Including Layout Information in SBML Files Version 2.1.2

Ralph Gauges, Ursula Rost, Sven Sahle and Katja Wegner
European Media Laboratory
Schloss-Wolfsbrunnen Weg 33
69118 Heidelberg
Germany

April 13, 2005

Introduction

With SBML there now is a common standard for the exchange of dynamical systems data which has already been adopted by many applications in this field [1, 2, 3]. Since SBML had no means of storing layout information for reaction networks, we developed an extension to SBML that would allow us to store this layout information in SBML files. This extension was presented on the SBML workshop in St. Louis (November 2003). According to the discussion at the workshop some simplifications were made to this proposal. This simplified version is presented here.

Design principles and general structure

The overall structure of this proposal reflects some design decisions that will be explained in this paragraph. These decisions are mainly based on the discussion on the mailing list and during the workshop in St. Louis.

First it was requested that it should be possible to have several layouts in one SBML file. This leads to the obvious choice to have a **listOfLayouts** outside the **model** part of the SBML file instead of direct annotations to the model elements.

We think it is important that dealing with this SBML extension should be as easy as possible. This leads us to some other decisions:

The layout of a reaction network diagram should be described as graphical representations of species and reactions (and not as some arbitrary drawing or graph). This means that existing languages for the description of vector drawings (SVG) or general graphs cannot be used. While it may seem unnecessary to invent a new language when an existing one like SVG could in principle be used to describe the layout of a reaction network we think there are good reasons to have a language tailored specifically for the layout of SBML models. Presumably most programs that will use this SBML extension are primarily programs dealing with biochemical models. So internally most programs will have data structures like species and reactions (this is obviously also the reason why SBML is structured that way). So for these programs it is natural to describe the layout of the reaction network also in terms of species and reactions (and not in terms of polygons or splines). This leads to the **layout** object having a structure very similar to the structure of an SBML **model** object. It basically contains lists of graphical representations of compartments, species, and reactions (called **compartmentGlyph, speciesGlyph,** and **reactionGlyph** respectively).

Another important question is the level of detail that the description

should provide. For simplicity we focus only on the layout of the diagram, not the details of how it should be rendered. This means basically the position of the different graphical objects on the screen or on paper is given. As an illustration consider the following figure:

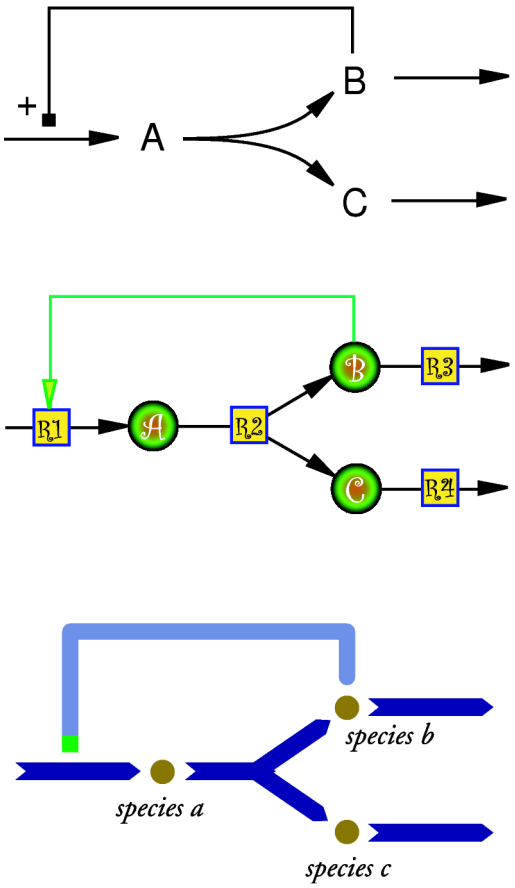

Figure 1: Illustration of different renderings of the same layout

All three diagrams could be renderings of the same layout and would be described by identical SBML files. No information about colors, line styles, fonts, etc., is present in the layout description as we propose it.

The next question is how the relation between the model and the layout should be established. There seems to be consensus that one model element can be represented by several layout elements. For example it can be useful to have several representations of one species in the layout to avoid lots of crossing arrows. This can be accomplished if every layout element has a field that refers to the id of a model element.

We also think that there are cases where a layout element does not correspondent to exactly one model element. This could occur if the layout shows a simplified version of the model where one reaction in the layout correspondents to several reactions and intermediate species in the model. This is the reason why the field in the layout elements that refers to the model elements is optional, allowing layout objects that do not have one specific counterpart in the SBML model.

The result of all this is a way to describe a graphical layout of a reaction network in biochemical terms. This layout can be closely tied to the biochemical model. A graphical model editor for example would typically create a layout that is closely connected (by a one-to-several relation from the model elements to the layout elements) to the model. A more general layout design program could also create a layout that is not so closely tied to the model, for example it could create a layout that shows a simplified version of the model.

All size information given for layout objects are understood to be Pt, which is defined to be 1/72 of an inch.

Nomenclature

The UML diagrams in this document show the name of the class on top. Below are the attributes specific to that class. Optional attributes have some default value which may be NULL. Arrays are written in square brackets where with the valid array length within those brackets. So a array [2..] would mean that it can hold from 2 to ∞ number of objects (assuming that you have some very large harddisk).

Inheritance tree

Figure 2 shows the relations between the different objects. Inheritance relations are shown as well as which object contains which other objects.

