Bsys_eval

Ionuț Cicio

20/10/2025

https://github.com/CuriousCI/bsys-eval

Contents

1 Bsys_eval

1.1 Introduction

Bsys_eval is a tool meant to help study the likelihood of a given scenario in a biological system.

Given a set of *target species*, a set of constraints on the *target species* (constraints which model a scenario that could present, for example, in a disease) and by taking into account all the reactions within a set *target pathways* that lead to the production, both directly and indirectly, of the *target species*, the goal is to find a subset of virtual patients for the described scenario.

TODO: find papers in literature that do similar things; what does this method add compared to other approaches? (i.e. using multiple pathways by generating the fixed point, ensemble of SAs etc…)

TODO: case studies

1.2 Requirements

The basic idea behind the software is to take the description of a scenario (with *target species*, *target pathways*, constraints on the *target species*, and the parameters $\varepsilon, \delta \in (0, 1)$ for the evalation of the constraints), to generate a SBML model with

- the reactions within the *target pathways* that, both directly and indirectly, generate the *target species*
- parameters for the reactions' speeds
- structural constraints on the reactions' speeds (some reactions are faster than others)

  > TODO: I still haven't figured out how to get that information out of Reactome, maybe I just have to search more

- constraints on the quantities of the entities (for which the model needs to be simulated)

Algorithm 1: eval

---

input: S_T, set of **PhysicalEntity**;
input: P_T, set of target **Pathway**;
input: C_T, set of constraints on S_T;
input: $\varepsilon, \delta \in (0, 1)$;
input: seed, random seed;

scenario $\leftarrow$ *biological_scenario_definition*(S_T, P_T, C_T)
(sbml_model, vp_definition, env) $\leftarrow$ *yield_sbml_model*(scenario)

$V = \emptyset$ // set of virtual patients
while $\neg$ halt requested **do**
  $v \leftarrow$ *instantiate*(vp_definition) // virtual patient
  if (
      v satisfies structural constraints $\wedge$
      APSG(sbml_model, v, env, seed, ε, δ)
) **then**
      $V \leftarrow V \cup \{v\}$;

return V

The bulk of the logic is in the *yield_sbml_model*() function.

Average quantities

- $S' = S \cup \{S_{\text{avg}} \mid s \in S\}$
- $S' = G(S')$
- $K : R \to \mathbb{R}_+^{|R|} = [10^{-6}, 10^6]^{|R|}$
- find k
- subject to
  - structural constraints
    - partial order on k due to
      - fast/non fast reactions (TODO: as given by Reactome, but how?)

$$\forall r_f, r_s \ \left(r_f \in R_{\text{fast}} \wedge r_s \in R_{\text{slow}}\right) \to r_f > r_s$$

      - reaction modifiers (like above?)
  - for all dynamics of environment
    - avg concentration of species consistent to knowledge

$$\exists t_0 \ \ \forall t \ \ \forall s$$
$$\left(t > t_0 \wedge s \in S_{\text{avg}}\right) \to s(t) \in [\text{known range}]$$

- $\mathbb{N}^+ = \{n \mid n \in \mathbb{N} \land n > 0\}$

Definition 1 (*biological graph*)

A *biological graph* G is a tuple (S, R, E, F) s.t.

- S is a set of species
- R is a set of reactions
- $E : S \times R \to \mathbb{N}^+$
  - $E = E_{\text{reactant}} \cup E_{\text{product}} \cup E_{\text{modifier}}$

Definition 2 (*"produces" relation*)

Given a *biological graph* $G = (S, R, E, F)$ let $\rightsquigarrow$ be a relation s.t.

- $\forall s, r \quad (s, r) \in \text{dom}(E_{\text{reactant}} \cup E_{\text{modifier}}) \Rightarrow s \rightsquigarrow r$
- $\forall s, r \quad (s, r) \in \text{dom}\big(E_{\text{product}}\big) \Rightarrow r \rightsquigarrow s$
- $\forall c, c', c'' \quad (c \rightsquigarrow c' \land c' \rightsquigarrow c'') \Rightarrow c \rightsquigarrow c''$

$\rightsquigarrow$ is the *"produces"* relation

Definition 3 (*model??*)

Given a set of target species S_T, a set of target pathways P_T and a biological graph $G = (S, E, R)$ s.t.

