Spatial Kappa Simulator
User Guide
v1.0.0

Donal Stewart
DemonSoft.org

March 4, 2011

Contents

Chapter 1

Spatial Kappa simulator User Guide

1.1 Obtaining the simulator

The simulator is available from GitHub as the source Eclipse project, or as a single executable jar file. Both are available at `https://github.com/donal-s/SpatialKappa/downloads`. The simulator in its current state was optimised specifically for the 2010 Edinburgh iGEM project. This is apparent mostly in the design of the compartment view pane. Extending this view for more generic use is a planned extension.

1.2 Starting the simulator

1.2.1 Running the executable jar

The simulator can be started by running the executable jar file:
`java -jar SpatialKappa-v1.0.0.jar`

Double clicking the jar file usually works too.

1.2.2 Running from the Eclipse project

The main class of the simulator is
`org.demonsoft.spatialkappa.ui.TransitionMatchingSimulatorGui`

Running as a Java Application will bring up the simulator.

1.3 Using the simulator

The initial screen appears as figure 1.1.

The toolbar options are:

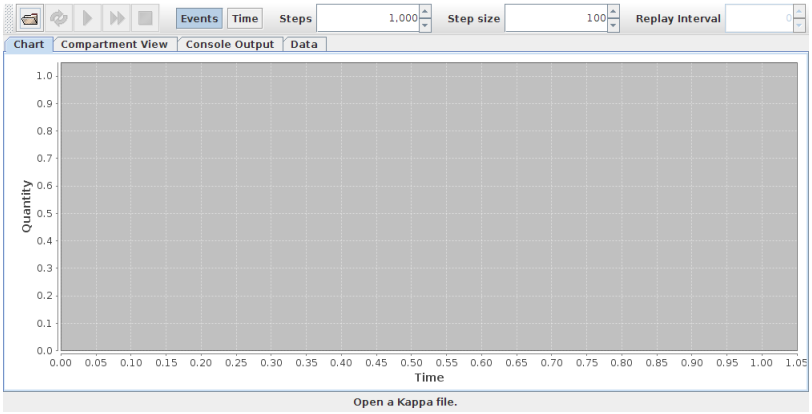
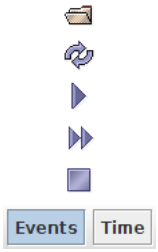


Figure 1.1: Initial view



Open a Kappa or Kappa replay file

Reopen the current file

Run the current file

Replay the current replay file or last run

Stop the currently running simulation

Choose event or time based simulation

1.3.1 Opening a Kappa or Kappa replay file

Select the 'Open' button on the toolbar and select the file to open. The current implementation expects Kappa source files to have the suffix `.ka` and Kappa replay files (discussed later) to have the suffix `.kareplay`. If the file is parsed successfully, a summary of the Kappa model is displayed in the 'Data' pane (see figure 1.2).

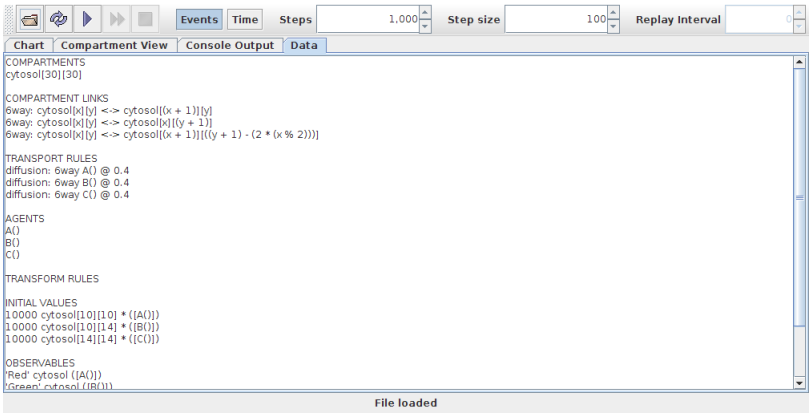


Figure 1.2: Data pane showing loaded Kappa model

Any errors in reading the Kappa file are shown in the 'Console Output' pane.

The currently open Kappa file can be refreshed from disk by selecting the 'Reopen' button. Useful when editing the Kappa model.

