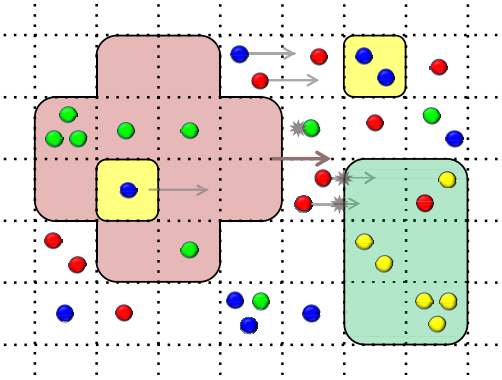
ML-Space: A quick start guide



Arne Bittig

Contents

1 Introduction

ML-Space is a rule-based modelling language developed for, but not limited to, cell biological systems with a simulator for particle-based and reaction-diffusion simulation. This document is a quick-start

guide for the stand-alone version of ML-Space[1] and is work in progress. Regarding feature requests, bugs or unclear documentation, please contact arne.bittig@informatik.uni-rostock.de.

1.1 Requirements

A Java Runtime Environment is needed (version >= 1.7). The ML-Space sandbox should run on any OS with Java support, but was tested on Windows 7 only so far.

1.2 License

The current version is for testing and reviewing purposes only. It is provided "as is", without warranties of any kind.

ML-Space will be released with a future version of JAMES II (bitbucket.org/jamesii/main) under the JAMES II license.[2] Some dependencies use other licenses, including the Apache license (`Ini4j`) and the MIT license (`args4j`).

2 Creating an ML-Space Model

For a start, ML-Space models are probably best written by analogy to the provided example models, `LipidRafts.mls`, `MitoQualityAndFissionCMSB.mls` (from [Bit+14b]) and `ActinModelBMC.mls` (from [Bit+14a] with updated syntax; The latter two make use of bindings between entities, which so far is useful primarily if bound entities are immobile.)

Since use of ML-Space outside the group that developed it is still limited, this document may not cover everyone's needs. Do not hesitate to contact the author (arne.bittig@informatik.uni-rostock.de or arne.bit@gmail.com).

2.1 Species, Rules, Init Step by Step

ML-Space model entities are of different species (molecule types) and may have attribute values (configurations).[3]

Entities may have a spatial extension, in which case they are potential compartments, i. e., may contain other entities within them, or may be dimensionless, i. e., to be simulated in a population based, reaction-diffusion-in/between subvolumes manner. (Which category each entity belongs is species-dependent.) The first important part is thus the species definition, indicating the types of entities in the model and the attributes they have.

```
1 Raft(shape:circle, size:3.14*10, diffusion:0.25)
2 Protein(shape:circle, size:4, diffusion:2);
```

Entities with a spatial extension, i. e., non-zero `size` attribute, must have a shape, where different keywords are accepted (`circle, disk, ball, sphere, square, cube, rectangle, cuboid` so far). All numeric values are default values that will be used when one is needed but not specified. One may specify different values for individual entities in rules or the initial state definition.

Key of each model are the reaction rules. Note that spatial entities are simulated individually in continuous space and can react only on collision with another entity. Therefore, second-order

[1] It can, in principle, also be used from inside JAMES II [HU07; Ewa+10]

[2] "Basically, it is a BSD license adapted for German law (regarding our liability and warranty), but which is also compatible with GPL (dual license)." – from jamesii.org/license.

[3] Programmers may think of species as classes, attributes as fields and entities as instances.

rules should not be associated with stochastic rates but with probabilities of them happening when sufficient proximity of the participating entities is already established. Also, third-order rules or higher could never be applied and should be broken down into steps of order ≤ 2.

Using the above species definitions, where rafts are regions on a membrane where (receptor) proteins diffuse more slowly than otherwise, an example of a rule is

```
Protein + Raft -> Raft()[Protein(diffusion*=0.5)] @ 1
```

where square brackets indicate nesting, i.e. that the enclosed entity (or entities) are/shall be inside the entity given before. `*=` is the change-by operator known from various programming languages[4] and `@ 1` indicates that this reaction shall happen always (i. e., with probability 1) when a protein and raft collide.

