Macromolecules

Initial comments

MCell's current (2009-04-24) implementation of macromolecules is a fairly restricted model, though one which is useful for a number of different applications. First, a few terms as they are used in this document.

macromolecule A macromolecule is a specially defined molecular species. When placed in the world, a macromolecule is a collection of a fixed number of *subunits*, each of which is a normal (albeit stationary) volume or surface molecule. The term *complex* may be used interchangeably with *macromolecule* in this document.

complex Synonymous with macromolecule.

subunit A subunit is any molecular species when it is used as a part of a macromolecule. At present, a macromolecule must be composed either entirely of surface molecule species, or of volume molecule species.

volume macromolecule A volume macromolecule is a macromolecule whose subunits are of volume molecule species.

surface macromolecule A surface macromolecule is a macromolecule whose subunits are of surface molecule species.

macromolecular reaction A macromolecular reaction is a reaction with one of the subunits of a macromolecule, whose rate may or may not be affected by the states of the other subunits in the macromolecule.

complex reaction rate A complex reaction rate is a reaction rate which depends on the states of one or more subunits in the macromolecule.

reference subunit The "reference" subunit will refer to the subunit undergoing a reaction, as opposed to all of the other subunits in the macromolecule.

related subunits The "related" subunits will refer to the set of all subunits which are reachable from the reference subunit by one or more of the relations defined in the macromolecule.

inverse-related subunits The "inverse-related" subunits will refer to the set of all subunits from which the reference subunit is reachable by one or more of the relations defined in the macromolecule. In the motivating example I've provided, the set of related subunits for any given reference subunit is exactly equal to the set of inverse-related subunits, but this is only because the set of relations defined in this example are complete under inversion.

unrelated subunits "Unrelated" subunits will refer to the set of all subunits which are neither the reference subunit, a related subunit, or an inverse-related subunit.

Logically, a macromolecule's subunits are a n-dimensional collection. This does not imply anything about the physical layout of the macromolecule, only the logical topology. The motivating example I will present in this document is a 2-dimensional macromolecule which is a 2x6 collection of subunits which are logically a 6-ring of dimer pairs.

Presently, macromolecules must be stationary. Once they have been placed, they will not diffuse or move under any circumstances. At some point, we may undertake to implement diffusion of macromolecules, but there are no such plans at present.

In addition to the subunits, macromolecules will show up in certain types of output. They are a species like any other and can be displayed in visualization and count output. The major difference is that the macromolecule species may not itself undergo reactions, and if it is a surface molecule, it will not occupy a tile.

