4 Proposal for FBC version 3: HARMONY 2018

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

The extensions and modifications contained in this document are proposed as version 3 of the Flux Balance Constraints package. This proposal is been formulated through open discussions on the FBC PWG mailing list and intense discussion during HARMONY 2018.

Please note that where specification text (or UML) has been modified and diverges from the Flux Balance Constraints package version 2 specification, it has been colored red.

4.1 The extended Model class (updates specification Section 3.3)

The SBML Model class is extended by a listOfUserConstraints of which it may contain at most one, see Figure 13.

4.1.1 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”.

4.1.2 The FBC listOfUserConstraints

As shown in Figure 13 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 4.4).
Figure 10: 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 Section 1.4 for conventions related to this figure. The individual classes are further discussed in the text.
4.2 The extended Species class (updates specification Section 3.4)

The Flux Balance Constraints package extends the SBML Level 3 Version 1 Core Species class with the addition of two attributes charge and chemicalFormula.

![Species (extended)](charge: double {use="optional"}
chemicalFormula: string {use="optional"})

Figure 11: A UML representation of the extended SBML Species class used in the Flux Balance Constraints package. See Section 1.4 for conventions related to this figure.

The charge attribute

The optional attribute charge contains a signed double referring to the Species object’s charge (in terms of electrons, not the SI unit coulombs). Note, that unlike FBC versions one and two a Species may, for the purposes of charge, be interpreted as a pseudoisomer or aggregate molecule and may assume a non-integer value. Non-integer charges should be used with caution as their use may have unintended side-effects, for example, with respect to the accuracy of reaction balancing.

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 (see Table 2) (Hill, 1900, 2012). Using this notation the number of carbon atoms in a molecule is indicated first,

\[
\begin{align*}
\text{H}_2\text{O}_4\text{S} & \quad \text{C}_2\text{H}_5\text{Br} & \quad \text{BrH} \\
\text{C}_{10}\text{H}_{12}\text{N}_5\text{O}_{13}\text{P}_3 & \quad \text{CH}_3\text{I} & \quad \text{CH}_4
\end{align*}
\]

Table 2: Examples of chemical formulas written using the Hill System. As described in Section 4.2

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 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

\[
RCONH_2 \quad RCOX \quad C_2H_4O_2(CH_2)_n
\]

Table 3: Examples of chemical formulas written using allowed non-Hill symbols, as described in Section 4.2.

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 \texttt{Reaction} in which such a \texttt{Species} occurs cannot necessarily be balanced. Therefore, any \texttt{chemicalFormula} that contains any of the aforementioned, non-Hill compatible symbols will raise a ‘best practices’ warning on model validation.

### 4.3 The FBC FluxObjective class

The FBC \texttt{FluxObjective} class is derived from \texttt{SBML SBase} and inherits \texttt{metaid} and \texttt{sboTerm}, as well as the subcomponents for \texttt{Annotation} and \texttt{Notes}. The \texttt{FluxObjective} class is a relatively simple container for a model variable that

![FluxObjective](image)

Figure 12: A UML representation of the Flux Balance Constraints package \texttt{FluxObjective} class. For a complete description see Figure 1 as well as Section 1.4 for conventions related to this figure.

can be expressed as a ‘linear’ or ‘quadratic’, weighted by a signed linear coefficient.

**The id and name attributes**

A \texttt{FluxObjective} has two optional attributes: \texttt{id} an attribute of type \texttt{SId} and \texttt{name} an attribute of type \texttt{string}.

**The reaction and coefficient attributes**

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

**The variableType attribute**

The required \texttt{variableType} attribute contains a \texttt{FbcVariableType} that represents the index to which a variable is raised in a \texttt{FluxObjective}. For example, where \( J_x \) represents a steady-state flux the \texttt{FbcVariableType} defines 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: \texttt{Maximize: 1 R1 + 2 R2}
Similarly, an objective with a quadratic term in LP format: Minimize: \( 1 \cdot R_1 + \frac{4 \cdot R_2^2}{2} \)

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 \( \text{extent} / \text{time} \) where the units of reaction “extent” and “time” are defined globally. Analogously, in the case of a quadratic FluxObjective, \( n \cdot J^2 \) this would be \( \text{extent}^2 / \text{time}^2 \).

4.4 The FBC UserConstraint class

The FBC UserConstraint class is derived from SBML SBase and inherits metaid and sboTerm, as well as the subcomponents 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.

Figure 13: A UML representation of the SBML Model class extended in the Flux Balance Constraints package by the ListOfUserConstraints. See Section 1.4 for conventions related to this figure.

Analogous to the attributes described in Section 3.8 the lowerBound and upperBound form the boundaries of the
The UserConstraint component class

\begin{align}
\text{UserConstraint} & = \text{value} \quad (4) \\
\text{UserConstraint} & \geq \text{lowerBound \ value} \quad (5) \\
\text{UserConstraint} & \leq \text{upperBound \ value} \quad (6)
\end{align}

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

The UserConstraint contains a ListOfUserConstraintComponents representing a linear combination of UserConstraintComponents. 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 4.5). This implies that if a UserConstraint is defined there should be at least one UserConstraintComponent contained in a ListOfUserConstraintComponents.

### 4.5 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.
Section 4.6 The FBC ListOfKeyValuePairs class

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 `UserConstraints`:

\[
RGLX - RXLG = 5 \tag{7}
\]

\[
2 \cdot Avar - RGDP \geq 2 \tag{8}
\]

```xml
<listOfParameters>
  <parameter id="uc1" value="5" constant="True"/>
  <parameter id="uc2lb" value="2" constant="True"/>
  <parameter id="uc2ub" value="INF" constants="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>
```

```xml
<fbc:listOfUserConstraints>
  <fbc:userConstraint 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:userConstraint>
  <fbc:userConstraint 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:userConstraint>
</fbc:listOfUserConstraints>
```

4.6 The FBC ListOfKeyValuePairs class

The **ListOfKeyValuePairs**, see Figure 14 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.

```xml
<annotation>
  <listOfKeyValuePairs xmlns="http://sbml.org/fbc/keyvaluepair">
    <keyValuePair key="keyX" uri="https://tinyurl.com/ybyr7b62" value="47"/>
    <keyValuePair key="ZZkey" uri="urn:tinyurl.com:example:kvp" value="level␣5"/>
    <keyValuePair key="x-factor" uri="https://tinyurl.com/ybyr7b62" 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 ListOfKeyValuePairs functions like a typical SBML ListOf___ class with the restriction that it must contain one or more elements of type KeyValuePair (see Section 4.7). In addition it defines a single mandatory attribute, xmlns, which identifies the annotation as belonging to the Flux Balance Constraints package.

**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.

### 4.7 The FBC KeyValuePair class

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

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 as having no value.

**The uri attribute**

The optional attribute uri is of type uri. This attribute identifies a resource that defines the associated key component of the KeyValuePair, see Table 4 for examples. As a best practice, it is highly recommended that all tools implementing a KeyValuePair also implement support for the uri attribute.
### Table 4: Examples of how the `uri` attribute can be used to identify key definitions by `urn` or `url`.

<table>
<thead>
<tr>
<th>Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>urn</td>
<td>urn:tinyurl.com:example:kvp</td>
<td>a tool specific namespace declaration</td>
</tr>
<tr>
<td>url</td>
<td><a href="https://tinyurl.com/ybyr7b62">https://tinyurl.com/ybyr7b62</a></td>
<td>a url identifying a set of key definitions</td>
</tr>
</tbody>
</table>