Spatial Modelling based on the BMK-Framework

Mary-Ann Blätke¹, Christian Rohr²

¹ Otto-von-Guericke-University Magdeburg, Germany
² Brandenburg University of Technology Cottbus, Germany
Background

- Systems Biology encouraged the modelling of bio-systems
  - detailed mechanistic & kinetic descriptions
  - ignore spatial aspects

- But space matters:
  - Integration of spatial aspects in existing models is not straightforward
    - models have to be rebuild
Our Idea

• Extend models with spatial information to keep track of position and movement

• Biomolecule-centred modularisation of models

→ Biomodelkit Framework

A module describes the functionality and interaction of a biomolecule in the form of a Petri net
BMK-Framework

Biomolecular Systems

- Molecular Mechanisms
- Experimental Data
- Boolean Networks
- SBML Models
- Forward Engineering
- Reverse Engineering
- Transformation

Gene Modules
mRNA Modules
Protein Modules
Protein Degradation Modules
Causal Influence Modules
Allelic Influence Modules

Biomodelkit Database

Model Composition

- Add Modules to Collection
- Set of Modules

Model-based Predictions

- Wildtype/Alternative Models
- Spatial Models

High-Throughput Analysis

- + Algorithmic Mutation
- + Space-Attributes
Running Example
Running Example

Protein A

Protein B

Protein C

LBD

LBD

LBD

LBD

LBD

LBD

LBD

YP

YP

YP

CD

CD

CD

SH2

SH2

SH2

Protein A

Protein B

Protein C

Protein A

Protein B

Protein C

Protein B

Protein C
Modules of the Running Example

Module of Protein A

Module of Protein B
Step 1: Explicit Encoding of Local Positions

- Define a grid for each component (1D, 2D or 3D)
  - e.g. 2D Grid: xDimA = 10, yDimA = 10

- Set a local position for each component
  - e.g. A(3,8), B(4,5), C(9,3)
Step 1: Explicit Encoding of Local Positions

- Define a grid for each component (1D, 2D or 3D)
  - e.g. 2D Grid: xDimA = 10, yDimA = 10
- Set a local position for each component
  - e.g. A(3,8), B(4,5), C(9,3)
Step 1: Explicit Encoding of Local Positions

- Define a grid for each component (1D, 2D or 3D)
  - e.g. 2D Grid: $x_{\text{DimA}} = 10$, $y_{\text{DimA}} = 10$
- Set a local position for each component
  - e.g. A(3,8), B(4,5), C(9,3)

Add a place for each component and axis
Step 1: Explicit Encoding of Local Positions

Module of Protein A

Module of Protein B

XY-Position of Protein A

XY-Position of Protein B

XY-Position of Protein C

ProteinA_X  ProteinA_Y

ProteinB_X  ProteinB_Y

ProteinC_X  ProteinC_Y
Step 2: Local Restriction of Interactions

- Components can only interact if they fulfil a defined neighbourhood relation.
- Define a neighbourhood relation.
  - e.g. local positions of components must be identical.
Step 2: Local Restriction of Interactions

• Components can only interact if they fulfil a defined neighbourhood relation

• Define a neighbourhood relation
  • e.g. local positions of components must be identical
Step 2: Local Restriction of Interactions

• Components can only interact if they fulfil a defined neighbourhood relation

• Define a neighbourhood relation
  
  • e.g. local positions of components must be identical

For each transition representing an interaction:

- add coordinate places of interacting components

- multiply firing-rate with a boolean expression evaluating the distance between the interacting components
Step 2: Local Restriction of Interactions

Module of Protein A

- ProteinA_LBD
- ProteinB_LBD
- AB_t1
- AB_t2
- ProteinA_LBD _ ProteinB_LBD
- ProteinA_CD_active
- A_t2
- A_t1
- ProteinA_CD_inactive