1.3.2 Running a simulation

With a successfully opened Kappa model, one can run a simulation by selecting the 'Run' button. Simulation parameters can be set on the toolbar before running. There is the option to do an event or a time based simulation. For an event based simulation, the number of steps for the simulation (i.e. data points on the time series chart), and the number of finite rate events per step can be set. Equivalent options for time based simulation can also be set.

The simulation can be halted at any point by selecting the 'Stop' button. Note that complex simulations may take a while to start up while data structures are being generated.

1.3.3 Running a simulation replay

As the simulation runs, the state of the simulation observables are logged to disk in a replay file after every step. Once the simulation is complete, this replay file can be rerun by selecting the 'Replay' button. The 'Replay Interval' field allows a delay (in ms) to be added between each step.

Note - the current storage format is binary, and version dependent. Creation of a more permanent trace storage format is a planned enhancement.

1.4 Spatial visualisation tools

While the raw data produced from simulations is useful, visualisation of the data is important. There are a couple of simple visualisation panels in the simulator. These are dynamically updated as the simulation runs to give the user an idea of how the simulation is progressing. They are however basic in comparison to some of the commercial simulation data visualisation tools available.

The excellent JFreeChart (Gilbert et al., 2010) library was used for generating the charting components. Both charts have formatting and save capability, and the time series panel is zoomable.

1.4.1 Time series chart

This chart is similar to the standard Gnuplot output from Simplx. It is a line graph showing observable quantity against time for all observable definitions in the model.

1.4.2 Compartment chart

This view allows more detailed visualisation of the transport of a species within the cells of a single compartment. It can show local positive feedback sites, or diffusion of a species through a compartment. There are a couple of visualisation options which can be selected at runtime to tailor the output.

Note that the current implementation is designed to view a 3-channel hexagonal mesh, and that view works only for observations labelled `'Red'`, `'Blue'`

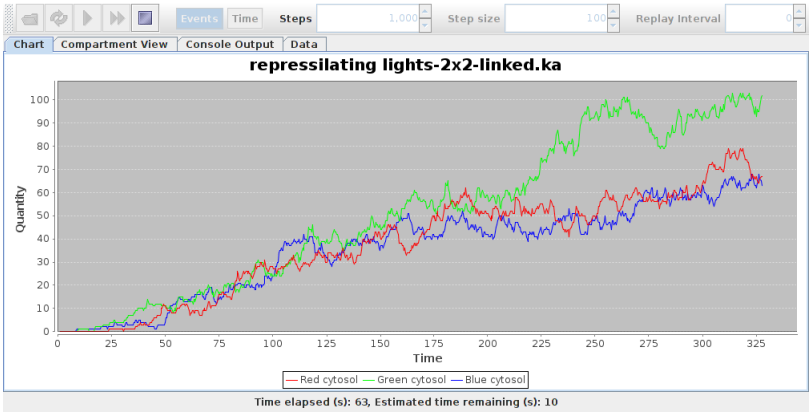


Figure 1.3: Sample time series chart output

and `'Green'` in a compartment labelled `'Cytosol'`. Making this more generic is also a planned area for extension.

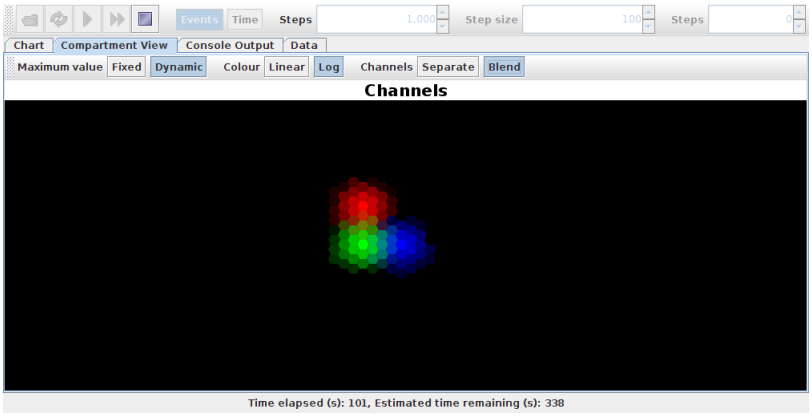


Figure 1.4: Sample compartment chart output

Chapter 2

Kappa Language Extensions

Note on terminology. In the following 'cell' can mean either a living cell, or a cell within a defined compartment. Hopefully the meaning in each situation will be clear from the context. Current Kappa rules are referred to as 'transform' rules as they cause some form of transformation of their substrates, be it change of agent state, agent creation or deletion, agent binding or unbinding. Species is mostly used to refer to a particular agent or complex in the model.

