Bsys_eval

Ionuţ Cicio

02/09/2025

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 situation in a biological system.

Given a set of *target species*, a set of constraints on the *target species* (constraints which model a situation 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 situation.

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: add case study, multiple if possible

1.2 Requirements

The basic idea behind the software is to take the description of a model (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

- all 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)

TODO: possibly take a configuration file as input, maybe PEtab could be good, otherwise JSON should be enough, as everything else is generated automatically from Reactome, the model should work with both StableIDVersion and ReactomeDbId;

The `TargetPathway`s should be optional. The ExtraConstraints should be optional. The PreferredCompartmentForSimulation could be specified.

TODO: helper functions are described at page 15

Algorithm 1: eval

```
input: S_T, set of PhysicalEntity;
input: C_T, set of constraints on S_T;
input: P_I, set of target pathways;
input: ε, δ ∈ (0, 1);
input: seed, random seed;

model_description ← describe_model(S_T, P_I)
model ← generate_model(model_description)
env ← define env for model
V = ∅ // set of virtual patients

while ¬ halt requested do
   v ← instantiate_model(model) // virtual patient
   if ¬ v satisfies structural constraints then
     continue;
   if APSG(v, env, seed, ε, δ) then
     V ← V ∪ {v};
```

The idea is to expand a portion of Reactome

TODO: this page is far from complete, you can skip to the next one

Definition 1 *(… Model).* A … model G is a tuple (S_T, S, R, E) where:

- S_T the set of target species
- S is the finite set of species s.t.
  - $S_T \subseteq S$
  - S is the transitive closure of S_I within the Reactome graph (to be more precise, the closure within the specified bounds, bounds yet to be defined)
  - $S' = S \cup \{s_{\text{avg}} \mid s \in S\}$.
  - $\dot{s} = f(s_1, s_2, s_3, ..., s_n)$
- R is the finite set of reactions
  - $R = R_{\text{fast}} \cup R_{\text{slow}}$
- E is the set edges in the graph (where and edge goes from a species to a reaction, it also has a stoichiometry)
  - $E \subseteq S \times R \times \mathbb{N}^1$
  - $E = E_{\text{reactant}} \cup E_{\text{product}} \cup E_{\text{modifier}}$
  - TODO: account for order (edges also have an "order" attribute, I have to check how it impacts the simulation and if it's optional)

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?)

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

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

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

2 Data types specification

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

Math

```
Interval = (min: Real [0..1], max: 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 = Integer >= 0
```

Reactome

```
ReactomeDbId = Integer [1]
StableIdVersion =
   String matching regex /^R-[A-Z]{3}-\d{8}\.\d{2,3}$/ [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 $(\texttt{min}, \texttt{max})$ s.t.

- when `min` is not defined, it is interpreted as $-\infty$
- when `max` is not defined, it is interpreted as $+\infty$

```
[C.Interval.min_leq_max]
  ∀ interval, interval_min, interval_max
      (
          Interval(interval) ∧
          min(interval, interval_min) ∧
          max(interval, interval_max)
      ) →
          interval_min ≤ interval_max
```

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 `SBMLModel` the `ReactomeDbId` must be converted into a `SId`.

