SBML Level 3 Package Specification

SBML Level 3 Package: Flux Balance Constraints (‘fbc’)

Brett G. Olivier
b.g.olivier@vu.nl
Systems Bioinformatics
VU University Amsterdam
Amsterdam, NH, The Netherlands

Frank T. Bergmann
fbergmann@caltech.edu
Computing and Mathematical Sciences
California Institute of Technology
Pasadena, CA, US

Version 3, Release 1, Proposal 1μ

September 27, 2018

Contents

1 HARMONY 2018 proposal

The contents of this section are a combination of discussion on the FBC PWG mailing list and discussions at HARMONY 2018 held in Oxford.

The proposed Flux Balance Constraints package version 3 extends the definition of the **FluxObjective**, extends the definition of the `chemicalFormula`, defines a **UserConstraint** and adds a generic **KeyValuePair** annotation.

1.1 The extended Model class

The **SBML Model** class is extended by a `listOfUserConstraints` of which it may contain at most one.

1.1.1 Type `FbcVariableType`

The Flux Balance Constraints package defines a new enumerated type `FbcVariableType` which represents the index of a variable that occurs in either the **FluxObjective** or **UserConstraintComponent**. It contains the following two values, "`linear`" or "`quadratic`".

1.1.2 The FBC listOfUserConstraints

As shown in Figure 2 the **ListOfUserConstraints** is derived from ***SBase*** and inherits the attributes `metaid` and `sboTerm`, as well as the subcomponents for **Annotation** and **Notes**. The **ListOfUserConstraints** must contain at least one **UserConstraint** (defined in Section 1.4).

1.2 The extended Species class

The `chemicalFormula` *attribute*

The optional attribute `chemicalFormula` containing a `string` that represents the **Species** objects elemental composition.

```
<species metaid="meta_M_atp_c" id="M_atp_c" name="ATP" compartment="Cytosol"
 boundaryCondition="false" initialConcentration="0" hasOnlySubstanceUnits="false"
 fbc:charge="-4" fbc:chemicalFormula="C10H12N5O13P3"/>

<species metaid="meta_M1" id="M1" compartment="C" boundaryCondition="false"
 initialConcentration="0" hasOnlySubstanceUnits="false" fbc:charge="0"
 fbc:chemicalFormula="RCONH2"/>

<species metaid="meta_M2" id="M2" compartment="c" boundaryCondition="false"
 initialConcentration="0" hasOnlySubstanceUnits="false" fbc:charge="0"
 fbc:chemicalFormula="C2H4O2(CH2)n"/>
```

While there are many ways of referring to an elemental composition, the purpose of the `chemicalFormula` attribute is to enable reaction balancing and validation, something of particular importance in constraint-based models.

The format of the `chemicalFormula` should, whenever possible, consist only of atomic names (as in the Periodic Table). Similarly, for enhanced inter-operability, the element order should be arranged according to the Hill system (**??**). Using this notation the number of carbon atoms in a molecule is indicated first, followed by the number of hydrogen atoms and then the number of all other chemical elements in alphabetical order. When the formula contains no carbon; all elements, including hydrogen, are listed alphabetically. Where there is more than a single atom present, this is indicated with an integer that follows the element symbol.

However, in certain situations it does become necessary to use a generic symbol in a user defined compound. For