Finally, the initial state of the model must be defined. (It may also be placed between species definition and rules.) The dimensions of the system to be simulated are implicitly defined by the size of the top-level entity, of which there will be just one:

```
1 Membrane(shape:square,size:40*40,position:(0,0))[
  200 Protein + 10 Raft
  ];
```

This species (`Membrane`) has to be added to the species definition. The length of the first vector given, here the position attribute value, determines whether the model is in 2D or 3D space. Only the top level entity needs to have an explicit, absolute position, all nested entities without one are simply randomly placed inside the given surrounding.

For convenience, the mentioned three sections may be preceded by constants definitions. There, constant names are assigned numeric values. The constant names can later be used everywhere numbers are required, e. g., for size attributes, entity amounts in the initial state or reaction rate constants.

The file `LipidRafts.mls` included in the ML-Space sandbox package shows a simple but complete model from which the above examples were taken.[5]

2.2 Advanced Constructs: Rule variables, Bindings and more

Complex species definition example including

- `size` attribute specification involving a numerical expression with a previously defined constant (`mitoRadius`; definition not shown)and the `PI` keyword (for the value π),
- `diffusion` attribute specification using a range, from which actual attribute values will be taken randomly if needed (uniform distibution).
- binding sites definition (in `<>`) with angles (which should be given in radians or suffixed with ° for degree-radians conversion when the model file is read; the angles give positions of the binding sites relative to each other):

```
Actin(shape:circle,size:PI*actinRadius^2,diffusion:[0...actinDiff])<pointed:0,barbed:180°>
```

Rule example making use of bindings, where `free` and `occupied` (or, shorter, `occ`) are keywords for matching the state of a binding site, and `bind` is a keyword for the appropriate binding site change

[4] i. e., multiply current value by number given after the operator, and let the result be the new value of the attribute given before

[5] The model was featured in [Bit+11], which should not be taken as reference, however, as the ML-Space syntax was not properly worked out yet.

(as is `release` not used here):

```
Actin<pointed:free> + Actin<pointed:occ,barbed:free>
   -> Actin(diffusion:0)<pointed:bind>.Actin<barbed:bind> @ 1
```

Note that bindings were added to ML-Space for our simulation experiments with actin filaments in cells on surfaces [Bit+14a], where bound entities are immobile, and were used later for mitochondria [Bit+14b] with similar constraints. In terms of expressibility, this feature is not as advanced as rule-based languages focussing on bindings like BioNetGen [Bli+04; HHF09], which do not have to deal with spatial constraints. Regarding spatial simulation, bound entities (complexes) can only move jointly, i. e., all entities move in parallel without rotation, and angles other than 180° can only be used in 2D simulation so far. (Compare SRSim [Gru+10], which supports particle-based simulation in 3D with flexible binding angles between molecules with a rule-based format for reaction specification, but less flexible attributes and no nested entities.)

2.3 Syntax

Table 1: Sketch of the ML-Space syntax: `ident` stands for alphanumeric identifiers, subscripts are added to distinguish the roles of different `ident`s if a construct involves several. `numexpr` is a numerical expression that may involve previously defined constants, a limited set of functions and, in reaction rules, local variables. X? denotes 0 or 1 occurrences of X, X* 0 or more, X+ 1 or more, X | Y denotes "either X or Y".