Namespace

For the extensions we use a separate namespace of the following form xmlns:sl2="http://projects.eml.org/bcb/sbml/level2". An SBML file that would utilize the extension could have the following form:

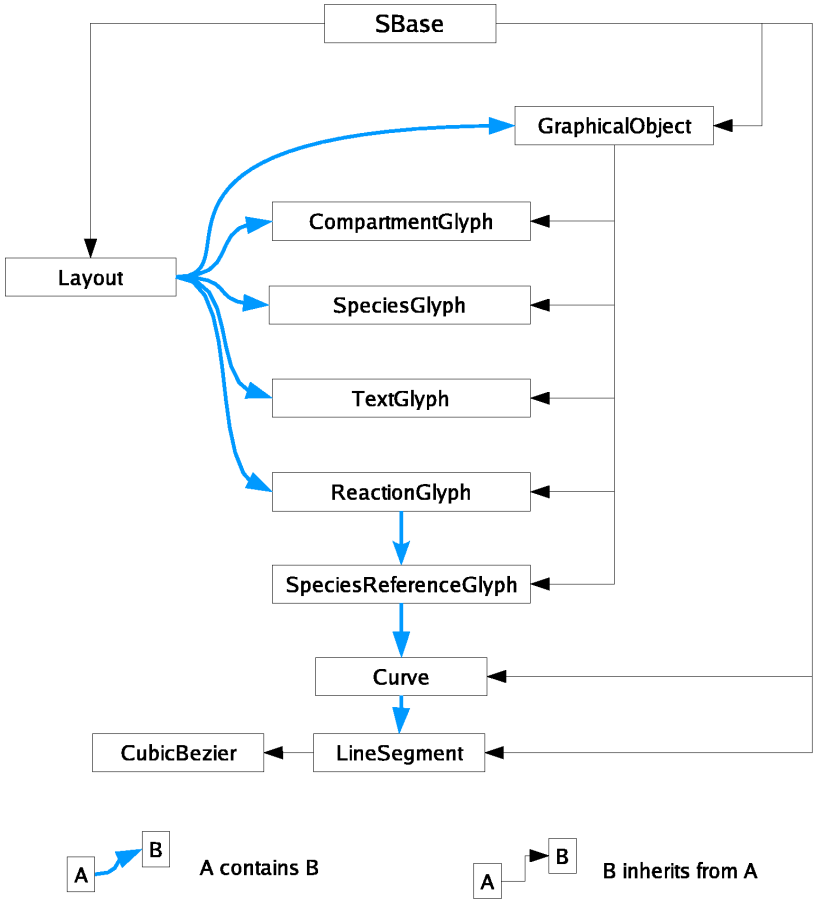

Figure 2: Containment and inheritance tree for the layout classes

```
<?xml version="1.0" encoding="UTF-8"?>
   <sbml xmlns:sbml="http://www.sbml.org/sbml/level2" level="2" \\
         version="1"
         xmlns:sl2="http://projects.eml.org/bcb/sbml/level2"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://projects.eml.org/bcb/sbml/level2
         http://projects.eml.org/bcb/sbml/level2/layout2.xsd">
```

Metainformation

All the layout classes below are derived from a class called SBase which was taken from the SBML Level 2 schema specification (http://www.sbml.org/sbml/level2/version1/). This enables programs to store metainformation with the layout objects.

SBase		
metaid	:	ID {use=”optional”}
notes	:	(ANY : {namespace=”http://www.w3.org/1999/xhtml”}) {minOccurs=”0”}
annotation	:	(ANY) {minOccurs=”0”}

XML Schema representation:

```
<xsd:complexType name="SBase" abstract="true">
 <xsd:sequence>
  <xsd:element name="notes" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:any namespace="http://www.w3.org/1999/xhtml"
              processContents="skip"
              maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
  <xsd:element name="annotation" minOccurs="0">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:any processContents="skip" maxOccurs="unbounded"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="metaid" type="xsd:ID" use="optional"/>
</xsd:complexType>
```

Coordinate System

The layout extension uses a Cartesian coordinate system. The origin of the coordinate system will be in the upper left corner of the screen. The positive x-axis runs from left to right, the positive y-axis run from top to bottom and the positive z-axis points into the screen. The reason to have the origin in the upper left corner of the screen is that most 2D daring packages do it that way. This coordinate system is also right handed just like the ones in OpenGL and Java3D, which should facilitate 3D implementations as well. For printing purposes a point in this coordinate system is presumed to be 1/72 of an inch (0.3527777778 mm) as in postscript.

<listOfLayouts > and <layout >

All layout information is stored in a tag called listOfLayouts which is placed within the annotation tag of the model tag. This list can hold one or more layout objects which in turn hold layout information for some or all elements of the sbml model plus additional objects that need not be connected to the model. In the <layout > tag an id which uniquely identifies it and the dimensions of the bounding box have to be specified. The dimensions of the bounding box are given as a width, height and an optional depth attribute, all of type double. If not specified, the depth value defaults to 0.0. Ids are defined to be the same as SId in SBML Level 2. The actual layout elements are contained in several lists in the <layout > tag, namely a **ListOfCompartmentGlyphs**, a **ListOfSpeciesGlyphs**, a **ListOfReactionGlyphs**, a **ListOfTextGlyphs**, and a **ListOfAdditionalGraphicalElements**.