```
ReactomeDbId_into_SId(db_id: ReactomeDbId): SId
  POSTCONDITIONS:
    . . .
```

2.3 StableIdVersion

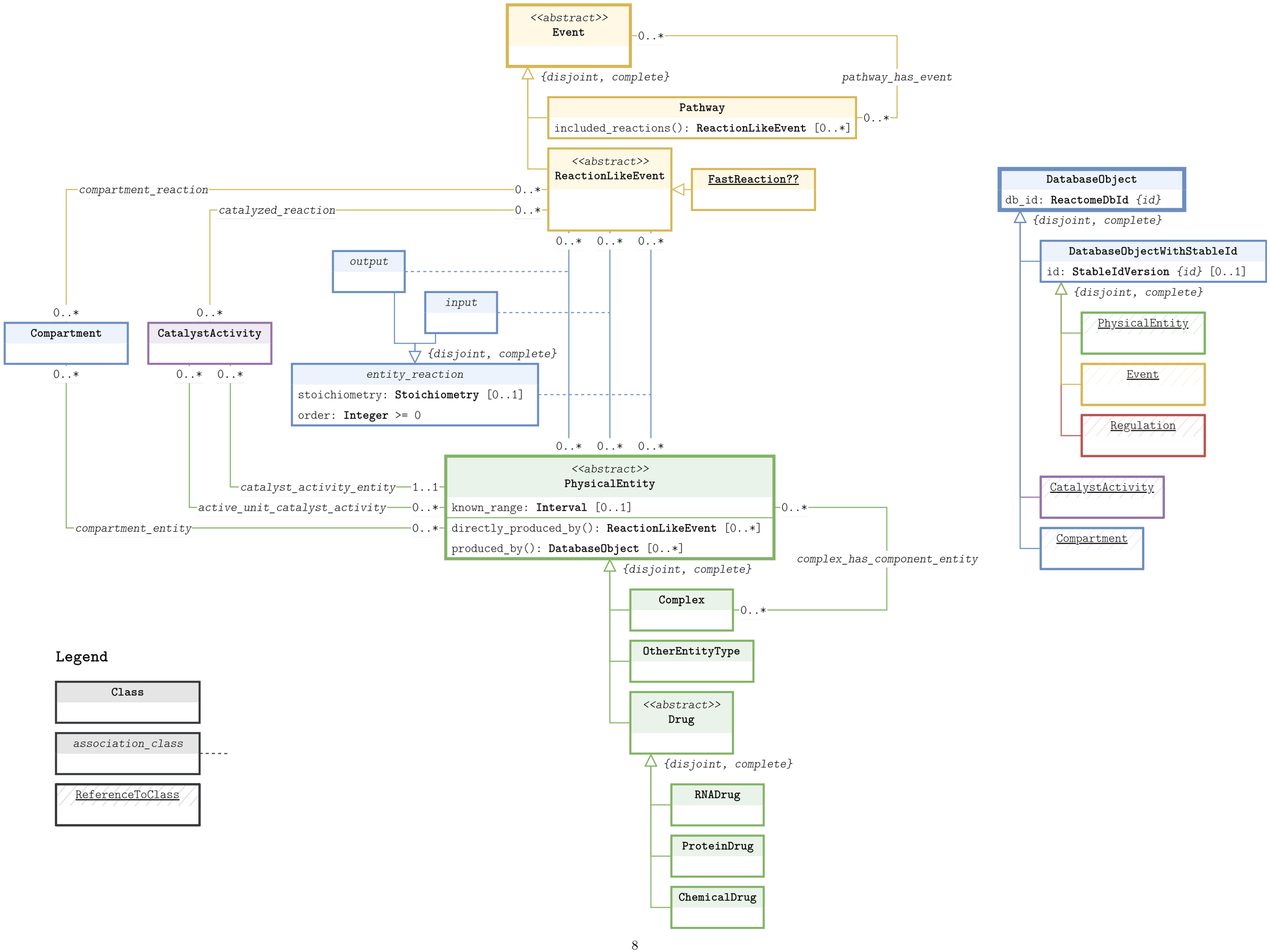
The `StableIdVersion` type is useful because is the one usually displayed in the Reactome Pathway Browser [4]. It is useful to accept it in the description of the models.

The `StableIdVersion` type is used to identify instances of `PhysicalEntity` or `Event` in Reactome, but it's pattern does not match the definition of `SId` used to identify objects in SBML.

In order to generate a correct `SBMLModel` the `StableId` must be converted.