- $S_T \subseteq S$

A *model* is a subgraph $G' = (S', E', R')$ s.t.

- $G' \subseteq G$
- $S' = \{s \mid \exists s_t \; s \in S \land s_t \in S_T \land s \rightsquigarrow_G s_t\}$
- $R' = \{r \mid \exists s_t \; r \in R \land s_t \in S_T \land r \rightsquigarrow_G s_t\}$
- $E' = ((s, r, n) \mid (s, r, n) \in E \land s \in S' \land r \in R')$

Definition 4 (*constraint problem on a biological model*)

Given a model G with target species S_T and target pathways P_T let the following be a constraint problem

- $k : R \to \mathbb{R}^{|R|}$

find k subject to

- partial order on k from the structure of the graph
- partial order on the quantities
- constraint on enzymes such that

$$E + S \overset{k_1, k_{-1}}{\leftrightarrow} ES \overset{k_2}{\to} E + P$$

$$k_1, k_{-1} \gg k_2$$

- for all dynamics of the environment
  - average concentration of species consistent to knowledge

$$\exists t_0 \ \ \forall t \ \ \forall s$$
$$\big(t > t_0 \wedge s \in S_{\text{avg}}\big) \rightarrow s(t) \in [\text{known range}]$$

Environment: all possible cuts
we can have excluded species!

$$\dot{x} = k_+ \prod_{i=1}^{s} S_i^{k_i} - k_- \prod_{j=1}^{p} P_j^{k_j}$$

$$\dot{x} = \sum_{i=1}^{p} \text{KP}_i - \sum_{j=1}^{n} \text{KN}_j$$

$$\left\{ \sum_{j=1}^{n} \text{KN}_j > \text{KP}_i \mid i \in \{1, ..., p\} \right\} \cup$$
$$\left\{ \sum_{i=1}^{n} \text{KP}_i > \text{KN}_j \mid j \in \{1, ..., n\} \right\}$$

2 Data types specification

- \d = /[0-9]/
- \w = /[A-Za-z0-9_]/

Math

Natural = **Integer** >= 0
Interval = (lower_bound: **Real** [0..1], upper_bound: **Real** [0..1])
MathML = **String** matching https://www.w3.org/1998/Math/MathML/
MathMLBoolean = **String** matching **MathML** returning a **Boolean**
MathMLNumeric = **String** matching **MathML** returning a **Number**
Stoichiometry = **Natural** > 0

Reactome

ReactomeDbId = **Natural** [1]
StableIdVersion =
    String matching regex /^R-[A-Z]{3}-\d{1,8}\.\d{1,3}$/ [2]

SBML

String1 = **String** matching regex //
SId = **String** matching regex /^[a-zA-Z_]\w*$/ [3, Section 3.1.7]
UnitSId = **String** matching regex /^[a-zA-Z_]\w*$/

2.1 Interval

The **Interval** type represents an open interval in $\mathbb{R}$ of the type (lower_bound, upper_bound) s.t.

- when lower_bound is not defined, it is interpreted as $-\infty$
- when upper_bound is not defined, it is interpreted as $+\infty$

```
C.Interval.lower_bound_leq_upper_bound
  ∀ interval, interval_lower_bound, interval_upper_bound
      (
          Interval(interval) ∧
          lower_bound(interval, interval_lower_bound) ∧
          upper_bound(interval, interval_upper_bound)
      ) →
          interval_lower_bound ≤ interval_upper_bound
```

2.2 ReactomeDbId

This is required because not all instances of **DatabaseObject** in Reactome have a **StableIdVersion**, which is the one usually displayed in the Reactome Pathway Browser [4]. Instances of **DatabaseObject** in Reactome can be identified with a **ReactomeDbId**, but its pattern does not match the definition of **SId** used to identify objects in SBML.

In order to generate a correct **SBMLDocument** the **ReactomeDbId** must be converted into a **SId**.