Motivating example

Here is an excerpt from an MCell macromolecules test case which omits everything not related to macromolecules. Briefly, it describes a macromolecule called `my_complex` which has 12 subunits which may be in an unbound (`subunitU`) or a bound (`subunitB`) state. The 12 subunits are arranged into 6 dimers, and *cooperativity* is exhibited, in that binding rates are higher for subunits whose dimer partner is bound, and unbinding rates are lower for doubly-bound dimer pairs:

```
/* Define subunit types */
DEFINE_MOLECULES {
  subunitU { DIFFUSION_CONSTANT_3D = 0 }   /* unbound */
  subunitB { DIFFUSION_CONSTANT_3D = 0 }   /* bound */
  Lig      { DIFFUSION_CONSTANT_3D = 2e-6 }
}

/* Define the complex type */
DEFINE_COMPLEX_MOLECULE my_complex {

  /* Define the dimensionality: 2x6 */
  NUMBER_OF_SUBUNITS = [2, 6]

  /* Set the initial states of all subunits. */
  SUBUNIT[1:2, 1:6] = subunitU

  /* Set the physical layout of the macromolecule */
  SHAPE {
    SUBUNIT[1, 1] = [-.10,  .0000,  .05]
    SUBUNIT[1, 2] = [-.05,  .0866,  .05]
    SUBUNIT[1, 3] = [ .05,  .0866,  .05]
    SUBUNIT[1, 4] = [ .10,  .0000,  .05]
    SUBUNIT[1, 5] = [ .05, -.0866,  .05]
    SUBUNIT[1, 6] = [-.05, -.0866,  .05]
    SUBUNIT[2, 1] = [-.10,  .0000, -.05]
    SUBUNIT[2, 2] = [-.05,  .0866, -.05]
    SUBUNIT[2, 3] = [ .05,  .0866, -.05]
    SUBUNIT[2, 4] = [ .10,  .0000, -.05]
    SUBUNIT[2, 5] = [ .05, -.0866, -.05]
    SUBUNIT[2, 6] = [-.05, -.0866, -.05]
  }

  /* Specify the relationships between the subunits */
  SUBUNIT_RELATIONSHIPS {
    ring_negative = [ 0, -1]
    ring_positive = [ 0, +1]
    dimer_partner = [+1,  0]
  }

  /* Specify the rules for reaction rates. */
  RATE_RULES {
    binding {
      dimer_partner != subunitU    : fw_rate * TMP_spec_F
      DEFAULT                      : fw_rate
    }
    unbinding {
      dimer_partner != subunitU    : bw_rate * TMP_spec_B
      DEFAULT                      : bw_rate
    }
    /* Unused example of multiple clauses */
    triplet {
      ring_positive != subunitU &
      ring_negative != subunitU    : 1e7
      DEFAULT                      : 1e6
    }
  }
}

DEFINE_REACTIONS {
  (subunitU) + Lig <-> (subunitB) [> COMPLEX_RATE my_complex binding,
                                   < COMPLEX_RATE my_complex unbinding]
}

REACTION_DATA_OUTPUT
{
  OUTPUT_BUFFER_SIZE = 1000
  STEP = 1e-5
  HEADER = "# "
  {
    COUNT[subunitU, WORLD]                             : "us",
    COUNT[subunitB, WORLD]                             : "bs",
    COUNT[SUBUNIT{
         my_complex : subunitU [
              dimer_partner == subunitU
         ]}, WORLD]                                    : "00",
    COUNT[SUBUNIT{
         my_complex : subunitU [
              dimer_partner != subunitU
         ]}, WORLD]                                    : "01",
    COUNT[SUBUNIT{
         my_complex : subunitB [
              dimer_partner == subunitU
         ]}, WORLD]                                    : "10",
    COUNT[SUBUNIT{
          my_complex : subunitB [
              dimer_partner != subunitU
          ]}, WORLD]                                  : "11",
    COUNT[SUBUNIT{
          my_complex : subunitB [
              ring_negative != subunitU &
              ring_positive != subunitU
          ]}, WORLD]                                  : "triple",
    COUNT[Lig, WORLD]                                 : "Lig"
  }  => countdir & "counts.dat"
}
```

In English, this example describe a macromolecule which consists of a 6-ring of dimer pairs. Each subunit in the molecule is initially of species `subunitU`. Because the subunits are volume molecules, the macromolecule as a whole is a volume macromolecule. The physical layout of the subunits is that of two stacked hexagonal rings. Only one (bidirectional) reaction is defined for the subunits of the macromolecule -- a "binding" reaction which binds `Lig` to turn `subunitU` into `subunitB`. An essential feature of this reaction is *cooperativity*, meaning that if one subunit in a dimer pair is bound, the binding rate increases for the other subunit. Similarly, the binding rate is decreased for doubly-bound dimer pairs. Finally, the reaction output from the simulation is specified to give counts of:

- The number of unbound subunits in the entire world
- The number of bound subunits in the entire world
- The number of doubly unbound dimer pairs
- The number of unbound subunits whose partners are bound
- The number of bound subunits whose partners are unbound
- The number of doubly bound dimer pairs
- The number of `Lig` molecules in the world

The syntax

Now, a quick walk through the syntax. I won't bother explaining the bits of syntax which are unrelated to macromolecules, such as the declaration of species for the subunits, which is indistinct from declaration of species for non-macromolecules.

First up is the structure of a macromolecule declaration:

```
DEFINE_COMPLEX_MOLECULE my_complex {
  /* Topology */
  /* Initial states */
  /* Physical layout */
  /* Relationships */
  /* Rate rules */
}
```

All of the sections listed above are currently required in a macromolecule, and they MUST occur in order at present. This may be relaxed in the future, but in order to properly error check the macromolecules, this ordering is currently required.