1	Model	M	::=	$C^+\ S^+\ I^+\ R^+$
2	Constants	C	::=	ident '=' numexpr
3	Species definition	S	::=	$\text{ident}_{species}$ '(' ($\text{ident}_{attribute}$ ':' range)* ')' '<'($\text{ident}_{bindingsite}$ ':' numexpr)* '>'
4	Initial state	I	::=	(numexpr EP[I*])+ ';'
5	Reaction rule	R	::=	RL -> RR @ numexpr
6	Rule left hand side	RL	::=	ES^+ | ES [ES*] ES?
7	Rule right hand side	RR	::=	EP* | EP [EP*] EP?
8	Entity (substrate)	ES	::=	$\text{ident}_{species}$ '(' (AM | AV))* ')' '<'($\text{ident}_{bindingsite}$ ':' ('FREE' | 'OCC' | ES))* '>'
9	Attribute match	AM	::=	$\text{ident}_{attribute}$ ((OpC numexpr) | ('in' range))
10	Att. with variable	AV	::=	'(' ($\text{ident}_{localvar}$ '=' $\text{ident}_{attribute}$ ')' (OpC numexpr)?
11	Entity (product)	EP	::=	$\text{ident}_{species}$ '(' ($\text{ident}_{attribute}$ OpA numexpr)* ')' '<'($\text{ident}_{bindingsite}$ ':' ('BIND' | 'RELEASE'))* '>'
12	Comparison operation	OpC	::=	'<' | '<=' | '==' | '>=' | '>'
13	Assignment operation	OpA	::=	'=' | '-=' | '+=' | '/=' | '*='

The basic patterns of ML-Space model specifications are shown in Table 1. The actual concrete syntax is more complex to allow, for example,

- separating commas between successive elements of the same type, for example in attribute-value pair lists (but not at their beginning or end),
- separating + signs between entities on the same side of a rule or on the same level in the initial state definition

- omission of adjacent opening and closing parentheses/brackets of same type (i. e., without enclosed content)
- terminating semicolons at the end of most constructs (optional except after the initial state)
- textual, i. e., qualitative attribute values.

3 Running a Simulation Experiment

An ML-Space model and a simulation setup are needed for running a simulation.

The simulation setup consists, among other things, of specification of the time the simulation should run, what should be observed (i.e. written, in some form, to a `csv` file, and/or to image files), and model parameters that should be changed (e. g., in a perturbation experiment). It can be specified via the command line or a separate file (`ini`, `cfg`, `conf` or `exp` are accepted extensions).

The provided `mlspace.bat` and `mlspace.sh` files expect the model file name as first argument and will look for a simulation settings file of the same name. When running `mlspace.bat` or `mlspace.sh` without arguments, a list of available simulation settings (to be supplied in a settings file or as command line arguments) will be displayed.[6]

3.1 A Simulation Setup File

An example setting file for a lipid raft model experiment setup is shown below (also included in `LipidRafts.ini`). The lines' meaning is described below.

```
1  [Overridden model variables]
2  ; Variable name = comma-separated list of values
3  protRadius=0,1
4
5  ; further section names do not actually matter
6  [Simulation parameters]
7  parallelthreads = 2
8  graphics = screen
9  simulationendtime = 400
10
11 [Observation]
12 ; Variable name = comma-separated list of values
13 observationtargets = Prot in Raft
14 snapshotinterval = 25
```

1. override the variable "protRadius" specified there with the values 0 and 1 (one simulation run for each, full factorial if several parameters are overridden)
2. run 2 simulations in parallel
3. display an animation of the simulation on screen
4. end simulation after 400 time units
5. observe the amount of "Prot" entities that are inside "Raft" entities
6. take snapshots of the observables (i.e. count them) every 25 simulation time units

Except for `[Overridden model variables]`, all section headings are optional and arbitrary, but the other settings must occur in a different section than the overridden model variables.

[6] Only some of the customization options of the actual ML-Space code or the James II framework in which it was developed are exposed via the command line / settings file interface (a GUI or SESSL bindings – http://sessl.org – for convenience may follow).

If a model shall be run with an experiment setup that has a different name, the setup file should be the second argument, e. g., `mlspace LipidRafts.mls RaftSizeVariationExp.ini`. Note that additional command line arguments, if present, will take precedence over the corresponding key-value pair in an experiment file.