ListOfLayouts		
layouts	:	Layout[1..∗]

Layout		
id	:	SId {use=”required”}
dimensions	:	Dimensions {use=”required”}
compartmentGlyphs	:	CompartmentGlyph[0..∗]
speciesGlyphs	:	SpeciesGlyph[0..∗]
reactionGlyphs	:	ReactionGlyph[0..∗]
textGlyphs	:	TextGlyph[0..∗]
additionalGraphicalObjects	:	GraphicalObject[0..∗]

XML Schema representation:

```
<xsd:complexType name="ListOfCompartmentGlyphs">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
    <xsd:element name="compartmentGlyph" type="CompartmentGlyph"
                 maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfSpeciesGlyphs">
 <xsd:complexContent>
  <xsd:extension base="SBase">
```

```
   <xsd:sequence>
    <xsd:element name="speciesGlyph" type="SpeciesGlyph"
                 maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfReactionGlyphs">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
    <xsd:element name="reactionGlyph" type="ReactionGlyph"
                 maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfTextGlyphs">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
    <xsd:element name="textGlyph" type="TextGlyph" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfAdditionalGraphicalObjects">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
    <xsd:element name="additionalGlyph" type="GraphicalObject"
                 maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Layout">
  <xsd:sequence>
    <xsd:element name="dimensions" type="Dimensions" />
    <xsd:element name="listOfCompartmentGlyphs" type="ListOfCompartmentGlyphs"
                 minOccurs="0"/>
    <xsd:element name="listOfSpeciesGlyphs" type="ListOfSpeciesGlyphs"
                 minOccurs="0"/>
    <xsd:element name="listOfReactionGlyphs" type="ListOfReactionGlyphs"
                 minOccurs="0"/>
```

```
    <xsd:element name="listOfTextGlyphs" type="ListOfTextGlyphs"
                 minOccurs="0"/>
    <xsd:element name="listOfAdditionalGraphicalObjects"
                 type="ListOfAdditionalGraphicalObjects" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="id" type="SId"/>
</xsd:complexType>

<xsd:complexType name="ListOfLayouts">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
    <xsd:element name="layout" type="Layout" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:element name="listOfLayouts" type="ListOfLayouts"/>
```

<GraphicalObject>

Most objects for which layout information is to be included in an SBML file have a corresponding object in the SBML model. As there might be cases where the user wants to include object types in the layout that do fall in any of the other categories described below, we include a listOfAdditionalGraphicalObjects in each layout object. This list holds an arbitrary number of **graphicalObject** elements. The **graphicalObject** basically defines a bounding box in a specific place in the layout without giving additional information about its contents. All the more specific layout elements (CompartmentGlyph, SpeciesGlyph, ReactionGlyph, TextGlyph, and SpeciesReferenceGlyph) are derived from **graphicalObject**.

The GraphicalObject has an id attribute of type SId through which it can be identified. Each GraphicalObject has an element called boundingBox, which specifies the position and the size of the object. Each BoundingBox has an element called position which is of type Point and an element called dimensions of type Dimensions. The position always specifies the upper left corner of the bounding box. Additionally BoundingBox has an optional attribute id that can be used to identify and reference the BoundingBox object. The Point element has four attributes. The first attribute id is optional and can be used to identify a point object. The other three attributes are called x, y and z and they specify the x- ,y- and z-coordinate of the point.

The z coordinate is optional and defaults to 0.0. Likewise the Dimensions element has an attribute id, which is optional and the attributes width, height and depth. The width specifies the size of the object in the direction of the positive x axis, the height attribute specifies the size of the object along the positive y axis and the depth attribute specifies the size of the object along the positive z axis. Again, the depth attribute is optional and defaults to 0.0. All sizes for Dimension objects are positive values.

Programs can use annotations to graphicalObjects in the listOfGraphicalObjects to describe program specific graphical objects.

Point		
id	:	SId {use="optional"}
x	:	double
y	:	double
z	:	double {use="optional" default="0.0"}

Dimensions		
id	:	SId {use="optional"}
width	:	double
height	:	double
depth	:	double {use="optional" default="0.0"}

BoundingBox		
id	:	SId {use="required"}
position	:	Point
dimensions	:	Dimensions

GraphicalObject		
id	:	SId {use="required"}
boundingBox	:	BoundingBox

XML Schema representation:

```
<xsd:complexType name="Point">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:attribute name="id" type="SId" use="optional" />
   <xsd:attribute name="x"  type="xsd:double" />
   <xsd:attribute name="y"  type="xsd:double" />
   <xsd:attribute name="z"  type="xsd:double" use="optional" default="0.0" />
  </xsd:extension>
 </xsd:complexContent>
```

```
</xsd:complexType>

<xsd:complexType name="Dimensions">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:attribute name="id" type="SId" use="optional" />
   <xsd:attribute name="width"  type="xsd:double" />
   <xsd:attribute name="height" type="xsd:double" />
   <xsd:attribute name="depth"  type="xsd:double" use="optional" default="0.0" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="BoundingBox">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
     <xsd:element name="position" type="Point"/>
     <xsd:element name="dimensions" type="Dimensions/>
   </xsd:sequence>
   <xsd:attribute name="id" type="SId" use="optional" />
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="GraphicalObject">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
     <xsd:element name="boundingBox" type="BoundingBox" />
   </xsd:sequence>
   <xsd:attribute name="id" type="SId"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

<Compartment> Layout Information

The CompartmentGlyph class is derived from GraphicalObject and has the same attributes. Additionally it has an optional reference to the id of the corresponding compartment in the model. Since the compartment id is optional, the user can specify compartments in the layout that are not part of the model.

CompartmentGlyph		
compartment	:	SId {use=”optional”}

XML Schema representation:

```
<xsd:complexType name="CompartmentGlyph">
 <xsd:complexContent>
  <xsd:extension base="GraphicalObject">
   <xsd:attribute name="compartment" type="SId" use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

example:

```
.
.
.
<listOfCompartmentGlyphs>
    <compartmentGlyph id="cGlyph" compartment="compartment" >
      <boundingBox>
        <position x="10.0" y="10.0" />
        <dimensions width="60.0" height="50.0" />
      </boundingBox>
    </compartmentGlyph>
</listOfCompartmentGlyphs>
.
.
.
<compartment id="compartment" size="1.0"/>
.
.
.
```

<Species> Layout Information

Since an sbml document can contain species that don't appear in any reaction a species can have zero or more representations on screen which are represented by <speciesGlyph> and are grouped in a <listOfSpeciesGlyphs> tag. In addition to the attributes from GraphicalObject, the speciesGlyph object has a **species** attribute which is the id of the corresponding species object in the model. The **species** attribute is optional to allow the program to specify species representations that do not have a direct correspondence in the model. This might be useful if some pathway has been collapsed, but is still treated by layout programs.

XML Schema representation:

```
<xsd:complexType name="SpeciesGlyph">
```

SpeciesGlyph		
species	:	SId {use=”optional”}

```
 <xsd:complexContent>
  <xsd:extension base="GraphicalObject">
   <xsd:attribute name="species" type="SId" use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

example:

```
            .
            .
            .
    <listOfSpeciesGlyphs>
      <speciesGlyph id="ATP_Glyph" species="ATP">
        <boundingBox>
          <position x="295.0" y="123.0" />
          <dimensions width="16.0" height="8.0" />
        </boundingBox>
      </speciesGlyph>
            .
            .
            .
    </listOfSpeciesGlyphs>
            .
            .
            .
    <species id="ATP" compartment="compartment" initialAmount="0">
            .
            .
            .
```

<Reaction> Layout Information

Reactions are represented by a **reactionGlyph** object in the layout. Since this is also a subclass of **graphicalObject** it inherits a bounding box. Just like the other glyphs also the **reactionGlyph** has a **reaction** attribute that specifies the id of the corresponding reaction in the model. Again, this reference is optional.

Since one species can have several graphical representations in the layout there must be a way to specify which **speciesGlyph** should be connected to the **reactionGlyph**. For this reason there is a **listOfSpeciesReferenceGlyphs** (see below) in the **reactionGlyph**.

Since the dimensions of a bounding box can no longer be negative, an optional curve attribute was added to ReactionGlyph. This Curve object (see decription below) can be used to describe a curve representation for the ReactionGlyph. If a ReactionGlpyh specifies a curve, the bounding box is to be ignored.

ReactionGlyph		
reaction	:	SId {use=”optional”}
curve	:	Curve {use=”optional”}
speciesReferences	:	SpeciesReferenceGlyph[1..∗]

XML Schema representation:

```
<xsd:complexType name="ListOfSpeciesReferenceGlyphs">
 <xsd:complexContent>
  <xsd:extension base="SBase">
   <xsd:sequence>
    <xsd:element name="speciesReferenceGlyph" type="SpeciesReferenceGlyph"
                 minOccurs="1" maxOccurs="unbounded"/>
   </xsd:sequence>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ReactionGlyph">
 <xsd:complexContent>
  <xsd:extension base="GraphicalObject">
   <xsd:sequence>
     <xsd:element name="curve" type="Curve" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
   <xsd:sequence>
    <xsd:element name="listOfSpeciesReferenceGlyphs"
                 type="ListOfSpeciesReferenceGlyphs"
                 minOccurs="1" maxOccurs="1"/>
   </xsd:sequence>
   <xsd:attribute name="reaction" type="SId" use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

example:

```
            .
            .
            .
    <listOfReactionGlyphs>
```

```
  <reactionGlyph id="reaction_0_Glyph" reaction="reaction_0" >
    <curve>
      <listOfCurveSegments>
        <curveSegment xsi:type="LineSegment">
            <start x="119.0" y="166.0" />
            <end   x="239.0" y="349.0" />
        </curveSegment>
      </listOfCurveSegments>
    </curve>
  </reactionGlyph>
        .
        .
        .
</listOfReactionGlyphs>
        .
        .
        .
<reaction id="reaction_0" reversible="false">
  <listOfReactants>
     <speciesReference species="P_i" stoichiometry="1"/>
        .
        .
        .
</reaction>
        .
        .
        .
```

<SpeciesReference> Layout Information

The graphical connection between a speciesGlyph and a reactionGlyph (which would be an arrow or some curve in most cases) is represented by the **speciesReferenceGlyph** object. A **listOfSpeciesReferenceGlyphs** is contained in a **reactionGlyph**.

SpeciesReferenceGlyph		
speciesGlyph	:	SId {use="optional"}
speciesReferences	:	SpeciesReferenceGlyph {use="optional"}
role	:	SpeciesReferenceRole {use="optional" default="UNDEFINED"}
curve	:	Curve {use="optional"}

The **speciesReferenceGlyph** has a **speciesGlyph** attribute that contains the id of a speciesGlyph object that is to be connected to the reactionGlyph. The **speciesReference** attribute refers to a speciesReference

(or a modifierSpeciesReference) in the model and is optional. Since species references in sbml level 1 as well as level 2 do not have ids, we choose to put a new tag called id which has an attribute called id that is of type SId into the annotation part of the corresponding SpeciesReference element. This tag has to be unique within the global namespace of the SBML model and can thus be used to reference a given species reference. In a later version of SBML the speciesReferences should have an id.