Module of Protein B

- ProteinA_LBD
- ProteinB_LBD
- AB_t1
- AB_t2
- BC_t1
- ProteinB_TYRp
- ProteinB_TYRp _ ProteinC_SH2
- ProteinC_SH2
- ProteinB_TYRp
- ProteinA_CD_active
- ProteinA_CD_active
- BC_t2
- AB_t3

Restricted Interaction - Only if ProteinA (X, Y) = ProteinB (X, Y)

(ProteinA_X = ProteinB_X) & (ProteinA_Y = ProteinB_Y)*h(t)
Step 2: Local Restriction of Interactions

Restricted Interaction - Only if ProteinB (X,Y) = ProteinC (X,Y)

(ProteinB_X = ProteinC_X) & (ProteinB_Y = ProteinC_Y)*h(t)
Step 3: Explicit Encoding of Local Position Changes

Case 1: Movement of components as a single entity

Case 2: Movement of components as complex
Case 1: Movement of components as a single entity

- Movement along the axes in respect to the defined grid size
  - e.g. 2D-Grid -> 2 direction of movement per axis
- Movement only if all interaction sites are unused
  - e.g. Protein B is not allowed to interact with Protein A or Protein C
Case 1: Movement of components as a single entity

- Movement along the axes in respect to the defined grid size
  - e.g. 2D-Grid -> 2 direction of movement per axis
- Movement only if all interaction sites are unused
  - e.g. Protein B is not allowed to interact with Protein A or Protein C
Case 1: Movement of components as a single entity

• Movement along the axes in respect to the defined grid size
  • e.g. 2D-Grid -> 2 direction of movement per axis

• Movement only if all interaction sites are unused
  • e.g. Protein B is not allowed to interact with Protein A or Protein C

For each component:
- add two transitions per axis to increase/decrease the value of the coordinate places in respect to the grid size
- connect places representing interaction states of the components via inhibitory arcs
**Case 1: Movement of components as a single entity**

### Movement of Protein A

- **ProteinA_X**
  - $XL_A$
  - $XR_A$
  - 2
  - $xDimA$

- **ProteinA_LBD** and **ProteinB_LBD**

### Movement of Protein B

- **ProteinB_X**
  - $XL_B$
  - $XR_B$
  - 2
  - $xDimB$

- **ProteinA_LBD** and **ProteinB_LBD**

### Movement of Protein C

- **ProteinC_X**
  - $XL_A$
  - 4
  - $xDimC$

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**

- **ProteinB_Tyrp** and **ProteinC_SH2**
Step 3: Explicit Encoding of Local Position Changes

Case 1: Movement of components as a single entity

Case 2: Movement of components as complex
Case 2: Movement of components as complex

- Determine all possible complexes
  - e.g. Protein A + Protein B, Protein B + Protein C, Protein A + Protein B + Protein C

- Movement only if the corresponding interaction sites of interacting components in a complex are used and all other are unused
  - e.g. Protein A + Protein B can only move if they interact with each other, but not with Protein C

- Update local positions of all components forming a complex simultaneously
Case 2: Movement of components as complex

- Determine all possible complexes
  - e.g. Protein A + Protein B, Protein B + Protein C, Protein A + Protein B + Protein C
- Movement only if the corresponding interaction sites of a complex are used and all other are unused
  - e.g. Protein A + Protein B can only move if they interact with each other, but not with Protein C
- Update local positions of all components forming a complex simultaneously
Case 2: Movement of components as complex

- Determine all possible complexes
  - e.g. Protein A + Protein B, Protein B + Protein C, Protein A + Protein B + Protein C

- Movement only if the corresponding interaction sites of a complex are used and all other are unused
  - e.g. Protein A + Protein B can only move if they interact with each other, but not with Protein C

- Update local positions of all components forming a complex simultaneously

For each complex:

- add two transitions per axis to increase/decrease the value of the coordinate places of the interacting components in respect to the grid size

- connect places representing the respective interaction via test arcs

- connect places representing interaction with other components via inhibitory arcs
Case 2: Movement of components as complex