Log and csv files (one for each run, one summary containing the last row of each run's file and the simulation run parameters) will be written to the current directory (this can be changed by adding a line `outputdirectory=..`).

3.2 Command-line Simulation Setup

All keys for the `ini`-file's key-value pairs can also be used as command-line arguments. The keys (there is a shorter form for each) shall be preceded by `-` and the value should follow after a space. The above example would correspond to `mlspace LipidRafts.mls -O protRadius=0,1 -par 2 -graphics screen -obs "Prot in Raft" -snapint 25 -simendtime 400`

3.3 Observation Targets

Observation targets are specifications of entities whose amounts or properties shall be recorded and put into each simulation run's resulting csv file.

The observation target key or command line switch may be followed by a semicolon-separated list of targets or target-value pairs (but no semicolon at the end of the list, please). It is, unfortunately, difficult to define these in a proper section of the settings file as the keys may themselves contain `=` and thus not be read correctly.

An observation target can be written like an entity to-be-matched on the left hand side of a rule. For example, consider a MAP kinase that can be phosporylated twice, represented by a numeric attribute `phos`. Then `MAPK(phos==0);MAPK(phos==1);MAPK(phos==2);MAPK` would result in a column for each phosphorylation state and one for the total amount (which should equal the sum of the preceding three).

Hierarchical nesting is to be expressed via the `IN` keyword. There may be comma-separated lists on both sides of it, resulting in all combinations being observed. For example, `Prot,Raft IN Raft,Cell` would result in four columns, for proteins in rafts, proteins in a cell, rafts in rafts and rafts in the cell. (The third one is not necessarily a reasonable observation target.)

When entities-to-match are specified as above, by default their amount is the observed property. One may specify other properties via `=property`, for example `MAPK=phos`. This would observe the state of the `phos` attribute of all `MAPK`s, resulting in a column full of entries giving minimum, median, mean, maximum, standard deviation and observation amounts as well as the "histogram counts" in map format, for example `[0.0, 0.0, 0.125, 2.0, 0.4389855730355308, 24]{0.0=22, 1.0=1, 2.0=1}` when there is one doubly and one singly phosphorylated `MAPK` among 24 in total.

When there is only one entity matching the observation target key, the observed property's value is saved directly. This is useful, for example, when an attribute of the system (i.e. outermost) entity is used like a "global" variable, e.g. to keep track of some molecule counts.

When several properties of one entity shall be observed, they should be separated by a comma. the `#` symbol shall be used to indicate "amounts". Thus, `MAPK=phos,#` would result in recording of a separate column with the total count and the phosphorylation state of the present `MAPK`s as above (which includes amounts already, though).

4 Current Limitations and Known Issues

Graphics

- When running a simulation from command line with screen output, the batch file may not be terminated correctly (i.e. will not return to the command line).
- Displaying the simulation on screen (option `-graphics screen`) results in much slower simulation.
- 3D simulation graphical output is rather crude (projection onto xy-,xz- and yz-plane without regard to depth).

Experiment Setup

Not all features that can be customised in JAMES II can be customised in the current version of the ML-Space tool, including

- the event queue (future event set) used (the default used here is a Calendar queue with optimised reverse lookup)
- the random number generator (the default used here is a 32 bit Mersenne Twister (MT19937)).

Observation

- Reaction counts are so far only available for reactions triggered in the continuous-space simulator, not those of subvolume entities.

Simulation Speed

ML-Space was designed to allow simulation of entities with continuous coordinates, at different organisational levels and possibly nested, combined with reaction-diffusion simulation of populations of entities in subvolumes. When using it for only one of these tasks, it will be much slower than tools specialising in only one purpose.

As it was implemented in what was considered by the author at the time the most straightforward manner and most suitable for various extensions[7], there may be some room for some not too difficult optimisations, but this is currently not the most pressing issue and ML-Space simulation speed should not be expected to be close to that of specialised particle-based or subvolume-based simulators anytime soon.

