

SBML extension: graphic notation

(proof of the concept)



Fedor Kolpakov

Institute of Systems Biology
(spin-off of DevelopmentOnTheEdge.com)

Laboratory of Bioinformatics,
Design Technological Institute of Digital Techniques

Novosibirsk, Russia

Graphic notation versus graph layout

- allows edit diagram
- allows to create new diagram
- different graphic notations can be applied to the same SBML model
- allows formally define SBGN and use it in SBML models
- allows to reuse graphic notation by many tools

- Formal definition of graphic notation as XML document.
- Integration between SBML format and graphic notation definition.
- General architecture of software for rendering (visualization) of models and database diagrams using specified graphic notation.
- Process Diagrams as a test case.

BioUML diagram type concept

Diagram type defines:

- types of biological components and their interactions that can be shown on the diagram;
- diagram view builder - it is used to generate view for each diagram element taking into account problem domain peculiarities;
- semantic controller - provides semantic integrity of the diagram during its editing

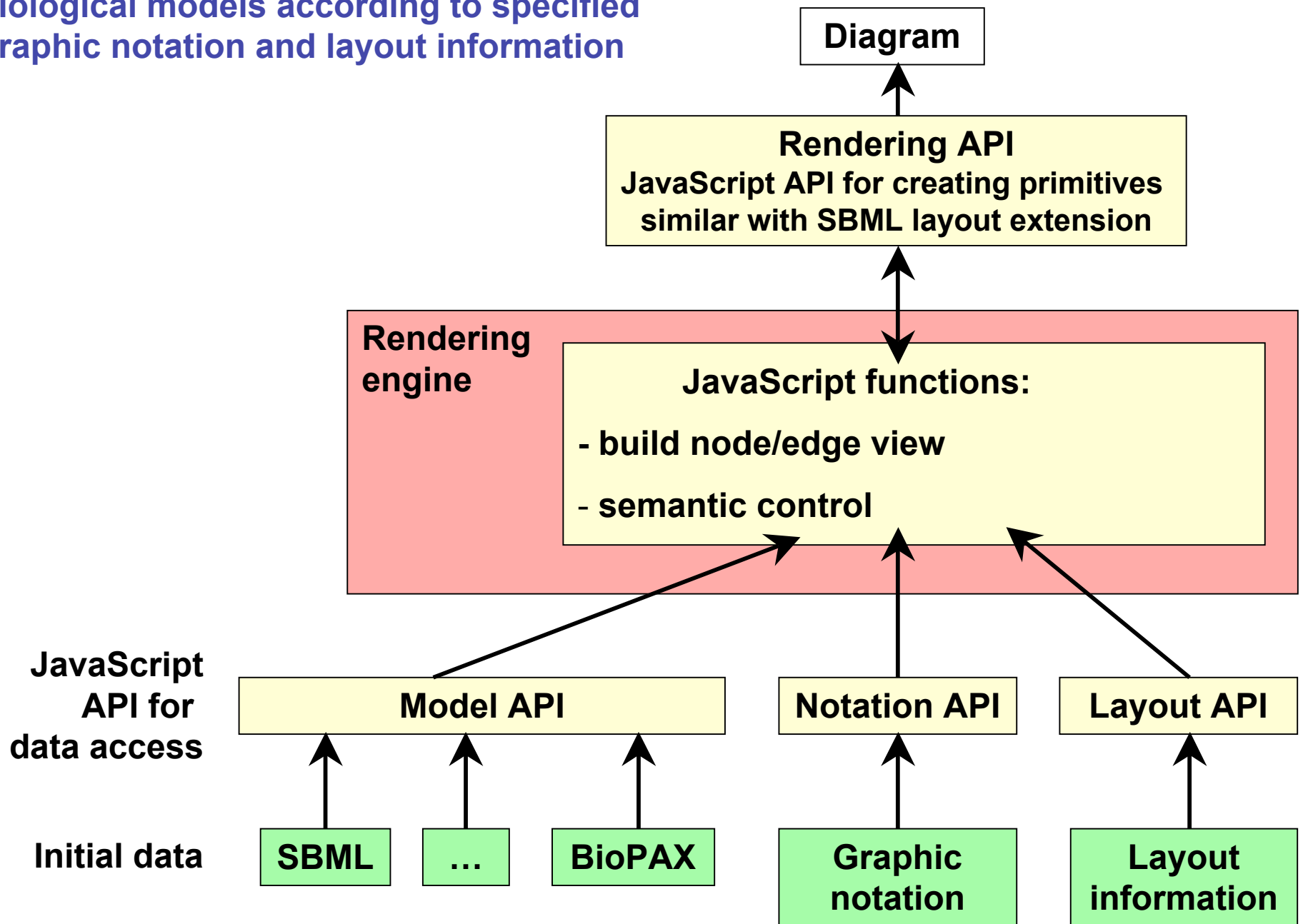
Definition of graphic notation

- Object types
(for example, protein, gene, reaction)
- Object properties
(for example, phosphorylated)
- User defined object properties - layout information
(for example, size, color, location)
- Rules for objects and their properties visualization
- Rules for semantic control of diagram integrity
- View options
- Test cases

Formal definition of graphic notation as XML document and integration with SBML format

Graphic notation components	Defined as	SBML
Object types	XML	<annotation s>
Object properties	XML	<annotation s>
User defined properties	XML	<annotations >
Rules for visualization	JavaScript	
Rules for semantic control	JavaScript	