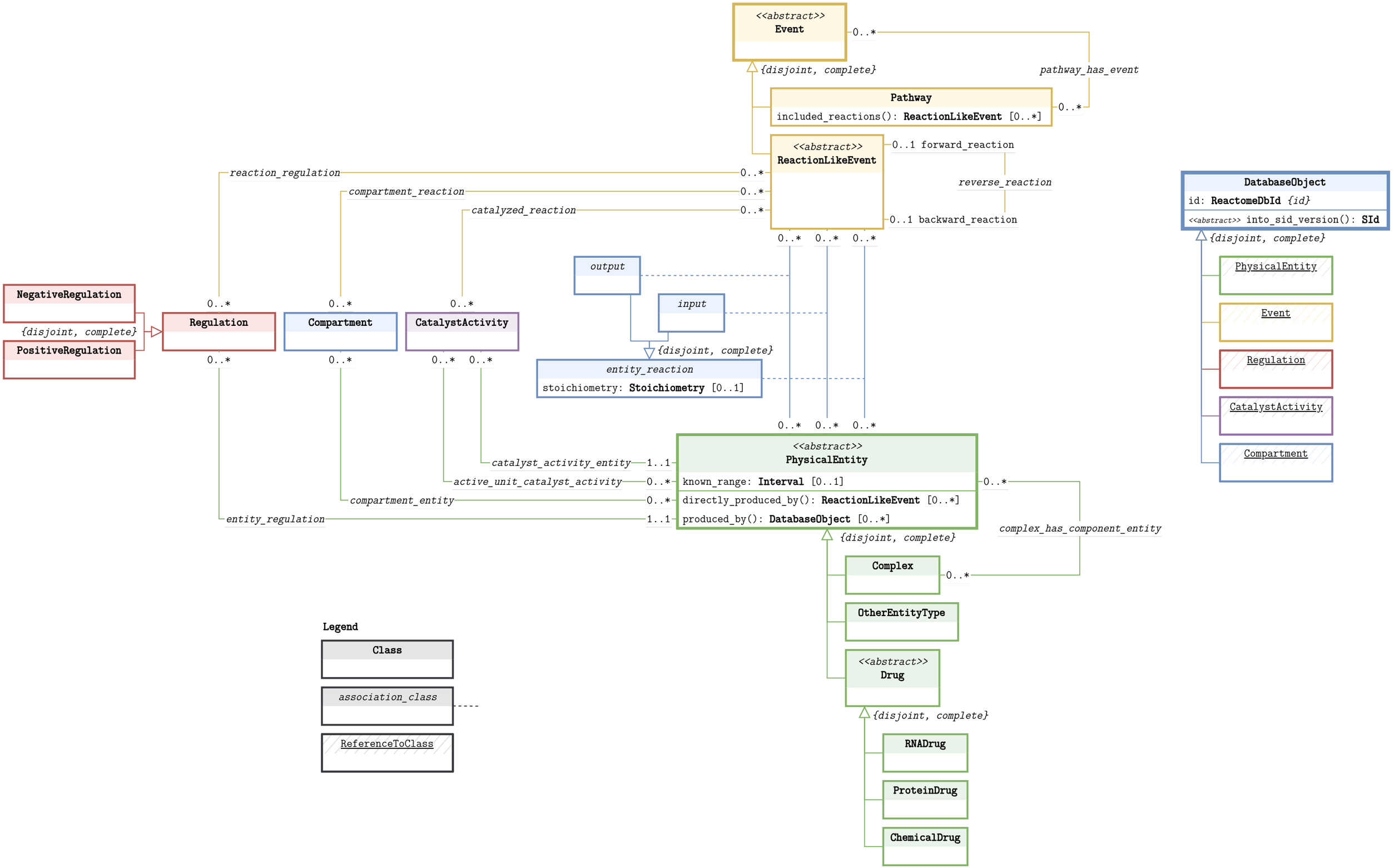
2.3 StableIdVersion

The **StableIdVersion** type is useful because is the one usually displayed in the Reactome Pathway Browser [4]. It is useful to accept it in the description of the models.

from_stable_id_version(stable_id_version: **StableIdVersion**) : **ReactomeDbId**

```
  postconditions:
    ...
```

3 *(Reactome)* UML class diagram



4 Classes specification pt. 1

4.1 CatalystActivity

The role of **PhysicalEntity** in *catalyst_activity_entity* has multiplicity 0..* because *"If a **PhysicalEntity** can enable multiple molecular functions, a separate **CatalystActivity** instance is created for each"* [5, Page 5].