This id defines a relation between a speciesReferenceGlyph and the corresponding speciesReference or modifierSpeciesReference in the model. From that we can also deduce what role a certain species plays in the reaction (whether it is a substrate or product or something else). Since this connection from the diagram to the model is optional there are cases where the role of the species can not be derived in that way. For that we propose an optional **role** attribute. This can be used to provide the role of a metabolite in the reaction as a string. There was some discussion in St. Louis whether something like this is necessary, so this attribute might get dropped in later revisions.

So far we have defined which graphical objects should be connected to the reaction glyph. This is the minimum information that a render program with biochemical knowledge needs to render the reaction layout. The standard way to render this connection would be a straight line. In most cases the relation of a species to a reaction will be graphically represented by a curve. In this case a **curve** tag that contains a **listOfCurveSegments** can be used. The **listOfCurveSegments** contains an arbitrary number of curve segments. For now we provide the definitions for two types of curve segments (**LineSegment** and **CubicBezier**) but leave it open if this should in future be restricted to only one type or even generalized to more different line types. CubicBezier is a direct subclass of LineSegment. The type of the curve segment has to be specified with a **xsi:type** attribute in the **curveSegment** tag (xsi:type=LineSegment or xsi:type=CubicBezier). The SpeciesReference should either contain a bounding box or a curve specification, if both are given, the bounding box should be ignored.

The LineSegment object consists of two elements of type Point. One is called start and it is the starting point of the line, the other is called end and it specifies the endpoint of the line. As mentioned above, CubicBezier is derived from LineSegment, so it consists of the same two elements start and end, which again specify the starting point and the endpoint of the CubicBezier curve. The two elements basePoint1 and basePoint2 specify the two additional basepoints that are needed to describe a CubicBezier curve. basePoint1 is the basepoint closer to the start point. This way, programmers who do not want to implement CubicBezier curves can just treat them as

straight lines and ignore the basepoints.

Curve		
curveSegments	:	LineSegment[1..∗]

LineSegment		
start	:	Point
end	:	Point

CubicBezier		
basePoint1	:	Point {use="optional"}
basePoint2	:	Point {use="optional"}

The **role** attribute is used to specify how the species reference should be displayed. Allowed values are substrate, product, sidesubstrate, sideproduct, modifier, activator and inhibitor. This attribute is optional and should only be necessary if the optional speciesReference attribute is not given or if the respective information from the model needs to be overridden. The values *substrate* and *product* are used if the species reference is a main product or substrate in the reaction. *sidesubstrate* and *sideproduct* are used for stuff like ATP, NAD+, etc. that some renderers might choose to display as side reactions. *activator* and *inhibitor* are modifiers where their influence on the reaction is known and *modifier* is a more general term if the influence is unknown or changes during the course of the simulation. This list is probably not exhaustive and will be updated as needed. Future versions of SBML may very well have an id for the SpeciesReference objects as well as some kind of role attribute. If this is the case, we will drop both attributes here since they are no longer necessary.

XML Schema representation:

```
<xsd:complexType name="LineSegment">
  <xsd:complexContent>
    <xsd:extension base="SBase">
    <xsd:sequence>
        <xsd:element name="start" type="Point" />
        <xsd:element name="end" type="Point" />
    </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="CubicBezier">
```

```
  <xsd:complexContent>
    <xsd:extension base="LineSegment">
        <xsd:element name="basePoint1" type="Point" minOccurs="0" />
        <xsd:element name="basePoint2" type="Point" minOccurs="0" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ListOfCurveSegments">
  <xsd:complexContent>
    <xsd:extension base="SBase">
      <xsd:sequence>
        <xsd:element name="curveSegment" type="LineSegment"
                          minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="Curve">
  <xsd:complexContent>
    <xsd:extension base="SBase">
      <xsd:sequence>
        <xsd:element name="listOfCurceSegments" type="ListOfCurveSegments"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:simpleType name="RoleString">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="substrate"/>
    <xsd:enumeration value="product"/>
    <xsd:enumeration value="sidesubstrate"/>
    <xsd:enumeration value="sideproduct"/>
    <xsd:enumeration value="modifier"/>
    <xsd:enumeration value="activator"/>
    <xsd:enumeration value="inhibitor)"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="SpeciesReferenceGlyph">
 <xsd:complexContent>
  <xsd:extension base="GraphicalObject">
   <xsd:sequence>
     <xsd:element name="curve" type="Curve" minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
   <xsd:attribute name="speciesGlyph" type="SId" use="optional"/>
   <xsd:attribute name="speciesReference" type="SId" use="optional"/>
```

```
   <xsd:attribute name="role" type="RoleString" use="optional"/>
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

example:

```
        .
        .
        .
    <speciesGlyph id="P_iGlyph" species="P_i_s1" >
      <boundingBox>
        <position x="0.0" y="1.8" />
        <dimensions width="1.0" height="1.0" />
      </boundingBox>
    </speciesGlyph>
        .
        .
        .
    <reactionGlyph id="reaction_0_Glyph" reaction="reaction_0" >
       <curve>
        <listOfCurveSegments>
           <curveSegment xsi:type="LineSegment">
               <start x="2.3" y="1.0" />
               <end   x="3.0" y="1.0" />
           </curveSegment>
        </listOfCurveSegments>
       </curve>
    </reactionGlyph>
      <listOfSpeciesReferenceGlyphs>
        <speciesReferenceGlyph id="SP1_Glyph" speciesGlyph="P_iGlyph"
                                speciesReference="P_i_sr1">
         <curve>
           <listOfCurveSegments>
             <curveSegment xsi:type="CubicBezier">
               <start x="0.0" y="1.8" />
               <end x="0.3" y="0.8" />
               <basePoint1 x="0.1" y="1.9" />
               <basePoint2 x="0.3" y="0.7"/>
             </curveSegment>
             <curveSegment xsi:type="LineSegment">
               <start x="0.3" y="0.8" />
               <end   x="2.3" y="1.0" />
             </curveSegment>
           </listOfCurveSegments>
         </curve>
        </speciesReferenceGlyph>
            .
            .
```