Movement of complex ProteinA-ProteinB
Movement of complex ProteinB-ProteinC
Movement of complex ProteinA-ProteinB-ProteinC
Explicit Encoding of Component Instances

• Define a number of instances for each component

  • e.g. numA = 3, numB= 3, numC = 3

• Duplicate the previously defined networks according to the number of instances
Explicit Encoding of Component Instances

• Define a number of instances for each component
  
  • e.g. numA = 3, numB = 3, numC = 3
  
  • Duplicate the previously defined networks according to the number of instances
Explicit Encoding of Component Instances

• Define a number of instances for each component

  • e.g. numA = 3, numB = 3, numC = 3

• Duplicate the previously defined networks according to the number of instances

Application of coloured Petri nets:

  - For each component define a simple colour-set and variable,
  - Add simple colour-sets to places representing non-interaction states

  - For each interaction define a product colour-set based on the respective simple colour-sets
  - Add product colour-sets to all places representing the respective interaction
Applying Coloured Petri Nets

- Define a simple colour-set and a variable for each component

  - **Protein A** -> csProteinA := int, 1-numA; csProteinA A
  - **Protein B** -> csProteinB := int, 1-numB; csProteinB B
  - **Protein C** -> csProteinC := int, 1-numC; csProteinC C
Applying Coloured Petri Nets

• Define a product colour-set for each interaction

• **Protein A + Protein B** -> csProteinA__ProteinB := product csProteinA, csProteinB

• **Protein B + Protein C** -> csProteinB__ProteinC := product csProteinB, csProteinC
Applying Coloured Petri Nets

Module of Protein A

Module of Protein B

Module of Protein C

XY-Position of Protein A

XY-Position of Protein B

XY-Position of Protein C

Restricted Interaction – Only if Protein A (X, Y) = Protein B (X, Y)

Restricted Interaction – Only if Protein B (X, Y) = Protein C (X, Y)
Stochastic Simulation

- Model parameter:
  - 5x5 2D-Grid for each component
  - 3 instances per component
- Model size:
  - |P|=57, |T|=267, |A|=1992
Stochastic Simulation

Components move as one entity while interacting.
Summary

• Spatial Transformation Algorithm

1. Explicit encoding of local positions
2. Local restriction of interactions
3. Explicit encoding of local position changes
4. Explicit encoding of component instances using coloured Petri nets

• No interference with the model structure
  ➡ Transformation is reversible

• Not restricted to discrete space
  ➡ Representation of continuous space by the use of continuous places
Summary

Biomolecular Systems

- Molecular Mechanisms
- Experimental Data
- Boolean Networks
- SBML Models

Modules

- Gene Modules
- mRNA Modules
- Protein Modules
- Protein Degradation Modules
- Causal Influence Modules
- Allelic Influence Modules

Biomodelkit Database

- Upload
- Add Modules to Collection

Model-based Predictions

- Model Composition
- + Algorithmic Mutation
- + Space-Attributes

Wildtype/Alternative Models
- Spatial Models

Experimental Data
- Forward Engineering
- Reverse Engineering
- Transformation

Molecular Mechanisms
- Forward Engineering

Boolean Networks
- Transformation

SBML Models
- Transformation

High-Throughput Analysis

Add Modules to Collection

Set of Modules
Summary

Automatable: Feature in the next release of the BMKdb

Versatile, unifying framework for multi-scale biomodel engineering
Cooperation Partners
Monika Heiner and Co-Workers, BTU Cottbus
David Gilbert, Brunel University London
Fred Scharper and Co-Workers, OvGU Magdeburg
Tim Hucho, University of Cologne
Fei Liu, Harbin Institute of Technology

Projects
Consortium „Modelling of Pain Switches” 2009-2011
Consortium „NoPain” 2013-2015