An additional constraint is required for active units, because *"If the **PhysicalEntity** is a **Complex** and a component of the complex mediates the molecular function, that component should be identified as the active unit of the **CatalystActivity**."* [5, Page 5]

```
C.CatalystActivity.active_unit_is_component_of_complex
  ∀ catalyst_activity, complex, complex_component
      (
          CatalystActivity(catalyst_activity) ∧
          Complex(complex) ∧
          PhysicalEntity(complex_component) ∧
          catalyst_activity_entity(catalyst_activity, complex)
  ∧
          catalyst_activity_active_unit(
              catalyst_activity,
              complex_component
          )
      ) →
          complex_has_component_entity(complex,
  complex_component)
```

4.2 Compartment

The **Compartment** class has some quirks. In Reactome, the **Compartment**'s role in the *compartment_entity* association has multiplicity 0..*. The problem is that the SBML model requires 1..1 multiplicity for this association to be simulated.

In Reactome there are currently (TODO: version??) 19 physical entities which don't have a compartment (see queries/helper.cypher), so this can be easily solved by just adding a **default compartment** to the SBML model to which these entities map to.

On the other hand there are 14046 entities which have multiple compartments (TODO: how many compartments has each exactly?), so the easiest choice right now is to just pick any of them. For this reason the

4.3 Pathway

The instances of **Pathway** are organized hierarchically, i.e. all the signaling pathways are collected under the Signal Transduction top level **Pathway** (**StableIdVersion** R-HSA-162582.13). This allows to easily extract a subset of reactions by specifying the *target pathways* in a model and taking into consideration only the reactions which are included, both directly or indirectly, in that pathway (see the *included_reactions*() operation).

There are about 34 top level pathways.

```
included_reactions(): ReactionLikeEvent [0..*]
  postconditions:
    result =
        { reaction |
            ReactionLikeEvent(reaction) ∧
            pathway_has_event(this, reaction) }
        ∪
        { reaction | ∃ pathway
            Pathway(pathway) ∧
            pathway_has_event(this, pathway) ∧
            included_reactions(pathway, reaction) }
```

4.4 PhysicalEntity

The set of reactions which produce this is needed to determine the transitive closure of the *target entities*.

