Contents

1 Introduction **2**

2 Reactant hierarchy **2**
2.1 UML class diagram . . . 2
2.2 Reactant . . . 2
2.3 Chemical . . . 2
2.4 BoundChemical . . . 3
2.5 ProcessiveChemical . . . 3
2.6 Loader . . . 3
2.7 ChemicalSequence . . . 4
2.8 BindingSiteFamily . . . 4

3 Reaction hierarchy **4**
3.1 UML class diagram . . . 4
3.2 Reaction . . . 4
3.3 ChemicalReaction . . . 5
3.4 Complexation . . . 5
3.5 SequenceBinding . . . 6
3.6 Translocation . . . 6
3.7 Loading . . . 7
3.8 Release . . . 8
3.9 Degradation . . . 9

4 Solver hierarchy **9**

1 Introduction

The aim of this document is not to go into technical details of the implementation (the code is documented using Doxygen, technical details are therefore best found in the Doxygen-generated manual). Rather we wish to walk through the choices in design that have been made. The technical manual hopefully contains necessary information for understanding the concept behind each class and how to use it. However it does not tell you what classes are central in the architecture and it is difficult to see at a glimpse how classes interact.

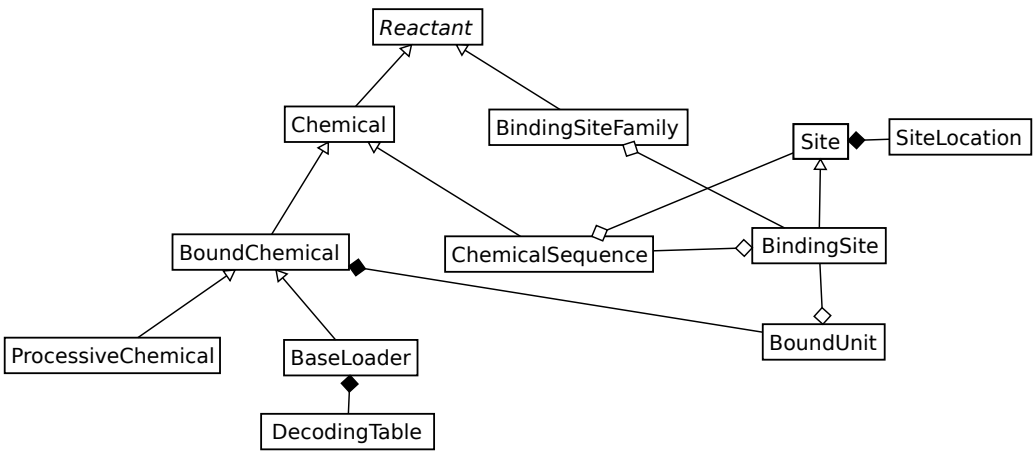
Describing global design should be more agreeable to read than the purely technical document. It helps understand how we tackled a certain number of efficiency issues (should it be speed or maintainability). Most efficiency issues in programming are related to architecture design rather than class implementation. Efficiency of an architecture can be related to information flowing between classes. Restricting information access through classes and dependencies between classes is generally considered good style, as it limits data corruption and enhances maintainability. Achieving this increases the probabilities that the simulator behaves the way it should and facilitates further expansions.

Once data protection and maintainability are ensured, speed issues are addressed only if they can be identified. Most parts of the simulator are not critical in that regard and do not need a particularly refined design or implementation. If speed issues arise, two level of solutions can be worked on. At the lowest level, class implementations can be changed to perform some routines more quickly (generally leading to at most a couple-fold speed increase). At the highest level, communication between classes can be tuned to ensure that only the necessary computations are done (generally leading to a drastic speed increase and a complexification of the architecture with new classes that "filter" communications).

To sum up, description at a global level gives critical insight into how the simulator works and where speed/design issues have been identified during development. It should facilitate discussions even with non-programmers (or at least non-C++-programmers).

2 Reactant hierarchy

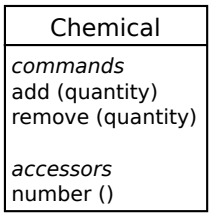
2.1 UML class diagram



2.2 Reactant

`Reactant` is a global abstract interface. All entities that can participate in a reaction *must* inherit from it.