```
StableIdVersion_into_SId(st_id: StableId): SId
  POSTCONDITIONS:
    . . .
```

3 *(Reactome)* UML class diagram



Legend

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).

Ignoring the *`inferred_to`* association there are about 34 top level pathways.

TODO: handle *`inferred_to`*

```
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

TODO: how should I handle complexes here?

The reactions which directly have `this` as a product.

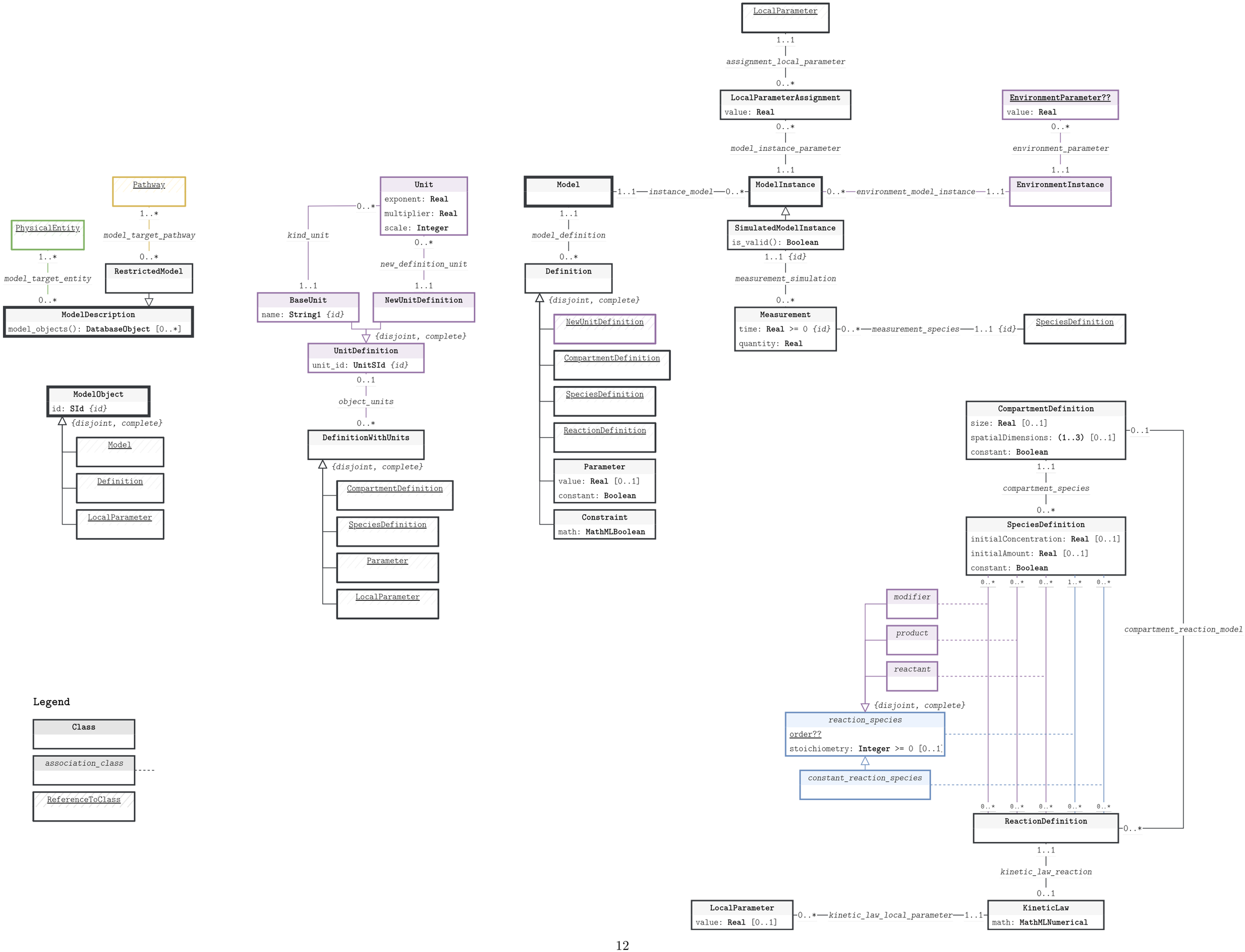
```
directly_produced_by(): ReactionLikeEvent [0..*]
  POSTCONDITIONS:
    result = { reaction |
        ReactionLikeEvent(reaction) ∧ output(this, reaction)
    }
```