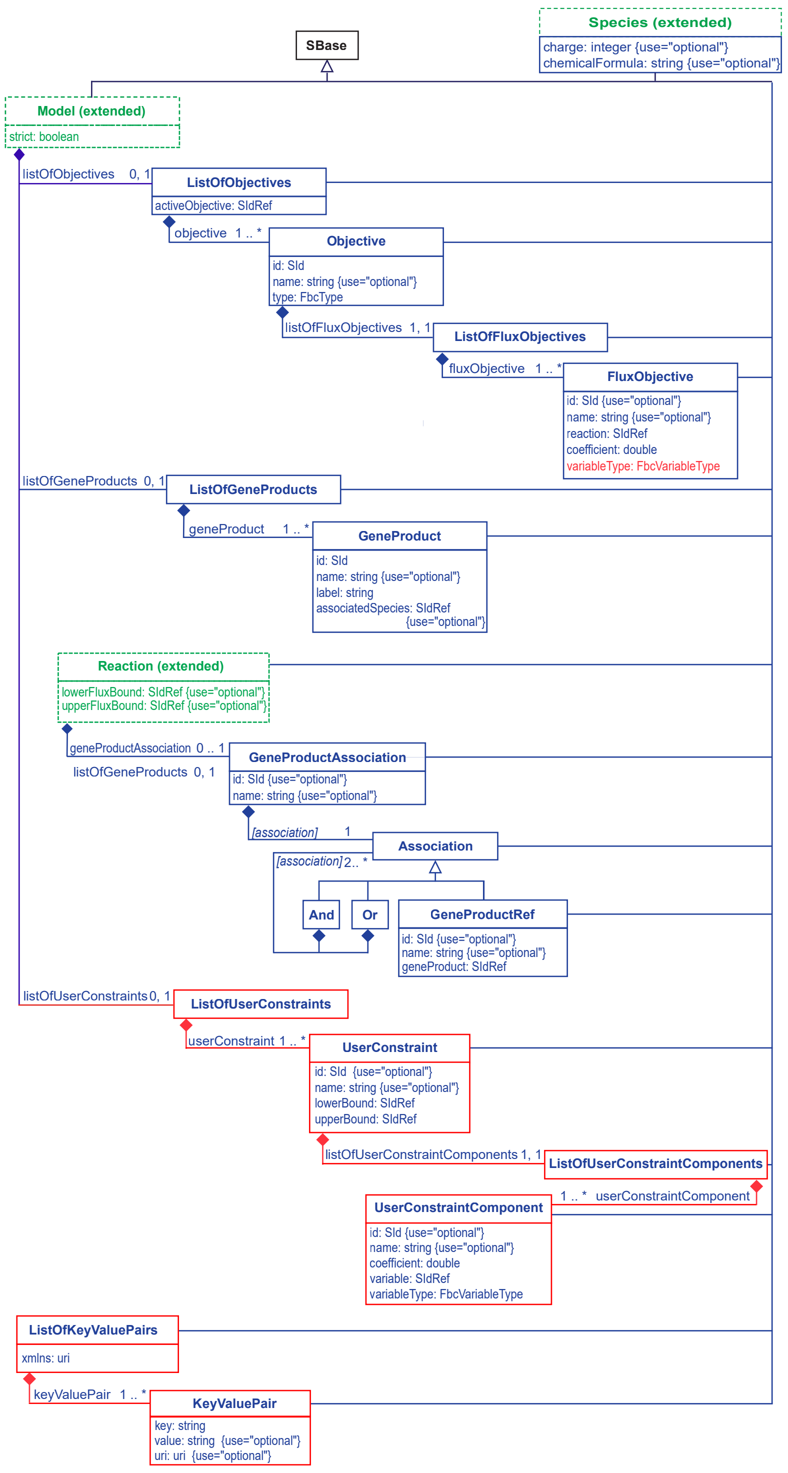

Figure 1: *A UML representation of the Flux Balance Constraints package version three. Derived from* ***SBase****, mostFBC classes inherit support for constructs such as SBML* ***Notes*** *and* ***Annotation****'s. The [association] element name is the name of the class, de-capitalized. In this case, the possible values are "and", "or", or "geneProductRef". See* **??** *for conventions related to this figure. The individual classes are further discussed in the text.*

example, such symbols can include R and X and have the general form of a single capital letter followed by zero or more lowercase letters. In addition, the undefined parenthesised group index $(\ldots)_n$ may also be used. Note that