Test cases	XML	model, module

Basic software architecture for rendering of biological models according to specified graphic notation and layout information



BioUML workbench

File Help

Modules Data Plugins

SBML model repository : kitanoExample.xml

protein1 gene1 rna1 phenotype1 substance1

protein2 gene2 rna2 phenotype2 C1

protein3 gene3 rna3 phenotype3 substance3

protein4 gene4 rna4 unknown1 unknown2

protein5 gene5 rna5 C3

protein6 gene6 rna6 C2

gene7 rna7 substance5

compartment3

compartment2

Node

Title: compartment3

Comment:

Size: Dimension (width = 50, height = 50)

Role:

View Edit Editors

WARN : Could not recover graph node edges after transform

WARN : Could not recover graph node edges after transform

WARN : Could not recover graph node edges after transform

WARN : Could not recover graph node edges after transform

WARN : Could not recover graph node edges after transform

Description Microarray Application Log Clipboard Layout






Parameters Variables JAVA MATLAB JavaScript

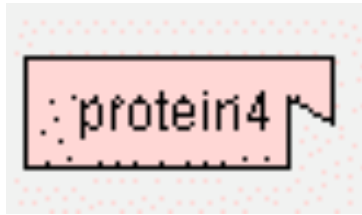
The screenshot displays the BioUML workbench interface. The main window shows an SBML model repository named 'kitanoExample.xml' containing a collection of biological entities: proteins (protein1-6), genes (gene1-7), RNAs (rna1-7), phenotypes (phenotype1-3), substances (substance1-5), and compartments (compartment2, compartment3). Each entity is represented by a distinct shape and color. A 'Node' panel is open at the bottom left, showing details for 'compartment3', including its title, comment, size, and role. A warning log at the bottom right indicates that the software could not recover graph node edges after a transform operation. The interface also includes a file explorer on the left, a menu bar (File, Help), and a toolbar with various icons for file operations and editing.

Examples:

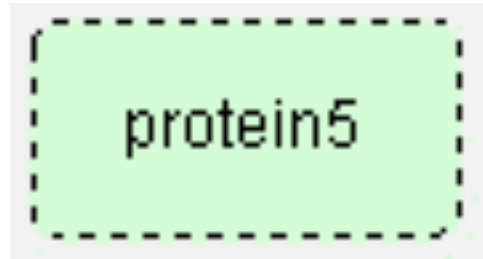
http://www.biouml.net/_sbml/

Index of /_sbml

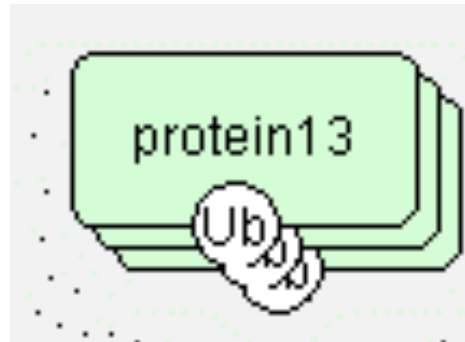
	<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
	Parent Directory	06-Oct-2007 21:00	-	
	graphic notation.dtd	06-Oct-2007 20:52	3k	
	kitano.xml	06-Oct-2007 20:52	36k	
	kitanoExample.png	06-Oct-2007 20:52	24k	
	kitanoExample.xml	06-Oct-2007 20:52	30k	



```
<specie name="protein4">  
  <annotations xmlns:biouml="http://www.biouml.org/ns">  
    <biouml:nodeInfo x="10" y="220">  
      <property name="proteinType" value="truncated"/>  
    </biouml:nodeInfo>  
    <biouml:specieInfo type="molecule-protein"/>  
  </annotations>  
</specie>
```



```
<specie name="protein5">  
  <annotations xmlns:biouml="http://www.biouml.org/ns">  
    <biouml:nodeInfo x="10" y="290">  
      <property name="proteinType" value="generic"/>  
      <property name="hypothetical" value="true"/>  
    </biouml:nodeInfo>  
    <biouml:specieInfo type="molecule-protein"/>  
  </annotations>  
</specie>
```



```
<specie name="protein13">  
  <annotations xmlns:biouml="http://www.biouml.org/ns">  
    <biouml:nodeInfo x="230" y="710">  
      <property name="proteinType" value="generic"/>  
      <property name="homomultimer" value="3"/>  
      <property name="residues" type="array">  
        <property name="residue1" type="composite">  
          <property name="id" value="id1"/>  
          <property name="name" value="name1"/>  
          <property name="angle" value="-1.5"/>  
          <property name="modification" value="ubiquitinated"/>  
        </property>  
      </property>  
    </biouml:nodeInfo>  
    <biouml:specieInfo type="molecule-protein"/>  
  </annotations>  
</specie>
```

```

<node icon="substance" type="molecule-substance">
  <propertyRef name="title"/>
  <propertyRef name="moleculeType"/>
  <propertyRef name="hypothetical"/>
  <propertyRef name="homomultimer"/>
  <propertyRef name="size"/>
  <propertyRef name="fill"/>
  <propertyRef name="border"/>
</node>

```

```

<specie name="C3">
  <annotations xmlns:biouml="http://www.biouml.org/ns">
    <biouml:nodeInfo x="450" y="360">
      <property name="moleculeType" value="ion"/>
      <property name="homomultimer" value="2"/>
    </biouml:nodeInfo>
    <biouml:specieInfo type="molecule-substance"/>
  </annotations>
</specie>

```

```

<nodeView type="molecule-substance">
<![CDATA[ function createMoleculeView(container, node, options, g)
{
  var title = new TextView(node.getTitle(), options.getNodeTitleFont(), g);
  var type = node.getAttributes().getValue('moleculeType');

  var homomultimer = 1;
  if( node.getAttributes().getValue('homomultimer') != nul l)
    homomultimer = node.getAttributes().getValue('homomultimer');

  var d, pen, brush;
  for(i=0; i<homomultimer; i++)
  {
    var composite = new CompositeView();
    if(type == 'ion')
    {
      d = new Dimension(25,25);
      brush = new Brush(new Color(0.6, 0.6, 1.0));
      pen = options.getDefaultPen();
      if( node.getAttributes().getValue('hypothetical') == 'true')
        pen = new Pen(new BasicStroke(1, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, 10,
          new Array (3,3), 0), pen.getColor());
      var ellipse = new EllipseView(pen, brush, 0, 0, d.width, d.height);
      composite.add(ellipse);
    }
    ...
    var textPosition = new Point((d.width-title.getBounds().width)/2, (d.height-title.getBounds().height)/2);
    container.add(title, CompositeView.X_LL | CompositeView.Y_TT, textPosition);
    return false;
  }
}
]]>

```



ru.biosoft.graphics

http://www.biouml.org/developer/javadoc_together/index.html

is a framework providing an API for painting all graphical elements when visualizing the biological data.