2.3 Chemical

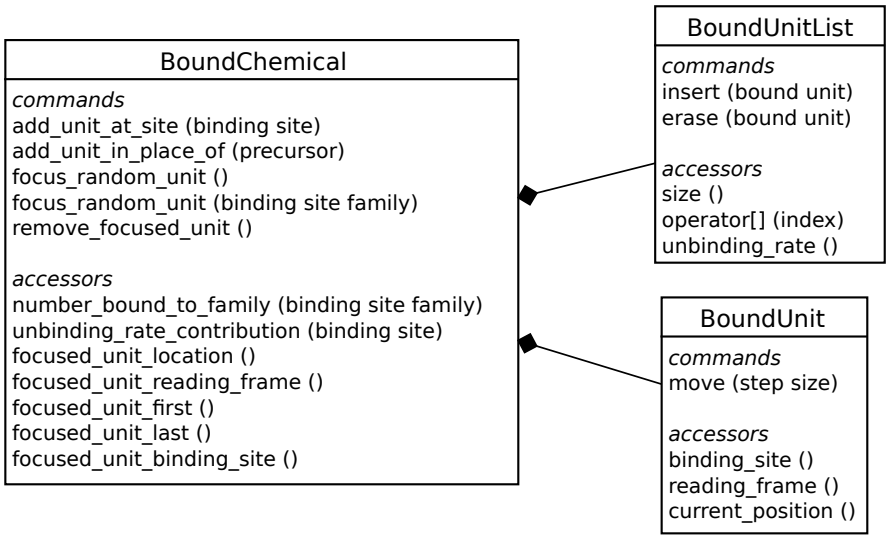


`Chemical` defines all standard chemical entities. Instances of the class (as opposed to instances of subclasses) are considered to be free chemical (*e.g.* molecules diffusing in the cytosol or extracellular medium). `Chemical` represents a *pool* of a given chemical species, meaning that one can add/remove a given quantity of it, and access its current number at any time.

Input format

```
unit Chemical <name> [<initial quantity>]
```

2.4 BoundChemical

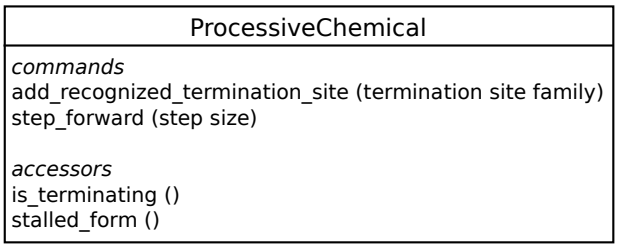


`BoundChemical` is a subclass of `Chemical` that represents chemicals that are bound to a sequence. It is important to note it only represents molecules bound to the sequence, *not* the complex formed by the chemical and the sequence. Even though `BoundChemical` represents a pool of molecules, single elements are not interchangeable, they are defined by their position on a sequence. `BoundChemical` uses two classes `BoundUnit` and `BoundUnitList` to represent molecules individually. An external user cannot access the latter directly, he can only focus a random unit of the pool.