Topology

The topology is defined by the `NUMBER_OF_SUBUNITS` keyword:

```
/* Define the dimensionality: 2x6 */
NUMBER_OF_SUBUNITS = [2, 6]
```

This defines a 2x6 macromolecule (i.e. with 12 subunits). The topology may be a list of any length >= 1. One could, for instance, define a macromolecule which has only a single subunit, as any of the following:

```
NUMBER_OF_SUBUNITS = [1]
NUMBER_OF_SUBUNITS = [1, 1]
NUMBER_OF_SUBUNITS = [1, 1, 1, 1, 1, 1]
```

Obviously, there isn't much point to a unimolecular macromolecule, so it's more common for macromolecules to have at least 2 subunits. The order of the dimensions does not have any significance other than determining the required order of the coordinates used to select macromolecules. Topologically, the "coordinates" in each dimension wrap. In the above [2, 6] case, the following coordinate pairs are "adjacent", in the sense that they differ by exactly one in exactly one coordinate. This adjacency is not relevant to the functional attributes of the macromolecule, but it's important to understand the topology in order to understand the subunit relations, which will be discussed later:

```
* [x, 1] and [x, 2]
* [x, 2] and [x, 3]
* [x, 3] and [x, 4]
* [x, 4] and [x, 5]
* [x, 5] and [x, 6]
* [x, 6] and [x, 1]
* [1, y] and [2, y]
```

Initial states

The initial states of the subunits are set via one or more `SUBUNIT[c1, c2, ..., cN] = species` statements:

```
SUBUNIT[1:2, 1:6] = subunitU
```

The array of 2x6 subunits need an initial state, consisting of a species and, for surface macromolecules only, an orientation, which is always measured relative to the orientation of the macromolecule itself. Note that, as with normal surface molecule placement, surface subunits may be given an orientation of 0, meaning that each subunit with an orientation of 0 will be randomly oriented with respect to the macromolecule itself (and with respect to each other). There will be more on orientation later, in case this isn't clear enough.

The notation for setting initial states of subunits allows setting several subunits' initial states at once, as long as the region is a cartesian product of ranges. For an n-dimensional macromolecule, we need n coordinate ranges for each `SUBUNIT[c1, c2, ..., cn] = species` statement, and can have an arbitrary number of such statements, which are processed in the order they are seen in the file. This way, we can set them all to some default species (such as `subunitU`), and then change only selected items:

```
SUBUNIT[1:2, 1:6] = subunitU
SUBUNIT[1:1, 2:4] = subunitB
SUBUNIT[2,   1:2] = subunitB
SUBUNIT[1:2, 6  ] = subunitB
```

Each coordinate range is in the form `low:high` or just `n` which is equivalent to `n:n`. The coordinate ranges are inclusive of both endpoints. So, the above example follows the following procedure:

1. Set all 2x6 subunits to `subunitU`
2. Set [1, 2], [1, 3], and [1, 4] to `subunitB`
3. Set [2, 1], and [2, 2] to `subunitB`
4. Set [1, 6], and [2, 6] to `subunitB`

The end result of this is that [1, 5], and [2, 3:5] are initialized to `subunitU` and the rest are initialized to the bound state.

Physical layout

The physical layout is contained in a single `SHAPE { }` section:

```
SHAPE {
  SUBUNIT[1, 1] = [-.10,  .0000,  .05]
  SUBUNIT[1, 2] = [-.05,  .0866,  .05]
  SUBUNIT[1, 3] = [ .05,  .0866,  .05]
  SUBUNIT[1, 4] = [ .10,  .0000,  .05]
  SUBUNIT[1, 5] = [ .05, -.0866,  .05]
  SUBUNIT[1, 6] = [-.05, -.0866,  .05]
  SUBUNIT[2, 1] = [-.10,  .0000, -.05]
  SUBUNIT[2, 2] = [-.05,  .0866, -.05]
  SUBUNIT[2, 3] = [ .05,  .0866, -.05]
  SUBUNIT[2, 4] = [ .10,  .0000, -.05]
  SUBUNIT[2, 5] = [ .05, -.0866, -.05]
  SUBUNIT[2, 6] = [-.05, -.0866, -.05]
}
```

Unlike in the initial states, the coordinates must be set for each subunit individually, and each subunit may be specified only once, though their specification may be in any order. MCell will yield an error if any coordinates are duplicated or omitted, or if any out-of-range coordinates are specified. The format here is fairly simple. Each subunit gets a vector which is its relative displacement from the *origin* of the macromolecule -- its location of placement. Note that these displacements are subject to various rotations. For a surface molecule, the displacements are 2-D displacements, ignoring the Z component, and the displacements are, after being randomly rotated around the surface normal, wrapped to the surface, akin to the wrapping that occurs for a 2-D diffusion step which crosses wall boundaries.