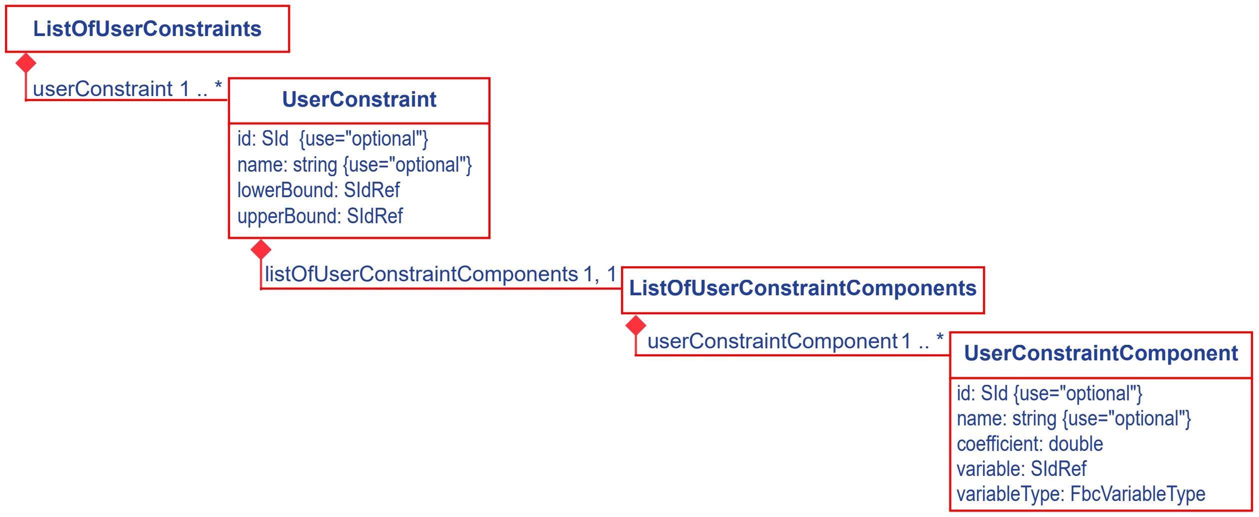

Figure 2: *A UML representation of the **SBML Model** class extended in the Flux Balance Constraints package by the **ListOfUserConstraints**. See **??** for conventions related to this figure.*

H_2O_4S	C_2H_5Br	BrH
$C_{10}H_{12}N_5O_{13}P_3$	CH_3I	CH_4

Table 1: *Examples of chemical formulas written using the Hill System. As described in Section 1.2*

$RCONH_2$	$RCOX$	$C_2H_4O_2(CH_2)_n$

Table 2: *Examples of chemical formulas written using allowed non-Hill symbols, as described in Section 1.2.*

in this case only the subscript n is allowed, integer values $(\ldots)_2$ and expressions such as $(\ldots)_{n-1}$ are considered invalid.

However, the use of R, X and $(\ldots)_n$ is not advised, as any **Reaction** in which such a **Species** occurs cannot necessarily be balanced. Therefore, any `chemicalFormula` that contains any of the aforementioned, non-Hill compatible symbols will raise a 'best practices' warning on model validation.

1.3 The FBC FluxObjective class

The FBC **FluxObjective** class is derived from **SBML *SBase*** and inherits `metaid` and `sboTerm`, as well as the subcomponents for **Annotation** and **Notes**. The **FluxObjective** class is a relatively simple container for a model variable that can be expressed as a 'linear' or 'quadratic', weighted by a signed linear coefficient.

The `id` *and* `name` *attributes*

A **FluxObjective** has two optional attributes: `id` an attribute of type `SId` and `name` an attribute of type `string`.

The `reaction` *and* `coefficient` *attributes*

The required `reaction` is of type `SIdRef` and is restricted to refer only to a **Reaction** while the `coefficient` attribute holds a `double` referring to the coefficient that this **FluxObjective** takes in the enclosing **Objective**.

The `variableType` *attribute*

The required `variableType` attribute contains a `FbcVariableType` that represents the index to which a variable is raised in a **FluxObjective**. For example, where J_x represents a steady-state flux the `FbcVariableType` defines

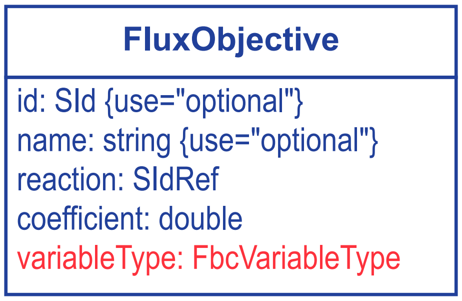

Figure 3: A UML representation of the Flux Balance Constraints package ***FluxObjective*** *class. For a complete description see* **??** *as well as* **??** *for conventions related to this figure.*

either a "`linear`", J_x^1 or "`quadratic`", J_x^2 term.

