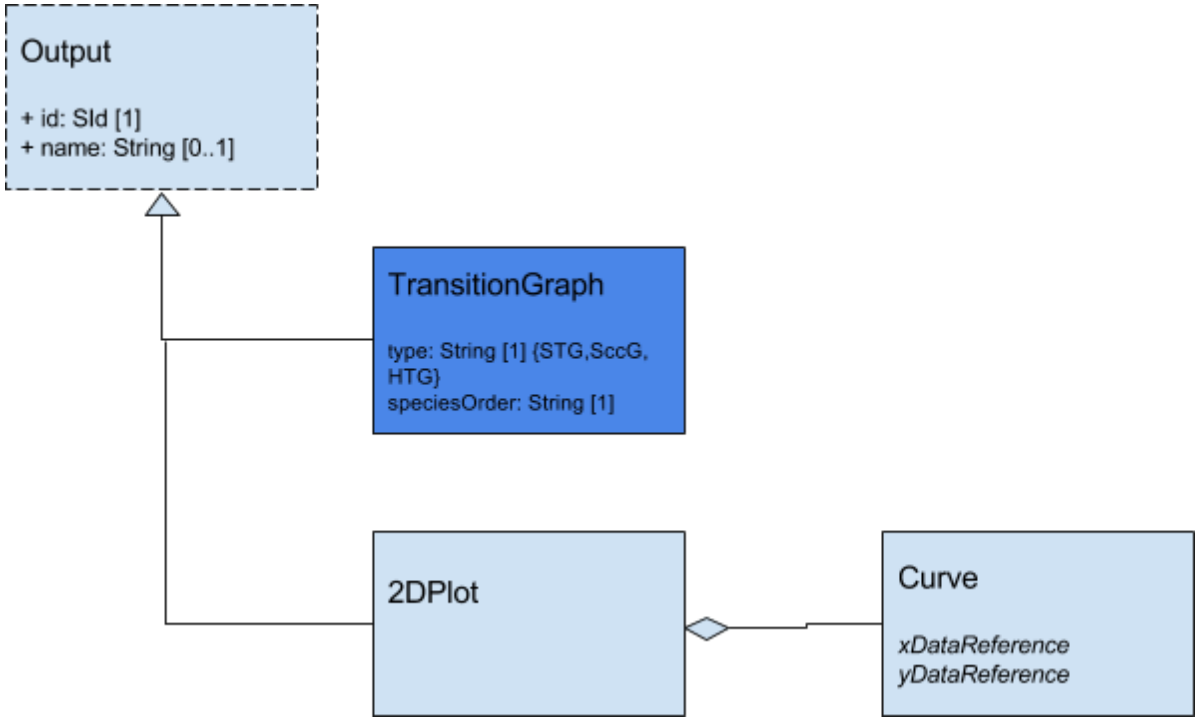

Figure 3: DynamicProfile and TransitionGraph classes

Transition graphs have an *id* and an optional *name*. They furthermore have a *type* attribute which provides the type of graph that should be shown in the output.

STG: State transition graph - A State Transition Graph (STG) is a directed graph representing the dynamical behaviour of a qualitative model. Nodes of this graph represent states of the model, assigning a level to each species. Arcs of the STG represent transitions from one state to another (*i.e.* level changes of one or several species).

SCCG: Strongly connected component graph - A Strongly Connected Component of a directed graph is a maximal subgraph encompassing mutually reachable nodes. The Strongly Connected Component graph of the STG (SCCG) is thus a compressed representation of the STG, in which each node represents either a single state or a SCC.

HTG: Hierarchical transition graph - A Hierarchical Transition Graph (HTG) is a compressed representation of the STG, in which each node represents either a SCC, or a (set of) linear chain(s) leading to the same set of SCCs and attractors (see [PMID:23822512]).

Transition graphs must provide an order of elements in the output. This is done through a list of species ids, encoded in the *speciesOrder* attribute of type string. Give the order of elements by listing the IDs of the species, separated by a space.

Example

<listOfOutputs>

```
  <transitionGraph id="tg1" name="example of a state transition graph" type="STG"
      taskReference="task1" speciesOrder="A B C">
  <transitionGraph id="tg2" name="example of a hierarchial transition graph" type="HTG"
                          taskReference="task2" speciesOrder="A B C">
</listOfOutputs>
```

Examples for the use of existing SED-ML concepts

1. Changes to the model (preprocessing)
    a. Set initial state
       To define a particular initial state of the model, the *Change* class can be used to set (or update) the levels of the (optional) *initialLevel* attribute inside the *qualitativeSpecies* element.
       Note: As per convention of the SBML Qual, the absence of the *initialLevel* attribute is interpreted as "use all possible levels".
    b. Define mutants
       To define mutations, the *Change* class can be used to modify the *transition* element in the SBML Qual model. The *listOfFunctionTerms* are modified to encode mutations. For example, to define a knockout, the *FunctionTerm* elements are deleted and the *DefaultTerm* is set to 0 (but to the maximum value to define an ectopic activity). It is possible to further describe the nature of change (as a mutation) using existing ontology terms.
    c. Define reduction
       To store a reduced version of the model, the unnecessary species and transitions can be removed using the *RemoveXML* class.

2. Output types
    a. Dynamic profiles
        i. Dynamic profiles are a specific type of output used in logical modeling. They are tabular displays of species behaviours for specific simulation experiments. The most common just lists the values of the (selected) species over time.
        ii. The best practice is to define dynamic profiles using the Report class, as described in the SED-ML spec. The order of the species in the profile can be given through annotation.
    b. Curves
        i. Another common output type are 2d plots. Each plotted species is defined in the *listOfDataSets*, which links to the relevant *DataGenerator*. This is the identical procedure already used to define plots in SED-ML.