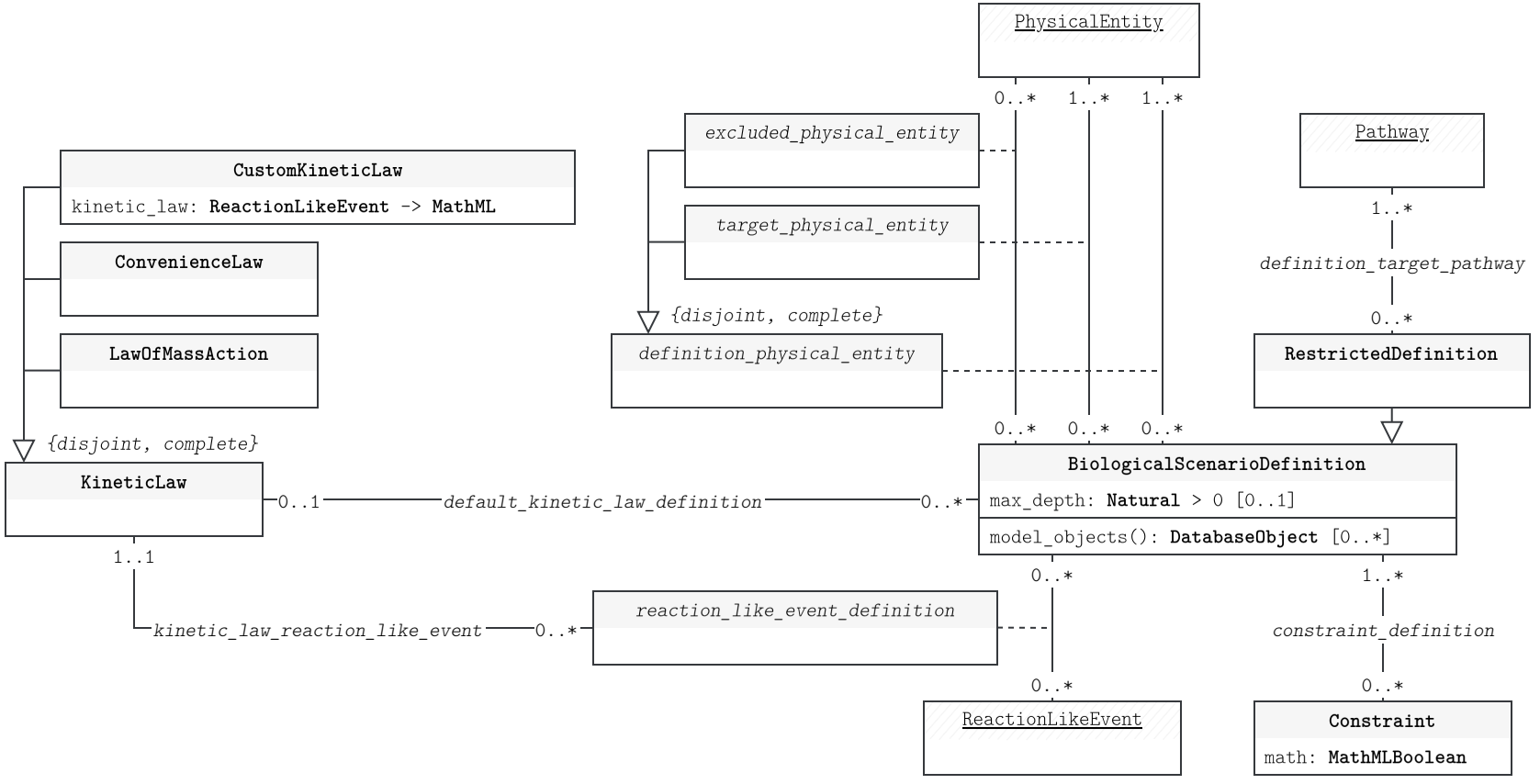
directly_produced_by() : **ReactionLikeEvent** [0..*]

```
  postconditions:
    result = { reaction |
        ReactionLikeEvent(reaction) ∧ output(this, reaction)
    }
```

The set of instances of **DatabaseObject** which are directly or indirectly involved in the production of this.

produced_by() : **DatabaseObject** [0..*]

```
  postconditions:
    result =
        { this } ∪
        { reaction | directly_produced_by(this, reaction) } ∪
        { object | ∃ reaction, reaction_input
            directly_produced_by(this, reaction) ∧
            (
                *input*(reaction, reaction_input) ∨
                (∃ catalyst_activity
                    CatalystActivity(catalyst_activity) ∧
                    *catalyzed_event*(
                        catalyst_activity,
                        reaction
                    ) ∧
                    *catalyst_activity_entity*(
                        catalyst_activity,
                        reaction_input
                    )
                )
            ) ∧
            produced_by(reaction_input, object)
        }
```


PhysicalEntity
0..*
1..*
1..*
excluded_physical_entity
target_physical_entity
{disjoint, complete}
definition_physical_entity
0..*
0..*
0..*
Pathway
1..*
definition_target_pathway
0..*
RestrictedDefinition
CustomKineticLaw
kinetic_law: ReactionLikeEvent -> MathML
ConvenienceLaw
LawOfMassAction
{disjoint, complete}
KineticLaw
0..1
default_kinetic_law_definition
0..*
BiologicalScenarioDefinition
max_depth: Natural > 0 [0..1]
model_objects(): DatabaseObject [0..*]
1..1
kinetic_law_reaction_like_event
0..*
reaction_like_event_definition
0..*
1..*
constraint_definition
0..*
0..*
ReactionLikeEvent
Constraint
math: MathMLBoolean