Flux objectives: example code

An objective with purely linear terms in LP format: `Maximize: 1 R1 + 2 R2`

```
<fbc:listOfObjectives fbc:activeObjective="obj1">
 <fbc:objective fbc:id="obj1" fbc:type="maximize">
  <fbc:listOfFluxObjectives>
   <fbc:fluxObjective fbc:reaction="R1" fbc:coefficient="1" fbc:variableType="linear"/>
   <fbc:fluxObjective fbc:reaction="R2" fbc:coefficient="2" fbc:variableType="linear"/>
  </fbc:listOfFluxObjectives>
 </fbc:objective>
</fbc:listOfObjectives>
```

Similarly, an objective with a quadratic term in LP format: `Minimize: 1 R1 + [4 R2^2]/2`

```
<fbc:listOfObjectives fbc:activeObjective="obj2">
 <fbc:objective fbc:id="obj2" fbc:type="minimize">
  <fbc:listOfFluxObjectives>
   <fbc:fluxObjective fbc:reaction="R1" fbc:coefficient="1" fbc:variableType="linear"/>
   <fbc:fluxObjective fbc:reaction="R2" fbc:coefficient="2" fbc:variableType="quadratic"/>
  </fbc:listOfFluxObjectives>
 </fbc:objective>
</fbc:listOfObjectives>
```

Units

As described above the linear **FluxObjective** defined here as $n \cdot J$ where the `coefficient` (n) is dimensionless and the `value` (J) takes the units of the `reaction` flux i.e., "extent per time". Therefore, the linear **FluxObjective** ($n \cdot J$) has the unit "extent per time" where the units of reaction "extent" and "time" are defined globally. In the case of a quadratic objective $n \cdot J^2$ this would be "extent per time squared."

1.4 The FBC UserConstraint class

The FBC **UserConstraint** class is derived from **SBML** ***SBase*** and inherits `metaid` and `sboTerm`, as well as the sub-components for **Annotation** and **Notes**. It's purpose is to define non-stoichiometric constraints, that is constraints that are not necessarily defined by the stoichiometrically coupled reaction network. In order to achieve we defined a new type of linear constraint, the **UserConstraint**.

Analogous to the attributes described in **??** the `lowerBound` and `upperBound` form the boundaries of the **UserCon-**

straint.

$$\begin{aligned}\textbf{UserConstraint} &= \texttt{value} & (1)\\ \textbf{UserConstraint} &\geq \texttt{lowerFluxBound value} & (2)\\ \textbf{UserConstraint} &\leq \texttt{upperFluxBound value} & (3)\end{aligned}$$

Defining either an equality if both lowerBound and upperBound refere to the same parameter (Equation 4) or set of inequalities (Equations 4 and 5).

The **UserConstraint** contains a **ListOfUserConstraintComponents** representing a linear combination of **UserConstraintComponent**s. Similar to a **FluxObjective** each **UserConstraintComponent** contains a coefficient–variable pair where the coefficient refers to a **Parameter**. In addition to a **Reaction** a **UserConstraintComponent** allows the variable to refer to non-constant **Parameter** thus allowing the definition of non-reaction, artificial, variables.

The id *and* name *attributes*

A **UserConstraint** has an optional id of type SId and an optional attribute name of type string.

The lowerBound *attribute*

The required lowerBound attribute contains an SIdRef that references a **Parameter** which contains the lower boundary value of the **UserConstraint**.

The upperBound *attribute*

The required upperBound attribute contains an SIdRef that references a **Parameter** which contains the upper boundary value of the **UserConstraint**.