```
        .
      </listOfSpeciesReferenceGlyphs>
    </reactionGlyph>
        .
        .
```

TextLabels

When dealing with text labels we basically have two possibilities: Either let the program decide whether and where to put text labels, or specify it in the sbml file. Since it was argued that the positioning of text labels to avoid collisions with other layout elements was an integral part of the layout we do provide a way to specify the labels.

A **listOfTextGlyphs** in the layout class contains an arbitrary number of **TextGlyphs**. Each **TextGlyph** describes one text label. This can be just an independent text like a title or comment to the diagram, in this case only the **text** attribute of the TextGlyph should be given (and obviously contain the text that should be displayed).

An optional **graphicalObject** attribute contains the SId of any GraphicalObject and specifies that the TextGlyph should be considered to be a label to that object. This could mean the label is moved together with the object in an editor. Additionally there is another optional attribute called **originOfText** which holds the SId of an SBML model object (or any other object with a name). If this is given the displayed text is taken from the name attribute of the referenced object. Obviously exactly one of the two optional attributes (originOfText or text) should be given. If both are given the **text** attribute overrides the **OriginOfText**.

TextGlyph		
graphicalObject	:	SId {use=”optional”}
text	:	string {use=”optional” default=””}
originOfText	:	SId {use=”optional”}

XML Schema representation:

```
<xsd:complexType name="TextGlyph">
 <xsd:complexContent>
  <xsd:extension base="GraphicalObject">
   <xsd:attribute name="graphicalObject" type="SId" use="optional"/>
   <xsd:attribute name="text" type="xsd:string" use="optional"/>
   <xsd:attribute name="originOfText" type="SId" use="optional"/>
  </xsd:extension>
```