Graduate School
IMPRS Magdeburg
Recent Publications

Chapter 6
A Petri-Net-Based Framework for Biomodel Engineering

Mary Ann Bläcke, Christian Rohr, Monika Heiner, and Wolfgang Marwan

Abstract: Petri nets provide a unifying and versatile framework for the synthesis and engineering of computational models of biochemical reaction networks and of gene regulatory networks. Starting with the basic definitions, we provide an introduction into the different classes of Petri nets that represent a Petri net graph as a qualitative, stochastic, continuous, or hybrid model. Static and dynamic analysis in addition to simulation model checking provide a rich choice of methods for the analysis of the structure and dynamic behavior of Petri net models. Coloring of Petri nets of all classes is powerful for multiscale modeling and for the representation of location and space in reaction networks since it combines the concept of Petri nets with the computational mightiness of a programming language. In the context of the Petri net framework, we provide two most recently developed approaches to biomodel engineering, the database-assisted generation of Petri nets with the help of reusable, automated reconstruction of networks based features the framework provides multiple options of systems and synthetic biology.

Keywords: Automatic network reconstruction - Systems modelling - Module modelling - Petri nets - Reverse engineering

Chapter 7
BioModel Engineering with Petri Nets

Mary Ann Bläcke1, Monika Heiner2 and Wolfgang Marwan1

1Otto-von-Guericke University, Magdeburg, Germany. 2Brandenburg University of Technology, Cottbus, Germany

7.1 INTRODUCTION
BioModel Engineering

Systematically constructing, maintaining, and deploying artifacts are typical attributes of sound engineering, independent of the application field. Along these lines, BioModel Engineering is the science of engineering computational models of biochemical processes in an efficient, sustainable, and trustworthy manner.

In Systems Biology, models are used to describe our abstract understanding of biochemical processes and to predict their behavior. For example, in response to perturbations like mutations, chemical interventions, or changes in the environment. In Synthetic Biology, models support the reliable design and redesign of molecular networks and may serve as design templates for experiments. As models with high explanatory and predictive power.

In this chapter, we will explore how this area, the unifying power of Petri nets, serves as an underlying paradigm. An introduction to transaction pathways can be found in [1]; a more detailed overview is given in [2].

Petri Nets

The basic ideas of Petri nets, as we understand them in 1962 [3]: Petri nets are basically a formal language for semantics. They permit the modeling of relations and transitions in the Petri net terminology and typically interpreted as an (bio-)chemical species, such as genes, whereas events model (bio-)chemical reactions at a regulatory activation and deactivation, transport, or stoichiometry of (bio-)chemical reactions or the amount of a structure of a Petri net is unambiguous and explicit for concomity.

Enhancing these key modeling principles by the context of programming languages, yields colored Petri nets, a “discrete data type.” They are particularly strong, as similar processes in similar components.

Petri nets may represent species on different abstractions, such as multicellular organisms and populations. This allows...
Scaling Properties

- The unfolded model scales with:
  - number of axes (1D, 2D, 3D)
  - number of instances and the resulting number of complexes among the components

<table>
<thead>
<tr>
<th>Number of Instances</th>
<th>n = 1</th>
<th>n = 5</th>
<th>n = 10</th>
<th>n = 50</th>
<th>n = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Places</td>
<td>15</td>
<td>115</td>
<td>330</td>
<td>5650</td>
<td>21300</td>
</tr>
<tr>
<td>Transitions</td>
<td>31</td>
<td>895</td>
<td>5440</td>
<td>533200</td>
<td>4131400</td>
</tr>
<tr>
<td>Arcs</td>
<td>160</td>
<td>6880</td>
<td>42760</td>
<td>4253800</td>
<td>33007600</td>
</tr>
<tr>
<td>Unfolding Time (s)</td>
<td>0.124</td>
<td>0.305</td>
<td>1.536</td>
<td>180.763</td>
<td>1608.802</td>
</tr>
</tbody>
</table>

CPU: 2 x 2.66 GHz 6-Core Intel Xeon, RAM: 24 GB 1333 MHz DDR3 ECC, OS: MacOS X 10.10, Software: Snoopy (IDD-based unfolding)