The listOfUserConstraintComponents *element*

The element listOfUserConstraintComponents which contains a **ListOfUserConstraintComponents** is derived from and functions like a typical **SBML ListOf___** class with the restriction that it must contain one or more elements of type **UserConstraintComponent** (see Section 1.5). This implies that if a **UserConstraint** is defined there should be at least one **UserConstraintComponent** contained in a **ListOfUserConstraintComponents**.

1.5 The FBC UserConstraintComponent class

The FBC **UserConstraintComponent** class is derived from **SBML *SBase*** and inherits metaid and sboTerm, as well as the subcomponents for **Annotation** and **Notes**. The **UserConstraintComponent** class is a relatively simple container for a variable and a variable type specifier which is weighted by a signed coefficient.

The id *and* name *attributes*

An **UserConstraintComponent** has an optional id of type SId and an optional attribute name of type string.

The coefficient *attribute*

The required coefficient attribute contains an SIdRef that is restricted to reference only a constant **Parameter** which holds the coefficient value. (In **strict** mode a **Parameter** whose SId is referenced by a coefficient, as in the case of a **FluxObjective** coefficient, has to be constant and not take the value NaN or ±inf).

The variable *attribute*

The required variable attribute contains an SIdRef that is restricted to reference the SId of either a **Reaction** or a non-constant **Parameter**. Conversely, if such non-constant **Parameter**'s SId is referenced by a **UserConstraintComponent**'s variable attribute it may not be referenced by any coefficient, lowerFluxBound or upperFluxBound attribute.

The `variableType` *attribute*

The required `variableType` attribute contains a `FbcVariableType` that indicates whether a variable should be considered as 'linear' or 'quadratic'.

User constraints: example code

The following example illustrates the encoding of the following two **UserConstraint**s:

$$RGLX - RXLG = 5 \quad (4)$$
$$2 \cdot Avar - RGDP \geq 2 \quad (5)$$

```
<listOfParameters>
 <parameter id="uc1" value="5" constant="True"/>
 <parameter id="uc2lb" value="2" constant="True"/>
 <parameter id="uc2ub" value="INF" constant="True"/>
 <parameter id="ucco1a" value="1" constant="True"/>
 <parameter id="ucco1b" value="-1" constant="True"/>
 <parameter id="ucco2a" value="2" constant="True"/>
 <parameter id="ucco2b" value="-1" constant="True"/>
 <parameter id="Avar" value="NaN" constant="False"/>
</listOfParameters>

<fbc:listOfUserConstraints>
 <fbc:userContraint fbc:id="uc1" fbc:lowerBound="uc1" fbc:upperBound="uc1">
  <fbc:listOfUserConstraintComponents>
   <fbc:userConstraintComponent fbc:coefficient="ucco1a" fbc:variable="RGLX"
    variableType="linear"/>
   <fbc:userConstraintComponent fbc:coefficient="ucco1b" fbc:variable="RXLG"
    variableType="linear"/>
  </fbc:listOfUserConstraintComponents>
 </fbc:userContraint>
 <fbc:userContraint fbc:id="uc2" fbc:lowerBound="uc2lb" fbc:upperBound="uc2ub">
  <fbc:listOfUserConstraintComponents>
   <fbc:userConstraintComponent fbc:coefficient="ucco2a" fbc:variable="Avar"
    variableType="linear"/>
   <fbc:userConstraintComponent fbc:coefficient="ucco2b" fbc:variable="RGLX"
    variableType="linear"/>
  </fbc:listOfUserConstraintComponents>
 </fbc:userContraint>
</fbc:listOfUserConstraints>
```

1.6 The FBC ListOfKeyValuePairs class

The **ListOfKeyValuePairs**, see Figure 4 for details, forms the basis of a controlled annotation defined by the Flux Balance Constraints package. This element defines a 'structured note' or 'descriptive list' of keys and associated values.

```
 <annotation>
  <listOfKeyValuePairs xmlns="http://sbml.org/fbc/keyvaluepair">
   <keyValuePair key="keyX" url="http://bgoli.net/kvp/spec_example.html" value="47"/>
   <keyValuePair key="ZZkey" url="urn:sbml:fbc:kvp:html:spec_example" value="level␣5"/>
   <keyValuePair key="x-factor" url="http://bgoli.net/kvp/spec_example.html"
                                value="intangible␣metaphysical␣property"/>
  </listOfKeyValuePairs>
 </annotation>
```