Also this document uses the terms 'language' and 'calculus' interchangeably, for example κ-calculus and Kappa language. Generally the term calculus is used when referring to the formal definition, and language where referring to models defined in Kappa, as they are represented in the Kappa language syntax.

2.1 Existing Kappa language

The current Kappa language consists of the following constructs, as described in the Simplx documentation (Krivine, 2009). A more formal description of the language grammar (including the spatial extensions) is given in appendix A.

Comments:

```
# This is a comment
```

Agents:

```
AgentName()
AgentName(stateWithValue~stateValue, stateWithNoValue, unboundSite, boundSite!1)
```

Transform rules:

```
'state change' A(state~old) -> A(state~new) @ 0.1
'binding'      A(bindsite),B(bindsite) -> A(bindsite!1),B(bindsite!1) @ 0.1
'unbinding'    A(bindsite!1),B(bindsite!1) -> A(bindsite),B(bindsite) @ 0.1
'creation'     -> A() @ 0.1
'degradation'  A() -> @ 0.1
'reversible reaction'  A(state~old) <-> A(state~new) @ 0.1,0.2
A(state~old) -> A(state~new) @ 0.1 # Unnamed transform rule
```

Initial species values:

```
%init 1000 * (A(state~old),C())            # 1000 of each of A(..) and C()
%init 2000 * (A(bindsite!1),B(bindsite!1)) # 2000 of the bound complex A(..),B(..)
```

Observables and named variables: Variables can be referenced in perturbation calculations, but do not show in outputs as observations.

```
%obs 'Label' A()  # All agents A()
%obs A()          # Unnamed observation, will default to 'A()' in outputs
%obs A(state~old) # All agents matching A(state~old)
%obs 'binding'    # Activity of the transform rule named 'binding'

%var 'Named variable' B()
```

Perturbations: These only fire a single time, after which they are disregarded for the rest of a simulation.

```
# When simulation time exceeds 5 seconds, change the rate of transform rule
#   named 'unbinding' to 10.0
%mod: $T > 5 do unbinding:= 10.0

# When the species value mapped to observable 'mRNA' drops below 50
#   set the rate of transform named 'transcribe' to 0.5
%mod: 'mRNA' < 50 do 'transcribe' := 0.5
```

Currently unused: There are some Kappa syntax elements not directly relevant to the spatial extensions, and will not appear in the extended grammar. One is a representation of causal flows, which uses the keyword `%causal`.

```
%causal: C@s2
%causal: A..B,C@s1 => C@s2
```

2.2 Concepts to encapsulate

The spatial Kappa language requires some encoding of the following concepts to be useful.

- A description of **compartments** and their **dimensions**. A model may have multiple compartments each containing reacting species. The dimensions are necessary to define the shape of a compartment, be it a single cell, a 1 dimensional linear array of cells, a 2 dimensional grid of some form, or a 3 dimensional lattice structure. Relative differences in size of different compartments can be specified, e.g. the reacting volume of a nucleus relative to the surrounding cytosol. Relative differences in shape can also be specified, for example the thin layer of cytosol next to the inner surface of the plasma membrane versus the rest of the cytosol.
- A description of **linkage**, both intra-compartment and inter-compartment. The intra-compartment linkage specification should be rich enough to allow description of multiple structures, e.g. in 1D linear arrays or circles, in 2D square or hexagonal meshes, cylinders or tori, in 3D cubes, filled cylinders, spheres, etc.
- A means of **locating species** within compartments, e.g. all DNA would reside within the nucleus, or cell receptors would be limited to the plasma membrane.

- A means of **locating transform rules** within compartments, e.g. DNA transcription is isolated to the nucleus. The language should also allow the same transform rule to be specified with different rates depending on the location of the reacting species.
- A description of the **transport** (active or diffusive) of species within a compartment or between compartments along previously described linkage structures. The rates of transport should be general to all species, or species specific.

Additional concepts

- **Granularity** within compartments. It would be useful to be able to specify locations at the level of compartments or single cells within a compartment. This would allow the model to represent, for example, a signal cascade being initiated as one point in the cytosol, and the resulting signal molecules being diffused through the cytosol.
- **Backwards compatibility** with basic Kappa. Given the quantity of existing models, the extended language should allow the existing models to work as before without modification. The user should have the choice of not using the spatial aspects of the extended language with no rework penalty.