```
 </xsd:complexContent>
</xsd:complexType>
```

Further Plans

This document has undergone many changes and we hope that it has advanced enough to be considered for inclusion into the next version of the SBML specification. This specification was intended for the use with sbml level 2. With some slight changes it can also be used with sbml level 1 documents. Actually the only major changes that would have to be made is to change all references to SId to be references to SNames.

Example File

Last but not least, we include a small sample file to illustrate and complement the paragraphs above. Note that both the picture and the example code were manually modified. There does not exist an actual implementation of this latest version of our proposal yet. The model consists of two reactions. Which are the first reaction of glycolysis where glucose is converted to glucose-6-phosphate (G6P) and the reverse reaction of gluconeogenesis where glucose-6-phosphate is hydrolyzed to glucose. We did not include any coordinates in the third dimension, since we are only working in 2D space so far. As can be seen in the screenshot, the glucose SpeciesReference has two representational objects on screen whereas glucose-6-phosphate only has one. This difference is reflected in the file where the glucose species has two nodes in the listOfNodes whereas G6P only has one. This example shows not all but only the main features of our proposal.

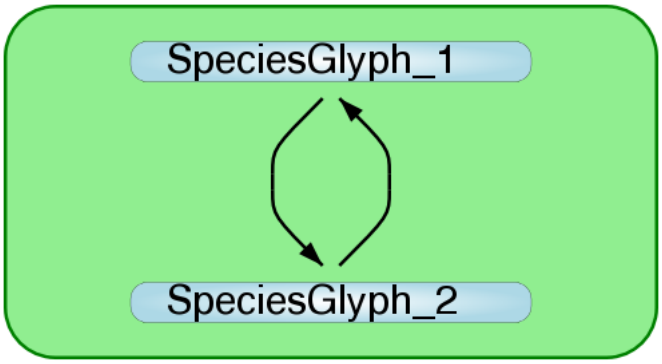

Figure 3: One possible rendering of the example layout.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model id="TestModel">
  <annotation>
  <listOfLayouts xmlns="http://projects.eml.org/bcb/sbml/level2"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <layout id="Layout_1">
      <dimensions width="400" height="220">
      </dimensions>
      <listOfCompartmentGlyphs>
        <compartmentGlyph id="CompartmentGlyph_1" compartment="Compartment_1">
          <boundingBox id="bb1">
            <position x="5" y="5">
            </position>
            <dimensions width="390" height="210">
            </dimensions>
          </boundingBox>
        </compartmentGlyph>
      </listOfCompartmentGlyphs>
      <listOfSpeciesGlyphs>
        <speciesGlyph id="SpeciesGlyph_1" species="Species_1">
          <boundingBox id="bb2">
            <position x="80" y="26">
            </position>
            <dimensions width="240" height="24">
            </dimensions>
          </boundingBox>
        </speciesGlyph>
        <speciesGlyph id="SpeciesGlyph_2" species="Species_2">
          <boundingBox id="bb3">
            <position x="80" y="170">
            </position>
            <dimensions width="240" height="24">
            </dimensions>
          </boundingBox>
        </speciesGlyph>
      </listOfSpeciesGlyphs>
      <listOfReactionGlyphs>
        <reactionGlyph id="ReactionGlyph_1" reaction="Reaction_1">
          <curve>
            <listOfCurveSegments>
              <curveSegment xsi:type="LineSegment">
                <start x="165" y="105">
                </start>
                <end x="165" y="115">
                </end>
              </curveSegment>
            </listOfCurveSegments>
          </curve>
```

```
  <listOfSpeciesReferenceGlyphs>
    <speciesReferenceGlyph id="SpeciesReferenceGlyph_1"
        speciesReference="SpeciesReference_1" speciesGlyph="SpeciesGlyph_1"
        role="1">
      <curve>
        <listOfCurveSegments>
          <curveSegment xsi:type="CubicBezier">
            <start x="165" y="105">
            </start>
            <end x="195" y="60">
            </end>
            <basePoint1 x="165" y="90">
            </basePoint1>
            <basePoint2 x="165" y="90">
            </basePoint2>
          </curveSegment>
        </listOfCurveSegments>
      </curve>
    </speciesReferenceGlyph>
    <speciesReferenceGlyph id="SpeciesReferenceGlyph_2"
          speciesReference="SpeciesReference_2" speciesGlyph="SpeciesGlyph_2"
          role="2">
      <curve>
        <listOfCurveSegments>
          <curveSegment xsi:type="CubicBezier">
            <start x="165" y="115">
            </start>
            <end x="195" y="160">
            </end>
            <basePoint1 x="165" y="130">
            </basePoint1>
            <basePoint2 x="165" y="130">
            </basePoint2>
          </curveSegment>
        </listOfCurveSegments>
      </curve>
    </speciesReferenceGlyph>
  </listOfSpeciesReferenceGlyphs>
</reactionGlyph>
<reactionGlyph id="ReactionGlyph_1" reaction="Reaction_2">
  <curve>
    <listOfCurveSegments>
      <curveSegment xsi:type="LineSegment">
        <start x="235" y="105">
        </start>
        <end x="235" y="115">
        </end>
      </curveSegment>
    </listOfCurveSegments>
```

```
    </curve>
    <listOfSpeciesReferenceGlyphs>
      <speciesReferenceGlyph id="SpeciesReferenceGlyph_3"
             speciesReference="SpeciesReference_3" speciesGlyph="SpeciesGlyph_2"
             role="1">
        <curve>
          <listOfCurveSegments>
            <curveSegment xsi:type="CubicBezier">
              <start x="235" y="115">
              </start>
              <end x="205" y="160">
              </end>
              <basePoint1 x="235" y="130">
              </basePoint1>
              <basePoint2 x="235" y="130">
              </basePoint2>
            </curveSegment>
          </listOfCurveSegments>
        </curve>
      </speciesReferenceGlyph>
      <speciesReferenceGlyph id="SpeciesReferenceGlyph_4"
             speciesReference="SpeciesReference_4" speciesGlyph="SpeciesGlyph_1"
             role="2">
        <curve>
          <listOfCurveSegments>
            <curveSegment xsi:type="CubicBezier">
              <start x="235" y="105">
              </start>
              <end x="205" y="60">
              </end>
              <basePoint1 x="235" y="90">
              </basePoint1>
              <basePoint2 x="235" y="90">
              </basePoint2>
            </curveSegment>
          </listOfCurveSegments>
        </curve>
      </speciesReferenceGlyph>
    </listOfSpeciesReferenceGlyphs>
  </reactionGlyph>
</listOfReactionGlyphs>
<listOfTextGlyphs>
  <textGlyph id="TextGlyph_01" graphicalObject="SpeciesGlyph_1"
             originOfText="SpeciesGlyph_1">
    <boundingBox id="bbA">
      <position x="92" y="26">
      </position>
      <dimensions width="228" height="24">
      </dimensions>
```

```
        </boundingBox>
      </textGlyph>
      <textGlyph id="TextGlyph_02" graphicalObject="SpeciesGlyph_2"
                 originOfText="SpeciesGlyph_2">
        <boundingBox id="bbB">
          <position x="92" y="170">
          </position>
          <dimensions width="228" height="24">
          </dimensions>
        </boundingBox>
      </textGlyph>
    </listOfTextGlyphs>
  </layout>
</listOfLayouts>
</annotation>
  <listOfCompartments>
    <compartment id="Compartment_1"/>
  </listOfCompartments>
  <listOfSpecies>
    <species id="Species_1" compartment="Compartment_1"/>
    <species id="Species_2" compartment="Compartment_1"/>
  </listOfSpecies>
  <listOfReactions>
    <reaction id="Reaction_1" reversible="false">
      <listOfReactants>
        <speciesReference species="Species_1">
          <annotation>
            <layoutId xmlns="http://projects.eml.org/bcb/sbml/level2"