As such it is analogous to the official **SBML** RDF annotation used to support MIRIAM annotations, as defined in the **SBML** specification documents. When an annotation that declares the `xmlns` `http://sbml.org/fbc/keyvaluepair` then it must have the format specified here. Tools may chose to support reading and interpreting the content as described, but may optionally ignore the annotation and merely round trip it with any other third party annotations. As is the case with the RDF/MIRIAM annotations, support for **ListOfKeyValuePairs** will be included in the **SBML** support libraries. The official Flux Balance Constraints package annotation, as shown in Figure 4, the **ListOfKeyValuePairs** functions like a typical **SBML ListOf___** class with the restriction that it must contain one or more elements of type **KeyValuePair** (see Section 1.7). In addition it defines a single mandatory attribute, `xmlns`, which identifies the annotation as belonging to the Flux Balance Constraints package.

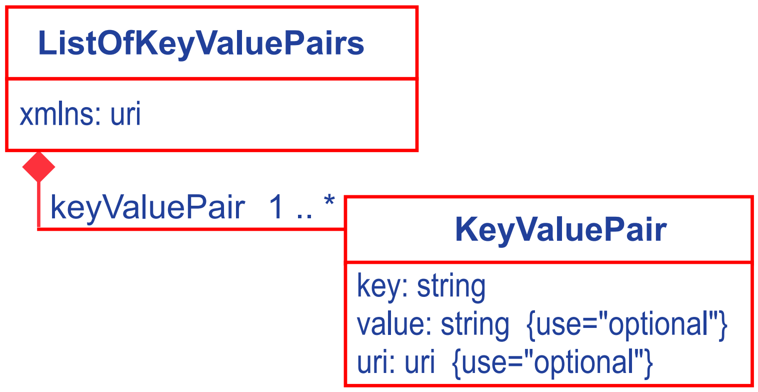

Figure 4: *A UML representation of the **SBML SBase** class extended in the Flux Balance Constraints package by the **ListOfKeyValuePairs**. See **??** for conventions related to this figure.*

The `xmlns` *attribute*

The `xmlns` is a mandatory component of the **ListOfKeyValuePairs**, is of the type `uri` and must have the value `http://sbml.org/fbc/keyvaluepair`.

1.7 The FBC KeyValuePair class

The FBC **KeyValuePair** class is derived from ***SBase*** and inherits the attribute `metaid`, `sboTerm` as well as the subcomponents needed for **Notes** . It's sole purpose is to define a key–value pair with an extended key definition.

I've purposely left out Annotation, BGO.

The **KeyValuePair** defines a single mandatory attribute the `key` as well as two optional attributes: `value` and `uri`.

The `id` *and* `name` *attributes*

A **KeyValuePair** has two optional attributes: `id` an attribute of type `SId` and `name`, an attribute of type `string`.

The `key` *attribute*

The `key` is the mandatory component of the **KeyValuePair** pair and is of type `string`. It has the special property that every `key` in an enclosing **ListOfKeyValuePairs** must be unique.

The `value` *attribute*

The optional `value` attribute is of `string` and contains the value associated with a particular `key`. If not present, the **KeyValuePair** is defined to have no value.

The `uri` *attribute*

The optional attribute `uri` is of type `uri`. This attribute is included and references a resource that contains a description of the `key` component of the **KeyValuePair**. Note that the nature or structure of this resource is not defined. For example, it could be an HTML page with a textual description of the `key`(s) used by a particular tool or another that contains a table of key definitions[1].

However, the concomitant development of a central `key` resource and associated standardised `key` definition would enable greater model reusability and tool interoperability and be of benefit to the broader **SBML** community.

[1] `http://bgoli.net/kvp/spec_example.html` (Temporary, will change.)