2.3 New language constructs

A full BNF description of the extended Kappa grammar is given in appendix A.

2.3.1 Compartments and cells

Compartments are defined as single cells or regular multidimensional arrays of cells as follows

```
'%compartment:' LABEL ('[' INT ']')*
```

For example

```
%compartment: 'Single cell'
%compartment: '1d array' [10]       # 10 cells in size
%compartment: '2d array' [10][5]    # 10x5 cells in size
%compartment: '3d array' [10][5][4] # 10x5x4 cells in size
```

Compartments or individual cells within a compartment can be referenced using the following syntax

```
LABEL ( '[' mathExpr ']' )*
```

where

```
mathExpr :
  mathAtom OPERATOR mathAtom
  | mathAtom

mathAtom :
  '(' mathExpr ')'
```

```
  | INT
  | VARIABLE_NAME

OPERATOR :
  '+' | '-' | '*' | '/' | '%'
```

For example

```
'my compartment'                      # the compartment as a whole
'my compartment' [0][0][0]            # the first cell in a 3d array compartment
'my compartment' [4]                  # the fifth cell in a 1d linear arrays

'my compartment' [x][y][z]            # variable name usage described in
'my compartment' [x*2][y-1][z+(x*3)] #   compartment links section below
```

In all situations where compartment references are used, it is only legal to refer to the compartment as a whole by omitting the cell indices, or refer to a single cell, by fully defining the correct number of cell indices to match the dimensions of the compartment. Also, variable names are only permitted in compartment references within compartment link definitions, described below.

2.3.2 Compartment links

The structure of a compartment is further defined by how cells within the compartment are linked to each other and to connected compartments. This linkage is defined as follows

```
'%link:' LABEL compartmentReferenceExpr ('<->' | '<-' | '->') compartmentReferenceExpr
```

Where `compartmentReferenceExpr` is as described above. For example

```
%compartment: '2d array' [10][200]    # 10x200 cells in size

# Link all cells to their horizontally adjacent neighbours
%link: 'meshlinks' '2d array'[x][y] <-> '2d array'[x+1][y]

# Link all cells to their vertically adjacent neighbours
%link: 'meshlinks' '2d array'[x][y] <-> '2d array'[x][y+1]

# Wrap around the cells on the left and right edges to create a cylinder
%link: 'meshlinks' '2d array'[0][y] <-> '2d array'[9][y]

# Wrap around the cells on the top and bottom edges to create a torus
%link: 'meshlinks' '2d array'[x][0] <-> '2d array'[x][199]
```

The above code defines a thin torus composed of a 2d mesh.

Compartment references on the left hand side of the link definitions above may contain either constant values or single variable names, not complex expressions. The variable names are used to define the dimensions which will be iterated through to produce links. Compartment references on the right hand side allow constant values or complex expressions involving the variables defined on the left had side of the expression. It is invalid for the right hand expression to use variables not defined on the left. If setting the values of variables references valid cells on both the left and right, then those cells are deemed to be linked. References which refer to cells outside the dimensions of the compartment are ignored, and no link is created. The references in a link expression can refer to the same compartment, or two different compartments. The modulus

operator `%` is useful in defining regular, repeating linkage patterns within the compartment, for example the 2D hexagonal mesh described in appendix B.1.

Multiple link definitions can use the same label (like `meshlinks` in the example above), in which case references to that label elsewhere in the model acts on the union of all the link definitions. This allows complex linkage definitions, for example compartments constructed in the form of a circle or sphere, to be referenced concisely in transport rules.

Further examples of compartment and link specifications for common structures are given in appendix B.1.

2.3.3 Transport rules

These rules define the rate at which species move from one compartment (or cell) to another.

```
'%transport:' (transportName=LABEL)? linkName=LABEL (agentGroup)? transportKineticExpr
```

where

```
agentGroup :
  agent (',' agent)*

transportKineticExpr :
  '@' a=rateValueExpr

rateValueExpr :
  number | '$INF'   # $INF means infinite rate
```

See appendix A for the definition of the remaining constructs. For examples of usage

```
# All species diffusing around the torus defined above
%transport: 'general diffusion' 'meshlinks' @ 0.1

# Immediate exit of the named species through a uni-directional channel.
# Note the label for the transport is optional.
%transport: 'Ca channel' Calcium() @ $INF
```

When referencing bidirectional compartment links (i.e. ones specified with '`<->`', the rate applies equally in either direction across the link. In the second example above, care would need to be taken in defining the compartment link `Ca channel` in the correct orientation.