5 Classes specification pt. 2

5.1 BiologicalScenarioDefinition

The following operation finds the transitive closure of the *target entities* specified in the scenario, by including only reactions within the *target pathways* if necessary.

model_objects() : **DatabaseObject** [1..*]

```
  postconditions:
    result = { object | ∃ entity
        PhysicalEntity(entity) ∧
        DatabaseObject(object) ∧
        target_physical_entity(this, entity) ∧
        produced_by(entity, object) ∧
        (
            ¬ RestrictedDefinition(this) ∨
            ∃ pathway, reaction
                Pathway(pathway) ∧
                ReactionLikeEvent(reaction) ∧
                included_reactions(pathway, reaction) ∧
                (
                    object = reaction ∨
                    entity_reaction(object, reaction) ∨
                    catalyzed_reaction(object, reaction)
                )
        )
    }
```

5.2 ModelInstance

C.**ModelInstance**.no_local_parameters_without_value

```
∀ model_instance, model, reaction, kinetic_law,
local_parameter
    (
        ModelInstance(model_instance) ∧
        Model(model) ∧
        ReactionDefinition(reaction) ∧
        KineticLaw(kinetic_law) ∧
        LocalParameter(local_parameter) ∧
        *instance_model*(model_instance, model) ∧
        *model_definition*(model, reaction) ∧
        *kinetic_law_reaction*(kinetic_law, reaction) ∧
        *kinetic_law_local_parameter*(kinetic_law,
local_parameter) ∧
        ¬ ∃ value
            value(local_parameter, value)
    ) →
        ∃ local_parameter_assignment

LocalParameterAssignment(local_parameter_assignment) ∧
            *model_instance_paramenter*(
                model_instance,
                local_parameter_assignment
            ) ∧
            *assignment_local_paramenter*(
                local_parameter_assignment,
                local_parameter
            )
```

5.3 SimulatedModelInstance

is_valid()

```
postconditions:
  ...
```

5.4 Measurement

C.**Measurement**.species_in_model

```
∀ measurement, model_instance, model, species
    (
        Model(model) ∧
        SimulatedModelInstance(model_instance) ∧
        Measurement(measurement) ∧
```

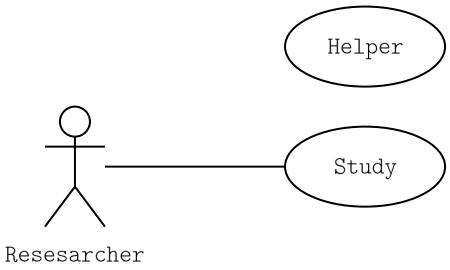
```
        Species(species) ∧
        *measurement_species*(measurement, species) ∧
        *measurement_simulation*(measurement, model_instance)
∧
        *instance_model*(model_instance, model)
    ) →
        *model_definition*(model, species)
```

5.5 UnitDefinition

[3, page 45]

TODO: better description

6 Use-case diagram



6.1 “Helper” use-case

yield_sbml_model(description: **BiologicalScenarioDefinition**) : (**SBMLDocument**,)

`postconditions:`

TODO:

- create necessary units (**TODO**: which? how?)
- create default **CompartmentDefinition**
- create **CompartmentDefinition** from **Compartment**
  - convert id to **SId**
- create **SpeciesDefinition** from **PhysicalEntity**
  - convert id to **SId**
  - add one of the compartments if the entity has any
  - otherwise assign to default
- create **ReactionDefinition**
  - convert id to **SId**
  - connect products (inputs)
  - connect reactants (outputs)
  - connect modifiers (catalysts)
  - add kinetic law (either manually specified ∨ automatic, like **LawOfMassAction**)
  - add local parameters
- create constraints
  - i.e. from known_range attribute
-

instantiate_model(model: **SBMLDocument**) : **ModelInstance**

`postconditions:`

TODO:

- add **LocalParameterAssignment** for undefined **LocalParameters**
- add environment parameters to model (**Parameter**)

simulate_model(instance: **ModelInstance**) : **SimulatedModelInstance**

`postconditions:`

TODO:

- generate measurements

7 OpenBox on the Slurm Workload Manager

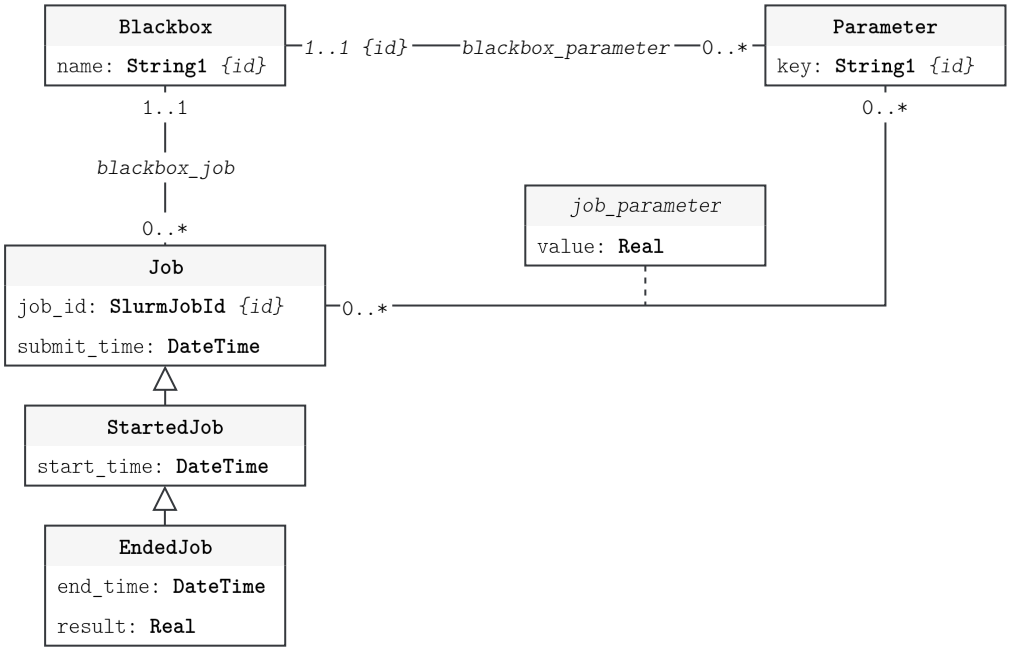
The HPC cluster at the Computer Science Department has some restrictions in place, as it's used by many different teams / students and no single user can request the indefinite usage of the whole cluster for a single job (jobs have a time limit of 6, 24 or 72 hours based on permissions and resoruces required).

The goals of this section are to

- be able to run an OpenBox throught **multiple sessions**
- run **multiple smaller jobs** to increase **fairness** among users, instead of running a single big job for the whole simulation
- provide a simple framework that can be used **locally to simulate** executions on the cluster

7.1 Analysis

In order to use OpenBox on the cluster in different sessions, it's a good idea to store the results of the simulations in a database (i.e. PostgreSQL) to retrieve the data of different session for an overall analysis.



7.1.1 Data types specification

SlurmJobId = **Integer** >= 1
String1 = **String** matching regex /^\S$|^\S.*\S$/

7.1.2 Classes specification

7.1.2.1 Job

C.**Job**.all_parameters_are_instantiated

$$
\begin{aligned}
&\forall\ \text{job}, \text{blackbox}, \text{parameter} \\
&\quad (\\
&\qquad \mathbf{Job}(\text{job}) \wedge \\
&\qquad \mathbf{Blackbox}(\text{blackbox}) \wedge \\
&\qquad \mathbf{Parameter}(\text{parameter}) \wedge \\
&\qquad \textit{blackbox\_job}(\text{blackbox}, \text{job}) \wedge \\
&\qquad \textit{blackbox\_parameter}(\text{blackbox}, \text{parameter}) \\
&\quad) \rightarrow \\
&\qquad \textit{job\_parameter}(\text{job}, \text{parameter})
\end{aligned}
$$