Input format

```
unit BoundChemical <name>
```

2.5 ProcessiveChemical

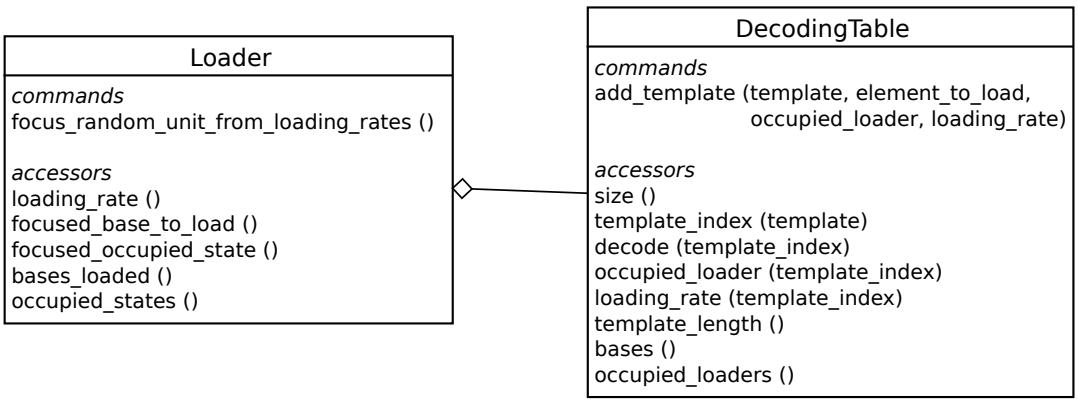


`ProcessiveChemical` is a subclass of `BoundChemical`. Compared to a standard `BoundChemical`, it is able to move along the sequence and recognize a termination site. A stalled form is associated with it. It is used when the chemical meets a termination site or the end of the sequence and cannot move anymore.

Input format

```
unit ProcessiveChemical <name> <stalled form> <termination site>
```

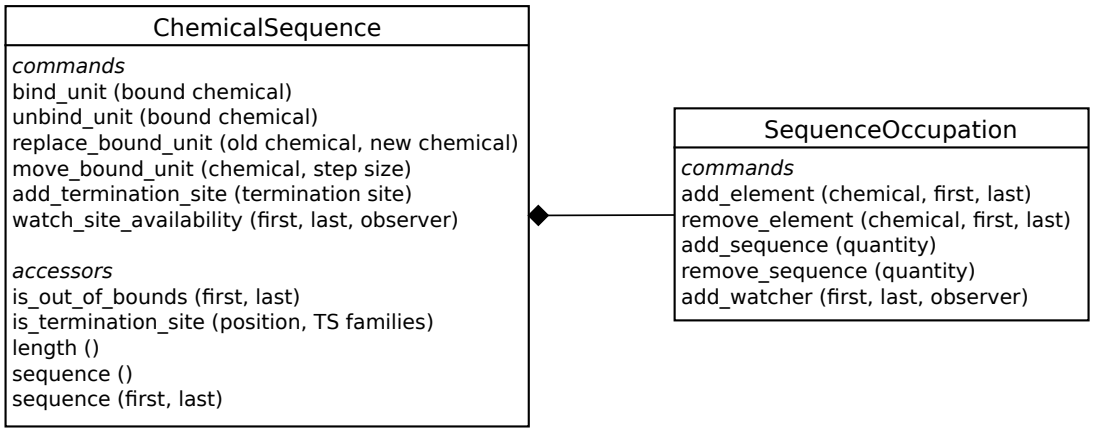
2.6 Loader



`Loader` is a subclass of `BoundChemical`. Compared to a standard `BoundChemical`, it is able to read templates on the sequence and load elements matching the sequence. The template element matching is done using a `DecodingTable`, which also specifies the state of the loader in its occupied form (when an element has effectively been loaded on its template) and the loading rate for every template-element pair.

Input format

```
unit Loader <name> <decoding table>
unit DecodingTable <name> \
  [<template> <element to load> <element-loader complex> <loading rate>]^N
```