2.4 Additions to existing language constructs

New clauses were added to existing Kappa language constructs to allow the use of compartments and links. In all cases, existing Kappa syntax is still valid. Where necessary, the optional label in existing Kappa statements is non-optional in statements using compartments to avoid ambiguity. When basic Kappa syntax is used in the context of a spatially aware model, the statement is generally taken to apply to all compartments in the model equally.

2.4.1 Transform rules

The following transform statement was added

```
transformName=LABEL locationExpr transformExpr transformKineticExpr
```

where

```
transformExpr :
  (a=agentGroup)? ( '->' | '<->' ) (b=agentGroup)?

transformKineticExpr :
  '@' a=rateValueExpr (',' b=rateValueExpr)?
```

For example

```
# mRNA degradation happens outside the nuclear membrane
'degradation' 'cytosol' mRNA() -> @ 6.21

# Heating of the agent occurs in cell 0 of the cytosol compartment
'heating' 'cytosol'[0] A(state~blue) -> A(state~red) @ 1.0
```

As mentioned before valid locations are either entire compartments or single compartment cells.

2.4.2 Initial values

The following initial value statement was added

```
initExpr :
  '%init:' locationExpr INT '*' '(' agentGroup ')'
```

For example

```
# Distribute 2000 blue agents within the cells of cytosol equally
%init: 'cytosol' 2000 * (A(state~blue))

# Add 500 red agents to cell 5 of the cytosol compartment
%init: 'cytosol'[5] 500 * (A(state~red))
```

2.4.3 Observations

The following observation statement was added

```
obsExpr :
  '%obs:' LABEL locationExpr agentGroup
```

For example

```
# Count all blue agents in all cells of the cytosol
%obs: 'cytosol blue' 'cytosol' A(state~blue)

# Count all red agents in cell 0 of the cytosol compartment
%obs: 'red[0]' 'cytosol'[0] A(state~red)
```

For example models demonstrating the use of the language extensions, refer to appendix B.

Appendix A

Spatial Kappa Grammar

The following is a cut down version of the Antlr grammar used in the Kappa simulator. The syntax has been trimmed for readability, as the original Antlr grammar has artificial constructs for dealing with left recursion, etc. It is read basically as BNF notation with assignments (`variable=bnfConstruct`). The existing basic Kappa grammar is shown in `black`, with the spatial constructs shown as `blue`.

```
prog :
  (line)+

line :
  ruleExpr NEWLINE!
  | compartmentExpr NEWLINE!
  | compartmentLinkExpr NEWLINE!
  | transportExpr NEWLINE!
  | initExpr NEWLINE!
  | obsExpr NEWLINE!
  | varExpr NEWLINE!
  | modExpr NEWLINE!
  | COMMENT!
  | NEWLINE!

ruleExpr :
  LABEL? transformExpr transformKineticExpr
  | LABEL locationExpr transformExpr transformKineticExpr

transformExpr :
  (a=agentGroup)? transformTransition (b=agentGroup)?

transportExpr :
  '%transport:' (transportName=LABEL)? linkName=LABEL (agentGroup)?
    transportKineticExpr

agentGroup :
  agent (',' agent)*

agent :
  id '(' (iface (',' iface)*)? ')'

iface :
  id stateExpr? linkExpr?
```

```
stateExpr :
  '~' marker

linkExpr :
  ( '!' INT | '!' '_'  | '?' )

transformKineticExpr :
  '@' a=rateValueExpr (',' b=rateValueExpr)?

transportKineticExpr :
  '@' rateValueExpr

rateValueExpr :
  a=number
  | b='$INF'

initExpr :
  '%init:' INT '*' '(' agentGroup ')'
  | '%init:' locationExpr INT '*' '(' agentGroup ')'

compartmentExpr :
  '%compartment:' LABEL ('[' INT ']')*

compartmentLinkExpr :
  '%link:' linkName=LABEL sourceCompartment=locationExpr transportTransition
     targetCompartment=locationExpr

locationExpr :
  sourceCompartment=LABEL compartmentIndexExpr*

compartmentIndexExpr :
  '[' mathExpr ']'

obsExpr :
  '%obs:' LABEL? agentGroup
  | '%obs:' LABEL locationExpr agentGroup
  | '%obs:' LABEL

varExpr :
  '%var:' LABEL agentGroup

modExpr :
  '%mod:' concentrationInequality 'do' assignment
  | '%mod:' timeInequality 'do' assignment

timeInequality :
  '$T' '>' number

concentrationInequality :
  concentrationExpression '>' concentrationExpression
  | concentrationExpression '<' concentrationExpression


assignment :
  LABEL ':=' '$INF'
  | LABEL ':=' a=concentrationExpression

concentrationExpression :
```

```
  '(' concentrationExpression ')'
  | LABEL operator concentrationExpression
  | number operator concentrationExpression
  | LABEL
  | number

mathExpr :
  a=mathAtom operator b=mathAtom
  | a=mathAtom

mathAtom :
  '(' mathExpr ')'
  | INT
  | VARIABLE_NAME

id :
  ALPHANUMERIC ( ALPHANUMERIC | '_' | '^' | '-' )*

marker :
  ALPHANUMERIC

number :
  ( INT | FLOAT )

operator :
  ( '+' | '*' | '-' | '/' | '%' )

transformTransition :
  ( '->' | '<->' )

transportTransition :
  ( '->' | '<-' | '<->' )

INT :
  NUMERIC

FLOAT :
    NUMERIC '.' NUMERIC EXPONENT?
    | '.' NUMERIC EXPONENT?
    | NUMERIC EXPONENT

VARIABLE_NAME :
  ('a'..'z' | 'A'..'Z') ('a'..'z' | 'A'..'Z' | '0'..'9')*

ALPHANUMERIC :
  ( NUMERIC | 'a'..'z' | 'A'..'Z' )+

NUMERIC :
  ('0'..'9')+

EXPONENT :
  ('e'|'E') ('+'|'-')? NUMERIC

LABEL :
  '\'' .* '\''

COMMENT :
  '#' ~( '\n' | '\r' )* NEWLINE
```

```
NEWLINE :
  '\r'? '\n' | '\r'

WS :
  ( ' ' | '\t' )+
```