[7] One examples would arbitrary shapes for spatial entities – currently, only circles/spheres and axis-aligned boxes are supported, but new ones can be added without any changes in the simulator, only an interface providing methods for collision handling with other shapes needs to be implemented. Another example would be subvolumes of different sizes, e. g., to add adaptive coarse-graining and subdivision as in [JU08].

References

[Bit+11] Arne T. Bittig, Fiete Haack, Carsten Maus, and Adelinde M. Uhrmacher. "Adapting rule-based model descriptions for simulating in continuous and hybrid space". In: *Proceedings of the 9th International Conference on Computational Methods in Systems Biology*. CMSB '11. New York, NY, USA: ACM, 2011, pp. 161–170. ISBN: 978-1-4503-0817-5. DOI: 10.1145/2037509.2037533.

[Bit+14a] Arne T. Bittig, Claudia Matschegewski, J. Barbara Nebe, Susanne Stählke, and Adelinde M. Uhrmacher. "Membrane related dynamics and the formation of actin in cells growing on micro-topographies: a spatial computational model". In: *BMC Systems Biology* 8 (2014), pp. 106+. ISSN: 1752-0509. DOI: 10.1186/s12918-014-0106-2.

[Bit+14b] Arne T. Bittig, Florian Reinhardt, Simone Baltrusch, and Adelinde M. Uhrmacher. "Predictive Modelling of Mitochondrial Spatial Structure and Health". In: *Computational Methods in Systems Biology*. Ed. by Pedro Mendes, Joseph O. Dada, and Kieran Smallbone. Vol. 8859. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 252–255. DOI: 10.1007/978-3-319-12982-2_20.

[Bli+04] Michael L. Blinov, James R. Faeder, Byron Goldstein, and William S. Hlavacek. "BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains." In: *Bioinformatics (Oxford, England)* 20.17 (Nov. 2004), pp. 3289–3291. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bth378.

[Ewa+10] Roland Ewald, Jan Himmelspach, Matthias Jeschke, Stefan Leye, and Adelinde M. Uhrmacher. "Flexible experimentation in the modeling and simulation framework JAMES II–implications for computational systems biology". In: *Brief Bioinform* 11.3 (Jan. 2010), pp. 290–300. ISSN: 1477-4054. DOI: 10.1093/bib/bbp067.

[Gru+10] Gerd Gruenert, Bashar Ibrahim, Thorsten Lenser, Maiko Lohel, Thomas Hinze, and Peter Dittrich. "Rule-based spatial modeling with diffusing, geometrically constrained molecules". In: *BMC Bioinformatics* 11.1 (2010), pp. 307+. ISSN: 1471-2105. DOI: 10.1186/1471-2105-11-307.

[HHF09] Leonard A. Harris, Justin S. Hogg, and James R. Faeder. "Compartmental rule-based modeling of biochemical systems". In: *Proceedings of the 2009 Winter Simulation Conference (WSC)*. Ed. by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls. Austin, TX, USA: IEEE Computer Science, Dec. 2009, pp. 908–919. ISBN: 978-1-4244-5770-0. DOI: 10.1109/wsc.2009.5429719.

[HU07] Jan Himmelspach and Adelinde M. Uhrmacher. "Plug'N Simulate". In: *Proceedings of the 40th Annual Simulation Symposium*. ANSS '07. Washington, DC, USA: IEEE Computer Society, Mar. 2007, pp. 137–143. ISBN: 0-7695-2814-7. DOI: 10.1109/anss.2007.34.

[JU08] Matthias Jeschke and Adelinde M. Uhrmacher. "Multi-resolution spatial simulation for molecular crowding". In: *Proceedings of the 2008 Winter Simulation Conference*. Ed. by S. J. Mason, R. R. Hill, L. Moench, and O. Rose. Miami, FL, USA: IEEE, Dec. 2008, pp. 1384–1392. ISBN: 978-1-4244-2707-9. DOI: 10.1109/WSC.2008.4736214.