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)
        }
```

TODO: handle active units too

5 *(Simulation)* UML class diagram



Legend

6 Classes specification pt. 2

6.1 CompartmentDefinition

```
[C.CompartmentDefinition.entities_have_compartment_listed]
  ∀ compartment_Definition, compartment, species, physical_entity
      (
          CompartmentDefinition(compartment_instance) ∧
          Compartment(compartment) ∧
          SpeciesDefinition(species) ∧
          PhysicalEntity(physical_entity) ∧
          compartment_Definition(compartment, compartment_instance) ∧
          compartment_species(compartment_Definition, species) ∧
          physical_entity_species(physical_entity, species)
      ) →
          compartment_entity(compartment, physical_entity)
```

TODO: what happens if it a PhysicalEntity has some compartments?

6.2 ModelDescription

TODO: add possibility to specify KineticLaw for each reaction, or a subset of reactions

```
model_objects(): DatabaseObject [1..*]
  POSTCONDITIONS:
    result = { object | ∃ entity
        PhysicalEntity(entity) ∧
        DatabaseObject(object) ∧
        model_target_entity(this, entity) ∧
        produced_by(entity, object) ∧
        (
            ¬ RestrictedModel(this) ∨
            ∃ pathway, reaction
                Pathway(pathway) ∧
                ReactionLikeEvent(reaction) ∧
                included_reactions(pathway, reaction) ∧
                (
                    object = reaction ∨
                    entity_reaction(object, reaction) ∨
                    catalyzed_reaction(object, reaction)
                )
        )
    }
```

6.3 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
              )
```

6.4 SimulatedModelInstance

```
is_valid()
  POSTCONDITIONS:
    . . .
```

6.5 Measurement

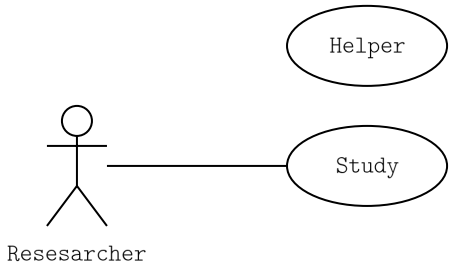
```
[C.Measurement.species_in_model]
  ∀ measurement, model_instance, model, species
      (
          Model(model) ∧
          SimulatedModelInstance(model_instance) ∧
          Measurement(measurement) ∧
          Species(species) ∧
          measurement_species(measurement, species) ∧
          measurement_simulation(measurement, model_instance) ∧
          instance_model(model_instance, model)
      ) →
          model_definition(model, species)
```

6.6 UnitDefinition

[3, page 45]

TODO: better description

7 Use-case diagram



7.1 “Helper” use-case

`generate_model(description: ModelDescription): Model`

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 $\vee$ automatic, like `LawOfMassAction`)
  - add local parameters
- create constraints
  - i.e. from known_range attribute
-

`instantiate_model(model: Model): ModelInstance`

POSTCONDITIONS:

TODO:

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

```
simulate_model(instance: ModelInstance): SimulatedModelInstance
```

POSTCONDITIONS:

TODO:

- generate measurements

7.2 "Study" use-case

```
describe_model(
  target_entities: PhysicalEntity [1..*]
  target_pathways: Pathway [0..*]
): ModelDescription
```

PRECONDITIONS:

target species are within the target pathways

POSTCONDITIONS:

return a model description

```
evaluate(model: Model): VirtualPatient [0..*]
```

POSTCONDITIONS:

run algorithm at page 4

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