Appendix B

Spatial Kappa Examples

B.1 Spatial Kappa patterns

The following are generic shapes, with their equivalent Spatial Kappa representations. These are intended to be copied during model development.

B.1.1 1 dimensional patterns

Linear array

```
%compartment: 'array' [n] # Replace n with length of array
%link: 'intra-array' 'array' [x] <-> 'array' [x+1]
```

Circle

```
%compartment: 'circle' [n] # Replace n with number of cells in circle
%link: 'intra-circle' 'circle' [x] <-> 'circle' [x+1]
%link: 'intra-circle' 'circle' [n-1] <-> 'circle' [0] # Replace n-1 as above
```

B.1.2 2 dimensional surfaces

Rectangular mesh

There are 2 variants here, 4-way linked and 8-way linked.

```
%compartment: 'mesh' [n][m] # Replace n and m with dimensions of mesh

# 4-way diffusion
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x][y+1]

# or 8-way diffusion
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x][y+1]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y+1]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y-1]
```

Hexagonal mesh

Again, 2 variants depending on what overall shape is required. The first form has a simpler representation of intra-compartment links, but the overall structure

is rhomboid, whereas the second produces an overall rectangular shape at the expense of more complicated link statements.

The second variant demonstrates handling of alternate odd-even linkage depending on the column of the structure.

```
%compartment: 'mesh' [n][m] # Replace n and m with dimensions of mesh

# Variant 1 - rhomboid mesh
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x][y+1]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y+1]

# Variant 2 - rectangular mesh
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x][y+1]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][(y+1)-(2*(x%2))]
# The above statement alternates [x+1][y+1] and [x+1][y-1] as x increases
```

Cylinder and torus

By connecting together the top and bottom edges of a mesh as described above, we get a cylinder. By also connecting together the left and right edges we get a torus.

```
%compartment: 'mesh' [n][m] # Replace n and m with dimensions of mesh

# 4-way diffusion
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x+1][y]
%link: 'intra-mesh' 'mesh' [x][y] <-> 'mesh' [x][y+1]

# add this to get a cylinder
%link: 'intra-mesh' 'mesh' [x][m-1] <-> 'mesh' [x+1][0] # Replace m-1 as above

# add this to get a torus
%link: 'intra-mesh' 'mesh' [n-1][y] <-> 'mesh' [0][y] # Replace n-1 as above
```

B.2 Sample Spatial Kappa models

The following are a couple of simple spatial kappa models to demonstrate the use of the language features.