Main goals when introducing Graphics package were the following:

- Provide an API for drawing useful 2D graphical primitives such as line, polygon, oval, text, ruler etc.
- Provide mechanism for drawing complex images typical comprised form basic elements
- Provide hierarchical mechanism for managing graphical elements of complex images
- Provide mechanism for passing events to/from graphical objects and association of these graphical object with logical components of an application

View

View interface (abstract class in terms of Java) provides methods for drawing as well as some useful methods. This can be imagined as a root of class hierarchy of the Graphics package. Every object that intended to be painted must have another object extending View interface associated with it or must extend View itself.

```
public abstract class View
{
    protected Shape shape = null;
    protected int type = 0;
    protected Object model = null;

    public View(Shape shape);

    public Object getModel();
    public void setModel(Object model);

    public Rectangle getBounds();
    public void update Bounds(){}

    public boolean intersects(Rectangle rect);

    public void setLocation(int x,int y);
    public void setLocation(Point pt);

    abstract public void move(int x,int y) ;

    public void move(Point offset);
    public boolean isActive();
    public void setActive(boolean isActive)

    public boolean isVisible();
    public void setVisible(boolean isVisible)

    public void paint(Graphics2D g2);
}
```

Primitive Views

The following standard views are typically used for painting complex images being organized in component views . These components typically override *paint* method to provide customized behavior.

LineView – primitive for drawing lines;

OvalView – primitive for drawing circles/ovals;

BoxView – primitive for drawing rectangles;

TextView – primitive for drawing text strings;

PolygonView - primitive for drawing polygons

PolylineView – primitive for drawing non -closed polygons

The following classes are *CompositeViews*

Ruler – used to paint rulers when visualizing maps

ArrowView – used to paint arrow like figures

Resources

In fact, views are relatively high-level components representing some models i.e. an object having application defined logical meaning. In order to perform painting operations in its *paint* methods View may use the following “helper” classes:

Pen - logical pen

Brush – logical brush

ColorFont – logical font with color

CompositeView

CompositeView interface contains methods for organizing visual components into Composite pattern.

It is defined as follows:

```
public class CompositeView extends View
{
    public Enumeration getChildren();
    public View elementAt(int index);
    public int size();

    public void insert(View v, int i) throws ArrayIndexOutOfBoundsException;
    public void add(View v, int mode);
    public void add(View v, int mode, Point insets);
    public boolean remove(View v);
    ...
}
```

Method **add** performs some basic layout of the components being added using the following algorithm which is chosen via *mode* parameter

```
X_RL
|  ||
|  |--- boundary of new element
|  ---- boundary of minimal rectangle, described all
|         previous elements
----- x or y coordinate
```

X - the x coordinate:

L - left boundary of the object

C - center of the object

R - right boundary of the object

Y - the y coordinate:

T - top boundary of the object

C - center of the object

B - bottom boundary of the object

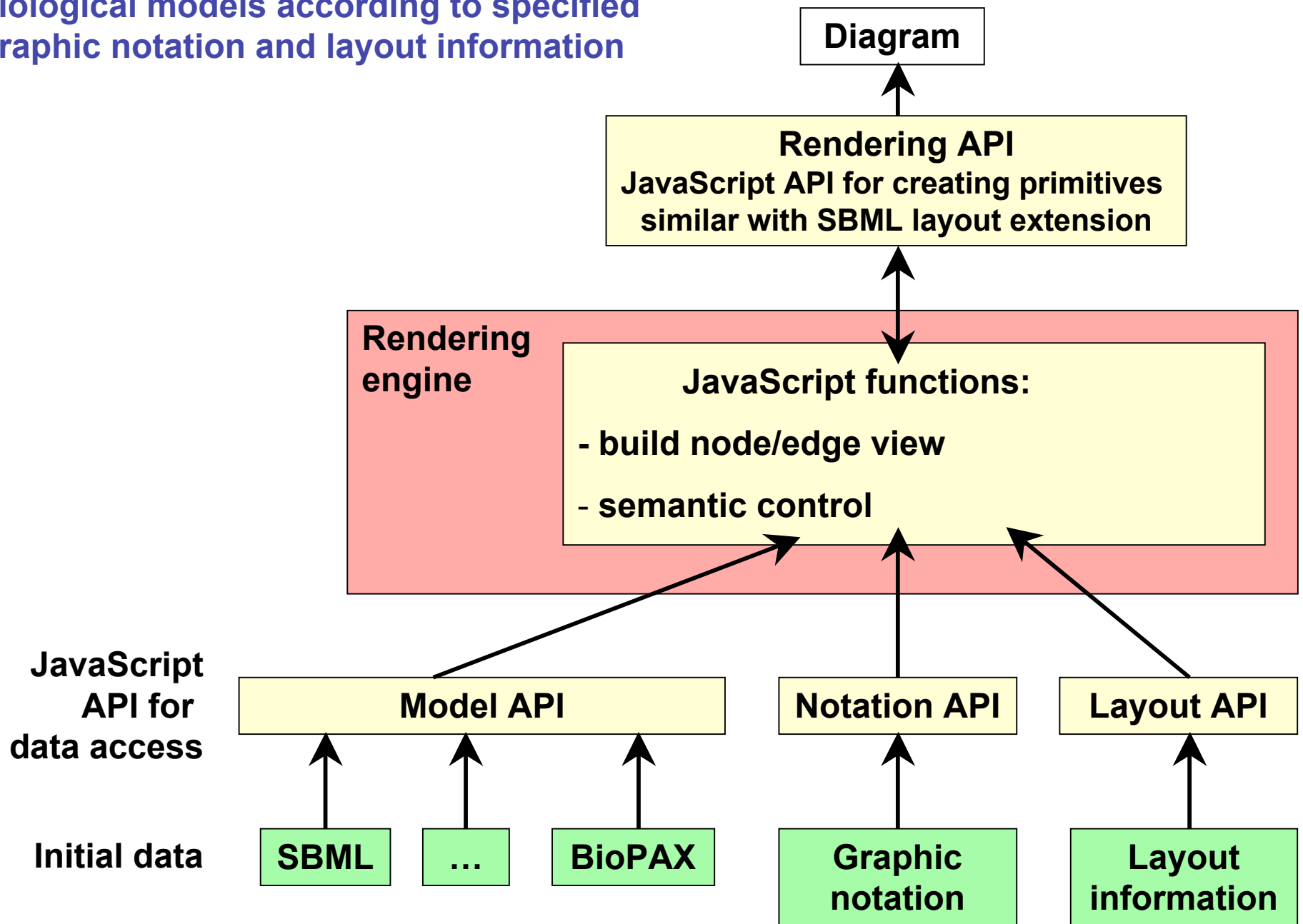
Special:

UN - don't change the corresponding x or y boundary of new element.

REL - If this bit is cleared, target coordinates is used from location of view.

For example when you specify **Y_BT** new components is added below the previous i.e **(B)ottom -(T)op**.

Basic software architecture for rendering of biological models according to specified graphic notation and layout information



Java library for SBML graphics notation extension

To simplify support of graphic notation extension.

May include:

- `ru.biosoft.graphics`
 - primitives for painting
 - interactive graphics pane (optional)
- `biouml.model` – access to
 - SBML models
 - properties declared in annotations
 - graphic notation
- `biouml.gui` – dialogs, editors
 - relation, reaction dialogs
 - node/edge properties dialog
 - property editors

Acknowledgements

Part of this work was partially supported by
European Committee grant №037590
“Net2Drug”

Software developers

Nikita Tolstyh

Mikhail Puzanov

Denis Ryumin