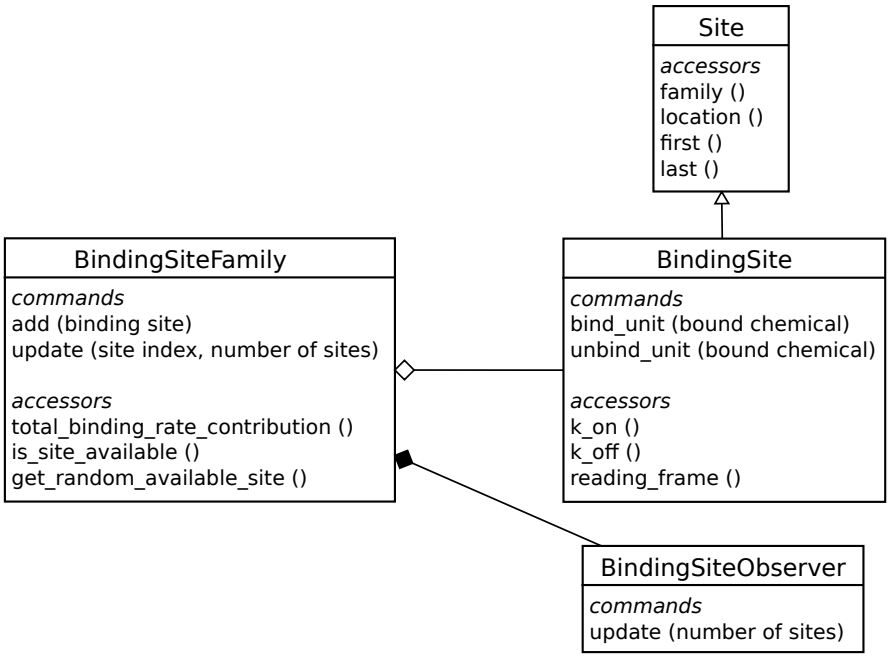
2.7 ChemicalSequence

`ChemicalSequence` is a subclass of `Chemical`. It is defined by a sequence and the ability to bind elements (more precisely `BoundChemical`). However, instances of a sequence are *not* treated individually, it is impossible to tell to which instance a given chemical bound. An object called `SequenceOccupation` maintains occupation levels at sites of interest. For example, suppose the sequence is an mRNA carrying a ribosome binding site for the protein DnaA. The number of available sites is obtained by removing the number of bound chemicals occupying the site from the number of instances of the mRNA currently in the cell.

Input format

```
unit ChemicalSequence <name> sequence <sequence> [<initial quantity>]
unit ChemicalSequence <name> product_of <parent sequence> \
  <starting position> <ending position> <product table> [<initial quantity>]
```

2.8 BindingSiteFamily



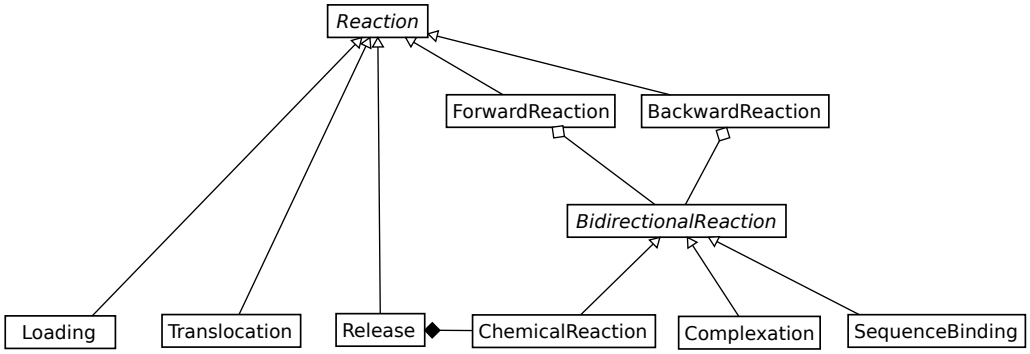
`BindingSiteFamily` is a subclass of `Reactant`. Contrary to `Chemical`, it does not represent a countable pool of molecules. Each family contains a number of related instances of `BindingSite`. A `BindingSiteObserver` is used to dynamically maintain the number of available sites for each binding site.

Input format

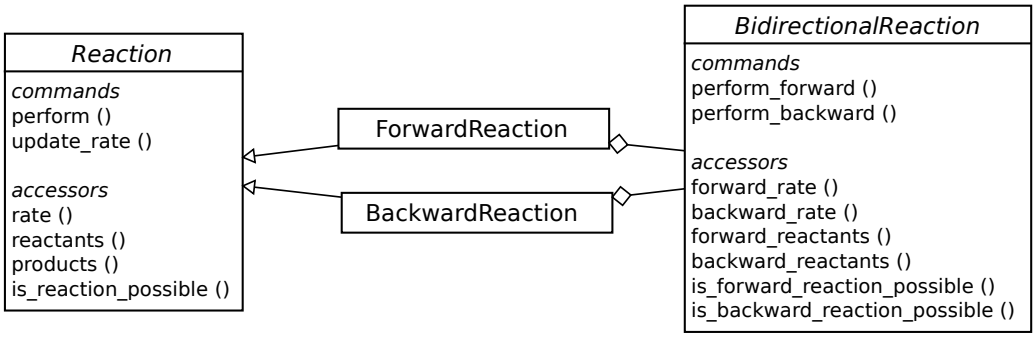
```
unit BindingSite <binding site family name> <chemical sequence> \
  <start> <end> <k_on> <k_off> [<reading frame>]
```

3 Reaction hierarchy

3.1 UML class diagram



3.2 Reaction



There are two abstract classes used to define reactions: `Reaction` for one-way reactions and `BidirectionalReaction` for reversible reactions. Two adapter classes `ForwardReaction` and `BackwardReaction` split reversible reactions in two one-way reactions. In the end, the solver only handles one-way reactions. A reaction can necessarily be performed, its rate updated and accessed and is composed of reactants and products.