B.2.1 2d diffusion model

This model shows simple diffusion from a point for three distinct species. It was developed as a test for the Compartment view pane. Note again that the current implementation of the Compartment view works only for observations labelled `'Red'`, `'Blue'` and `'Green'` in a compartment labelled `'Cytosol'`.

```
%compartment: 'cytosol' [30][30]

# 6-way diffusion
%link: '6way' 'cytosol' [x][y] <-> 'cytosol' [x+1][y]
%link: '6way' 'cytosol' [x][y] <-> 'cytosol' [x][y+1]
%link: '6way' 'cytosol' [x][y] <-> 'cytosol' [x+1][(y+1)-(2*(x%2))]

%transport: 'diffusion' '6way' A() @ 0.4
%transport: 'diffusion' '6way' B() @ 0.4
%transport: 'diffusion' '6way' C() @ 0.4

%init: 'cytosol'[10][10] 10000 * (A())
%init: 'cytosol'[10][14] 10000 * (B())
%init: 'cytosol'[14][14] 10000 * (C())

%obs: 'Red' 'cytosol' A()
%obs: 'Green' 'cytosol' B()
%obs: 'Blue' 'cytosol' C()
```

B.2.2 Bi-trivalent binding model

A variant of bi-trivalent binding test model (Yang et al., 2008). This spatial model allows unbound species to diffuse through the compartment, but bound species remain within a cell of the compartment. When run in the compartment view, this demonstrates the aggregation of the species involved.

```
%compartment: 'cytosol' [20][20]

# 6-way diffusion
%link: '6way' 'cytosol' [x][y] <-> 'cytosol' [x+1][y]
%link: '6way' 'cytosol' [x][y] <-> 'cytosol' [x][y+1]
%link: '6way' 'cytosol' [x][y] <-> 'cytosol' [x+1][(y+1)-(2*(x%2))]

%transport: 'diffusion' '6way' A(bindings~0) @ 0.1
%transport: 'diffusion' '6way' B(a,b,c) @ 1

A(a,b,bindings~0), B(a) <-> A(a!1,b,bindings~1),B(a!1) @ 1,0.01
A(a,b,bindings~0), B(b) <-> A(a!1,b,bindings~1),B(b!1) @ 1,0.01
A(a,b,bindings~0), B(c) <-> A(a!1,b,bindings~1),B(c!1) @ 1,0.01
A(a,b,bindings~0), B(a) <-> A(a,b!1,bindings~1),B(a!1) @ 1,0.01
A(a,b,bindings~0), B(b) <-> A(a,b!1,bindings~1),B(b!1) @ 1,0.01
A(a,b,bindings~0), B(c) <-> A(a,b!1,bindings~1),B(c!1) @ 1,0.01
A(a,b!_,bindings~1), B(a) <-> A(a!1,b!_,bindings~2),B(a!1) @ 1,0.01
A(a,b!_,bindings~1), B(b) <-> A(a!1,b!_,bindings~2),B(b!1) @ 1,0.01
A(a,b!_,bindings~1), B(c) <-> A(a!1,b!_,bindings~2),B(c!1) @ 1,0.01
A(a!_,b,bindings~1), B(a) <-> A(a!_,b!1,bindings~2),B(a!1) @ 1,0.01
A(a!_,b,bindings~1), B(b) <-> A(a!_,b!1,bindings~2),B(b!1) @ 1,0.01
```

```
A(a!_,b,bindings~1), B(c) <-> A(a!_,b!1,bindings~2),B(c!1) @ 1,0.01

%init: 'cytosol'       600 * (A(a,b,bindings~0))
%init: 'cytosol'       400 * (B(a,b,c))

%obs: 'Red' 'cytosol' A(bindings~2)
%obs: 'Green' 'cytosol' A(bindings~1)
%obs: 'Blue' 'cytosol' A(bindings~0)
```

Bibliography

Gilbert, D. et al. (2010). Jfreechart website http://www.jfree.org/jfreechart/.

Krivine, J. (2009). *Simplx 4 release documentation.*

Yang, J., Monine, M., Faeder, J., and Hlavacek, W. (2008). Kinetic Monte Carlo method for rule-based modeling of biochemical networks. *Physical Review E*, 78(3):31910.