C.**Job**.continuity_1

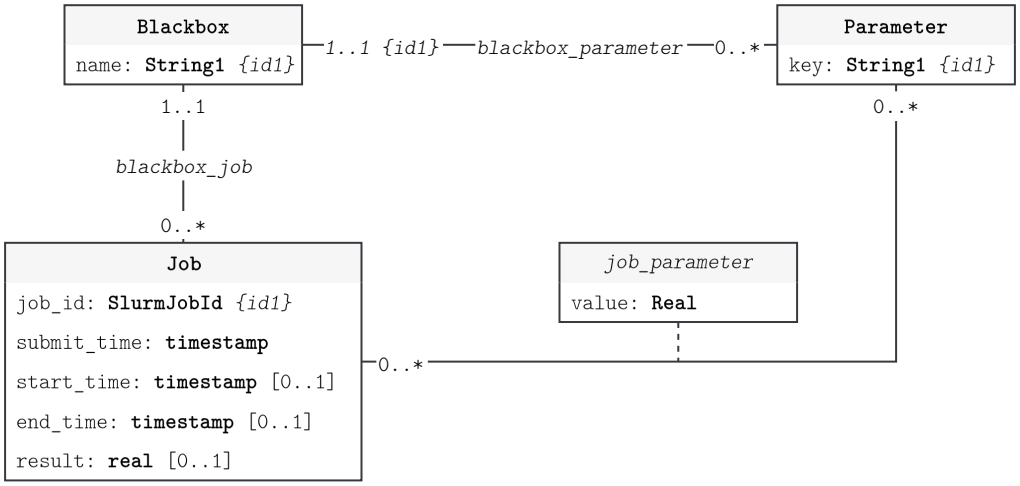
$$
\begin{aligned}
&\forall\ \text{job}, \text{submit\_time}, \text{start\_time} \\
&\quad (\\
&\qquad \mathbf{Job}(\text{job}) \wedge \\
&\qquad \underline{\text{submit\_time}}(\text{job}, \text{submit\_time}) \wedge \\
&\qquad \underline{\text{start\_time}}(\text{job}, \text{start\_time}) \\
&\quad) \rightarrow \\
&\qquad \text{submit\_time} \leq \text{start\_time}
\end{aligned}
$$

C.**Job**.continuity_2

$$
\begin{aligned}
&\forall\ \text{job}, \text{start\_time}, \text{end\_time} \\
&\quad (\\
&\qquad \mathbf{Job}(\text{job}) \wedge \\
&\qquad \underline{\text{start\_time}}(\text{job}, \text{start\_time}) \wedge \\
&\qquad \underline{\text{end\_time}}(\text{job}, \text{end\_time}) \\
&\quad) \rightarrow \\
&\qquad \text{start\_time} \leq \text{end\_time}
\end{aligned}
$$

7.2 Implementation

Diagram restructuration for PostgreSQL. The SQL code is available in the `migration.sql` file.



7.2.1 Data types definitions

```
CREATE DOMAIN String1 AS varchar CHECK(value ~ '^\S$|^\S.*\S$');
CREATE DOMAIN SlurmJobId AS integer CHECK(value >= 1);
```

7.2.2 Additional constraints

7.2.2.1 Job

`C.Job.end_implies_job_was_scheduled`

```
∀ job, end_time
  (Job(job) ∧ end_time(job, end_time)) →
    ∃ start_time start_time(job, start_time)
```

A result is present if and only if the job ended

`C.Job.result_only_on_end_time`

```
∀ job, job_result
  (Job(job) ∧ result(job, job_result)) →
    ∃ end_time end_time(job, end_time)
```

`C.Job.end_time_only_on_result`

```
∀ job, end_time
  (Job(job) ∧ end_time(job, end_time)) →
    ∃ job_result result(job, job_result)
```

Bibliography

[1] [Online]. Available: https://reactome.org/content/schema/DatabaseObject

[2] "Reactome." [Online]. Available: https://reactome.org/documentation/faq/37-general-website/201-identifiers

[3] [Online]. Available: https://raw.githubusercontent.com/combine-org/combine-specifications/main/specifications/files/sbml.level-3.version-2.core.release-2.pdf

[4] [Online]. Available: https://reactome.org/PathwayBrowser/

[5] [Online]. Available: https://download.reactome.org/documentation/DataModelGlossary_V90.pdf