3.3 ChemicalReaction

Input format

```
reaction ChemicalReaction [<chemical> <stoichiometry>]^N rates <k_1> <k_-1>
```

Formula A `ChemicalReaction` represents association/dissociation of an arbitrary number of elements. It is defined by

$$a_1 A_1 + a_2 A_2 + ... + a_r A_r \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} b_1 B_1 + ... + b_p B_p$$

where

- A_i and B_i are of type `Chemical`. They *must not* be of type `BoundChemical`, *except* if there is exactly one `BoundChemical` on each side of the equation.
- a_i and b_i are stoichiometric coefficients.
- k_1 and k_{-1} are rate constants.

Action When the reaction is performed, the number of chemicals involved is changed according to their stoichiometric coefficient. If `BoundChemical` are involved, the simulator will assume that the bound chemical that is consumed is replaced by the bound chemical on the other side of the equation (*i.e.* it will be bound at the location previously occupied by the precursor).

Rate The rates are given by

$$\lambda_{forward} = k_1 \prod_{i=1}^{r} [A_i]^{a_i}$$

$$\lambda_{backward} = k_{-1} \prod_{i=1}^{p} [B_i]^{b_i}$$

3.4 Complexation

Input format

```
reaction Complexation [<chemical> <stoichiometry>]^N rates <k_1> <k_-1>
```

Formula A `Complexation` represents association/dissociation of two elements. It is defined by

$$A + B \underset{k_{off}}{\overset{k_{on}}{\rightleftharpoons}} C$$

where

- A, B and C are of type `Chemical`. They *must not* be of type `BoundChemical`, *except* if there is exactly one `BoundChemical` on each side of the equation. In the latter case, the simulator will assume that the bound chemical that is consumed is replaced by the bound chemical on the other side of the equation (*i.e.* it will be bound at the location previously occupied by the precursor).
- k_{on} and k_{off} are rate constants.

Action When the reaction is performed, the number of chemicals involved is increased/decreased by 1. If `BoundChemical` are involved, the simulator will assume that the bound chemical that is consumed is replaced by the bound chemical on the other side of the equation (*i.e.* it will be bound at the location previously occupied by the precursor).

Rate The rates are given by

$$\lambda_{forward} = k_{on}[A][B]$$
$$\lambda_{backward} = k_{off}[C]$$

3.5 SequenceBinding

Input format

```
reaction SequenceBinding <chemical> <bound form> <binding site family>
```

Formula A `SequenceBinding` represents binding of a free element on a binding site of a sequence. It is defined by

$$C_{free} + BSF \rightleftharpoons C_{bound}$$

where

- C_{free} is of type `Chemical` but not `BoundChemical`.
- BSF is of type `BindingSiteFamily`.
- C_{bound} is of type `BoundChemical`.

Action When the forward reaction is performed, a random available binding site is drawn from the binding site family (drawing is weighted by affinity). A C_{free} molecule is removed from the pool and a C_{bound} added to the `ChemicalSequence` bearing the binding site. When the backward reaction is performed, a random molecule of C_{bound} is removed from the pool (and from its sequence) and a C_{free} molecule is added.

Rate The rates are given by

$$\lambda_{forward} = \frac{[C_{free}]}{V_c} \sum_{\text{sites } s \in BSF} (k_{on})_s \times \text{Number of sites } s \text{ available}$$

$$\lambda_{backward} = \frac{1}{V_c} \sum_{\text{molecules } m \in C_{bound}} (k_{off})_{\text{site on which } m \text{ is bound}}$$

- $(k_{on})_s$ is the association constant of C_{free} with binding site s.
- $(k_{off})_s$ is the dissociation constant of C_{bound} with binding site s.
- V_c is the volume of the cell.