Relationships

The relationships will require a little more explanation, though they are, syntactically fairly simple:

```
SUBUNIT_RELATIONSHIPS {
  ring_negative = [ 0, -1]
  ring_positive = [ 0, +1]
  dimer_partner = [+1,  0]
}
```

This defines three relationships between subunits on the macromolecule. We only use one of these relationships in the example above (`dimer_partner`). Essentially, the relationships should describe “how to get to” any subunit from any other subunit to the degree that you need to reference the other subunit in rules or count statements. In the example above, the only molecule that can affect any given subunit (in regard to the reaction rates or counting) is the dimer partner. The other 10 molecules are irrelevant. Each relation defines an additive offset which is applied to a subunit's "coordinates" within the macromolecule to find the related subunit. Again, these coordinates wrap, so addition is done modulo the size of each dimension. For instance, for the subunit at [1, 4], `ring_negative` is [1, 3], `ring_positive` is [1, 5] and `dimer_partner` is [2, 4], whereas for [2, 6], `ring_negative` is [2, 5], `ring_positive` is [2, 1], and `dimer_partner` is [1, 6].

A quick note about relationships: It *may* be beneficial to performance to keep the number of relationships as small as is consistent with the desired semantics. For this example model, it is possible I should remove `ring_negative` and `ring_positive` if performance is an issue. (I keep them in for didactic purposes and to increase the coverage of the testing.) The reason for this is that when the reference subunit changes state, all inverse-related molecules will have their unimolecular reaction times recomputed if they may undergo unimolecular reactions which have *complex rates*, whether or not the particular rates depend on each particular relation. So, for this example, when the molecule [1, 4] changes state, even though only [2, 4] could have a changed unimolecular rate, any of [2, 4], [1, 3], and [1, 5] may have their reaction rates recomputed if they can undergo unimolecular reactions. The performance impact of this has not been carefully studied. Numerically, it is valid, because the unimolecular events are Poisson.

Rate rules

Rate rules are tables, not tied to any particular reaction, which allow specification of reaction rates based on the states of related subunits. Note as you read this that the rate rules do not mention the reference subunit. The reference subunit is only present in the reaction definition. This allows us to use the same reaction rate table for several different reactions. The reaction rate rules are converted into a table which is scanned one row at a time until a matching row is found. The implication is that the ordering of rules is critically important:

```
RATE_RULES {
  binding {
    dimer_partner != subunitU    : fw_rate * TMP_spec_F
    DEFAULT                      : fw_rate
  }
  unbinding {
    dimer_partner != subunitU    : bw_rate * TMP_spec_B
    DEFAULT                      : bw_rate
  }
  /* Unused example of multiple clauses */
  triplet {
    ring_positive != subunitU &
    ring_negative != subunitU    : 1e7
    DEFAULT                      : 1e6
  }
}
```

`binding` will look first at the `dimer_partner` of the reference subunit. If the partner is not in the `subunit` state, it will pick the first reaction rate, which is `fw_rate * TMP_spec_F`. Otherwise, it will proceed to the next rule. `DEFAULT` always matches, so no rule after the `DEFAULT` rule can ever be matched. `unbinding` is similarly straightforward. `triplet` demonstrates a rate rule which depends upon the states of multiple subunits at once, though this rule is never used.

One minor twist is worth mentioning for oriented subunits. For `==` rules, the matching is straightforward. That is, each of:

- `relation == species,`
- `relation == species'`

- `relation == species;`

will work as expected. `!=` will match if either the species or orientation does not match. That is, `relation != species,` will match if either the `relation` is not of species `species`, or if the orientatation of `relation` is not negative. This is logical but easy to overlook. Thus, if the goal is to check only the orientation, it must be done in two steps:

1. `relation != species;` (check if the species does not match)
2. `relation != species,` (check if the orientation does not match)

Reactions

Macromolecule reactions are just like any other MCell reactions except for the fact that exactly one reactant and exactly one product **must** be specified as subunits. This is done by surrounding the reactant or product (orientation and all) with parentheses:

```
DEFINE_REACTIONS {
  (subunitU) + Lig <-> (subunitB) [> COMPLEX_RATE my_complex binding,
                                   < COMPLEX_RATE my_complex unbinding]
}
```

In the above, `subunitU` and `subunitB` are subunits. Furthermore, in a macromolecular reaction, the rate may be specified as in a normal reaction -- either as a fixed rate or a time-varying tabular rate to be loaded from a file, but it may also be specified as a complex rate. Either or both of the forward and reverse rates may be specified as complex. The syntax for a complex rate specification is:

```
COMPLEX_RATE <complex> <rate_table>
```

where `<complex>` gives the name of the complex species, and `<rate_table>` gives the name of the specific rate table within the complex.