                      id="SpeciesReference_1"/>
          </annotation>
        </speciesReference>
      </listOfReactants>
      <listOfProducts>
        <speciesReference species="Species_2">
          <annotation>
            <layoutId xmlns="http://projects.eml.org/bcb/sbml/level2"
                      id="SpeciesReference_2"/>
          </annotation>
        </speciesReference>
      </listOfProducts>
    </reaction>
    <reaction id="Reaction_2" reversible="false">
      <listOfReactants>
        <speciesReference species="Species_2">
          <annotation>
            <layoutId xmlns="http://projects.eml.org/bcb/sbml/level2"
                      id="SpeciesReference_3"/>
          </annotation>
        </speciesReference>
```

```
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="Species_1">
            <annotation>
              <layoutId xmlns="http://projects.eml.org/bcb/sbml/level2"
                        id="SpeciesReference_4"/>
            </annotation>
          </speciesReference>
        </listOfProducts>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```

Contact Information

To contact any of the authors, send an email to:
FIRSTNAME.LASTNAME@eml-r.villa-bosch.de
e.g.
ralph.gauges@eml-r.villa-bosch.de

Todo

- We need many more examples with nice pictures to go with them. This probably has to wait until the implementation is finished.
- Implement this proposal on top of libsbml with wrappers for Java, Python and C++

List Of Changes

Version 2.1.2

- replaced the sample file and the figure for it

Version 2.1.1

- Layout information moved from annotation of the sbml tag to annotation of the model tag.
- Dimensions can only have positive sizes. The standard interpretation for the bounding box for a reaction glyph was removed. This means

that it is no longer possible to guess reaction arrow layout from the direction of the bounding box of a reaction. Now if there is to be a reaction arrow, a SpeciesReferenceGlyph for that arrow has to be specified.

- Due to dimensions being only positive, there was no way to specify how a reaction glyph should be drawn. Therefor, we added an optional curve attribute to ReactionGlyph. Since this is the same attribute as in the SpeciesReferenceGlyph, it should not pose to much additional work for an implementation.
- Changed all examples to show layout information before model information due to the fact that the annotation has to come at the beginning of an element.
- Some cosmetics concerning the UML diagrams and some rephrasing to make certain paragraphs more clear.

Version 2.1

- Added a new section about the coordinate system.
- Added new Point, Dimensions and BoundingBox objects in the section of GraphicalObject
- Changed GraphicalObject to use a BoundingBox to specify the position and the size.
- Changed curve definitions to take advantage of the new point object.
- Layout objects are now specified without sl2 tags. Just the listOfLayouts tag get the namespace attribute. (This is the way it is done for MathML.)
- Changed the Layout object to use the new Dimensions element to specify the size of the layout instead of attributes.

Version 2.0

- This version has been simplified by removing groups and the render part completely (as discussed in St. Louis).
- A TextGlyph has been introduced.
- Updates on the documentation

Version 1.2

- Group class has been renamed to LayoutGroup to avoid future conflict with the abstract Group class of the render part
- This documents contains some new attributes to objects that are needed to connect this layout information part to the actual render information. All Glyph objects are now derived from GraphicalObject. GraphicalObject has the information for a bounding box and the reference to the id of the connected render object. A transformation to transform the render object prior to rendering can be given as well. GraphicalObject is also used in the listOfAdditionalGraphicalObjects to specify objects to include in the layout that have nothing to do with a chemical model. It could for example be used to draw some legend.
- We took the Transformation, SimpleTransformation and AffineTransformation from the render document and included them here as well since they are needed for GraphicalObject (see last point.)
- Since ReactionGlyph is now derived from GraphicalObject, the two points specified now have the notation of a bounding box and they can be seen as such. On the other hand, programs can still interpret them as two pseudo nodes as was the default so far. Maybe there should be a tag that specifies which it is.
- SpeciesReferenceGlyph, derived from GraphicalObject now includes all the attributes to specify a bounding box. Additionally it can hold a Curve object which overrides the information on the bounding box.

Version 1.1b

- Document now states that the default unit for the diagram is pt.
- Layout object gets three attributes width, height and depth.

Version 1.1a

- Curve Segments can now hold 3D information.

Version 1.1

- Added new edge information to the speciesReferenceGlyph. Edges can now be build from lists of straight lines and cubic bezier segments.

- Added the possibility of grouping of several graphical representations. These groups can then be used in the graphical representation instead of a SpeciesReferenceGlyph.
- Changed all tag names ending with GR(s) to to names ending with Glyph(s)
- Dropped refRole attribute from SpeciesReferenceGlyph in favor of an id in the annotations of the SpeciesReference.
- Changed refSpeciesGlyph attribute to ref so it is more consistent with the rest of the layout objects.
- Fixed the screenshot and corresponding example to correct the error in the pathway. Sample included does still not fully represent the screenshot with this new specification.
- speciesReferences can now be referenced by the speciesReference attribute of the speciesReferenceGlyph. The species reference id that is needed for this is added into the annotations tag of the speciesReference tag in the model.
- The naming of references is now more sbml like since the attributes are called after the object they reference.

References

[1] System Biology Markup Language Level 1 Website (*http://www.sbml.org/sbml/docs/index.html*)

[2] Michael Hucka, Andrew Finney, Herbert Sauro, Hamid Bolouri: Systems Biology Markup Language (SBML) Level 1: Structures and Facilities for Basic Model Definitions (*http://www.sbml.org/sbml/docs/papers/sbml-level-1/html/sbml-level-1.html*)

[3] Hucka M., Finney A., Sauro H.M., Bolouri H., Doyle J.C., Kitano H., Arkin A.P., Bornstein B.J., Bray D., Cornish-Bowden A., Cuellar A.A., Dronov S., Gilles E.D., Ginkel M., Gor V., Goryanin I.I., Hedley W.J., Hodgman T.C., Hofmeyr J.H., Hunter P.J., Juty N.S., Kasberger J.L., Kremling A., Kummer U., Le Novere N., Loew L.M., Lucio D., Mendes P., Minch E., Mjolsness E.D., Nakayama Y., Nelson M.R., Nielsen P.F., Sakurada T., Schaff J.C., Shapiro B.E., Shimizu T.S., Spence H.D.,

Stelling J., Takahashi K., Tomita M., Wagner J., Wang J. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, *Bioinformatics*, **19**, 524-31.

[4] JDesigner Website (*http://www.cds.caltech.edu/ hsauro/JDesigner.htm*)

[5] Herbert Sauro: JDesigner SBML Annotation (*http://www.cds.caltech.edu/ hsauro/JDSBMLEx.pdf*)