3.6 Translocation

Input format

```
reaction Translocation <processive chemical> <form after step> <step size> <rate>
```

Formula A `Translocation` represents movement of a bound element along a sequence. It is defined by

$$C \xrightarrow{k} C_{\text{after step}}$$

where

- C is of type `ProcessiveChemical`.
- $C_{\text{after step}}$ is of type `BoundChemical`.
- k is a rate constant.

Action When the reaction is performed, a random C is chosen. Generally, it is replaced by a $C_{\text{after step}}$, moved by a step of a given size along the sequence the original C is bound to. If the chemical cannot move because it reached the end of the sequence or it reaches a termination site, it is replaced by the stalled form of the `ProcessiveChemical` C.

Rate The rate is given by

$$\lambda = k[C]$$

3.7 Loading

Input format

```
reaction Loading <loader>
```

Formula A `Loading` typically represents loading of elements by a polymerase onto a template sequence. It is defined by

$$L + E \longrightarrow LE$$

where

- L is of type `Loader`.
- E is an element to load, of type `Chemical` but not `BoundChemical`. It is defined in L's `DecodingTable`.
- LE is the occupied form of the loader, of type `BoundChemical`. It is defined in L's `DecodingTable`.

Action Each instance of L reads a specific template. Using its `DecodingTable`, we know which E it tries to load, which LE is yielded if loading occurs and the loading rate associated with the template. When the reaction is performed, a random L is chosen according to loading rates. An element to load E is removed from the pool and L is replaced with LE.

Rate The rate is given by

$$\lambda = \sum_{t \in templates} k_t[L_t][E_t]$$

where

- k_t is the loading rate associated with template t.
- L_t corresponds to loaders L reading template t.
- E_t is the chemical to load onto template t.

3.8 Release

Input format

```
reaction Release <bound chemical> [<chemical> <stoichiometry>]^N rate <rate> \
  [produces <product table>]
```

Formula A `Release` represents detachment of a bound element from a sequence. An arbitrary number of elements can be used as coreactants to trigger the release, and an arbitrary number of products are released. It is defined by

$$B + a_1A_1 + ... + a_rA_r \xrightarrow{k} \delta_P P + c_1C_1 + ... + c_pC_p$$

where

- B is of type `BoundChemical`.
- P is of type `ChemicalSequence`. It is a product that is released if B is a polymerase. In the latter case $\delta_P = 1$ else $\delta_P = 0$.
- A_i are of type `Chemical`, but not `BoundChemical`.
- C_i are of type `Chemical`, but not `BoundChemical`.
- a_i and c_i are stoichiometric coefficient.
- k is a rate constant.

Action When the reaction is performed, a random B is chosen and removed from the pool (and detached from its sequence). The A_i and C_i pool are changed according to their stoichiometric coefficients. If a product table is defined, it means that the released chemical is a polymerase. A product (a chemical sequence) corresponding to the bound chemical's binding and release points is added.

Rate The rate is given by

$$\lambda = k[B] \prod_{i=1}^{r} [A_i]^{a_i}$$

3.9 Degradation

Input format

```
unit CompositionTable <name> <letter_1> [<chemical>]^N
...
unit CompositionTable <name> <letter_N> [<chemical>]^N

reaction Degradation <chemical sequence> <composition table> <rate>
```

Formula A `Degradation` represents decomposition of a sequence into base components. It is defined by

$$CS \xrightarrow{k} b_1 + b_2 + ... + b_N$$

where

- CS is of type `ChemicalSequence`.
- b_i are of type `Chemical`, but not `BoundChemical`. They are found in a `CompositionTable` specified in the reaction.
- k is the degradation constant.

Action When the reaction is performed, a CS is removed from the pool. A `CompositionTable` is specified along the reaction. It allows base-by-base conversion of the sequence of CS into components yielded by degradation. The pools of base components is updated accordingly. In the simulator, a degradation reaction is effectively implemented as a `ChemicalReaction`.

Rate The rate is given by

$$\lambda = k[CS]$$

4 Solver hierarchy