Counting

Counting uses the familiar syntax from the rate rule tables, but each count statement stands on its own, and thus, the ordering effects which are possible with rate tables are not possible for counting. As with rate rules, each related subunit may appear at most once:

```
REACTION_DATA_OUTPUT
{
  OUTPUT_BUFFER_SIZE = 1000
  STEP = 1e-5
  HEADER = "# "
  {
    COUNT[subunitU, WORLD]                                  : "us",
    COUNT[subunitB, WORLD]                                  : "bs",
    COUNT[SUBUNIT{
         my_complex : subunitU [
              dimer_partner == subunitU
         ]}, WORLD]                                         : "00",
    COUNT[SUBUNIT{
         my_complex : subunitU [
              dimer_partner != subunitU
         ]}, WORLD]                                         : "01",
    COUNT[SUBUNIT{
         my_complex : subunitB [
              dimer_partner == subunitU
         ]}, WORLD]                                  : "10",
    COUNT[SUBUNIT{
          my_complex : subunitB [
              dimer_partner != subunitU
          ]}, WORLD]                                 : "11",
    COUNT[SUBUNIT{
          my_complex : subunitB [
              ring_negative != subunitU &
              ring_positive != subunitU
          ]}, WORLD]                                 : "triple",
    COUNT[Lig, WORLD]                                : "Lig"
  }  => countdir & "counts.dat"
}
```

The syntax of a macromolecular count is:

```
COUNT[SUBUNIT{<complex> : <refsubunit> [<rules>]}, <location>]
```

where:

- `<complex>` is the name of the complex species to which this count applies[1]
- `<refsubunit>` is the molecular species for the reference subunit for this count
- `<rules>` are a series of clauses, just as in the rate rules tables, which specify the required states of each related subunit.

At least one rule must be specified if macromolecular counting is used[2]. The macromolecular counts above (`00`, `01`, `10`, `11`, and `triple`) count, respectively:

00 The number of unbound subunits whose dimer partner is unbound (i.e. twice the number of doubly-unbound dimer pairs).

01 The number of unbound subunits whose dimer partner is bound

10 The number of bound subunits whose dimer partner is unbound (== 01)

11 The number of bound subunits whose dimer partner is bound (i.e. twice the number of doubly-bound dimer pairs).

triple The number of bound subunits whose neighbors along the 6-ring are both bound.

Surface molecules

Beyond the syntax details listed above, there are a few details worth mentioning for surface molecules.

1. Placement: Each macromolecule subunit will occupy its own tile, and the tile will be found by tracing a path along the surface, wrapping across edges just as surface diffusion does.

   Obviously, surface macromolecule placement may fail if the search radius is too small and the phyical layout of the macromolecule is not much larger than (or is smaller than) the size of a tile.

2. Surface class: Macromolecules may not be placed on a surface via `SURFACE_CLASS` statements at present. There are some mildly confusing semantic issues to work through here.

3. Orientation: Surface macromolecule orientation is relative to the surface upon which it is placed. Surface macromolecule subunit orientation is relative to the orientation of the macromolecule to which it belongs. Thus, when a surface macromolecule is placed, if it is placed with "negative" orientation, the orientations of all of its subunits *with respect to the surface* will be inverted. This may benefit from a little explanation.

   The release statement in the following code block will cause the placement of roughly 500 `example'` macromolecules, and roughly 500 `example,` macromolecules upon the surface of `world.box`. Each `example'` macromolecule will have subunits `[1:2, 1:3]` oriented outward with respect to the surface, and `[1:2, 4:6]` oriented inward with respect to the surface. Each `example,` macromolecule will have exactly the opposite situation. The end result is that each macromolecule will have 6 outwards and 6 inwards subunits, and we will always have exactly 6*1000 subunits facing outwards and exactly 6*1000 facing inwards, even though the macromolecules themselves are randomly oriented with respect to the surface:

```
DEFINE_COMPLEX_MOLECULE example {
    // ..
    SUBUNITS[1:2, 1:3] = subunitU'
    SUBUNITS[1:2, 4:6] = subunitU,
    // ..
}

INSTANTIATE world OBJECT {
    box BOX {
        CORNERS = [0, 0, 0], [0.5, 0.5, 0.5]
    }
    rs RELEASE_SITE {
        SHAPE = world.box[ALL]
        MOLECULE = example;
        NUMBER_TO_RELEASE = 1000
    }
}
```

Footnotes

[1] If more than one macromolecule type incorporates the same subunit species, the count will NOT include all of the subunits which belong to the other macromolecule species.

[2] This restriction may be loosened in the future, if it seems worthwhile.