Simulation Experiment Description Markup Language (SED-ML) : Level 1 Version 1 (Draft)

August 11, 2010

Editors

Dagmar Waltemath	*Rostock University, Germany*
Nicolas Le Novère	*European Bioinformatics Institute, UK*
Frank T. Bergmann	*University of Washington, Seattle, USA*

The latest release of the Level 1 Version 1 specification is available at
http://biomodels.net/sed-ml#sedmlResources

To discuss any aspect of the current SED-ML specification as well as language details, please send your messages to the mailing list
sed-ml-discuss@lists.sourceforge.net.

To get subscribed to the mailing list, please write to the same address
sed-ml-discuss@lists.sourceforge.net.

To contact the authors of the SED-ML specification, please write to
sed-ml-editors@lists.sourceforge.net

1 Introduction

As Systems Biology transforms into one of the main fields in life sciences, the number of available computational models is growing at an ever increasing pace. At the same time, their size and complexity are also increasing. The need to build on existing studies by reusing models therefore becomes more imperative. It is now generally accepted that one needs to be able to exchange the biochemical and mathematical structure of models. The efforts to standardise the representation of computational models in various areas of biology, such as the *Systems Biology Markup Language* (SBML, Hucka et al. [2003]), *CellML* Lloyd et al. [2004] or *NeuroML* Goddard et al. [2001], result in an increase of the exchange and re-use of models. However, the description of the structure of models is not sufficient to enable the reproduction of simulation results. One also needs to describe the procedures the models are subjected to, as described by the *Minimum Information About a Simulation Experiment (MIASE)* [Waltemath et al.].

This document presents Level 1 Version 1 of the *Simulation Experiment Description Markup Language* (SED-ML), a format that allows for the encoding of simulation experiments. SED-ML files are encoded in the *eXtensible Markup Language* (XML) [Bray et al., 2006]). The SED-ML language is defined by an XML Schema [Fallside et al., 2001].

1.1 Motivation: A sample experiment

To demonstrate how a simulation experiment can be described simply and effectively, we make use of a rather simple, though famous, model that may yet display rich and variable behaviors. The simulation example is taken from [Waltemath et al.].

The *repressilator* is a synthetic oscillating network of transcription regulators in Escherichia coli [Elowitz and Leibler, 2000]. The network is composed of the three repressor genes Lactose Operon Repressor (lacI), Tetracycline Repressor (tetR) and Repressor CI (cI), which code for proteins binding to the promoter of the other, blocking their transcription. The three inhibitions toghether in tandem, form a cyclic negative-feedback loop. To describe the interactions of the molecular species involved in the network, the authors built a simple mathematical model of coupled first-order differential equations. All six molecular species included in the network (three mRNAs, three repressor proteins) participated in creation (transcription/translation) and degradation processes. The model was used to determine the influence of the various parameters on the dynamic behavior of the system. In particular, parameter values were sought which induce stable oscillations in the concentrations of the system components. Oscillations in the levels of the three repressor proteins are obtained by numerical integration.

1.1.1 A simple time-course simulation

The first experiment we intend to run on the model is the simulation that will lead to the oscillation shown in Figure 1c of the reference publication [Elowitz and Leibler, 2000]. The according simulation experiment can be described as:

1. Import the model identified by the Unified Resource Identifier [Berners-Lee et al., 2005] `urn:miriam:biomodels.db:BIOMD0000000012`.
2. Select a deterministic method.
3. Run a uniform time course simulation for 1000 min with an output interval of 1 min.
4. Plot the amount of `lacI`, `tetR` and `cI` against time in a 2D Plot.

Following those steps and performing the simulation in the simulation tool COPASI [Hoops et al., 2006] let to the result shown in Figure 1 on the following page.

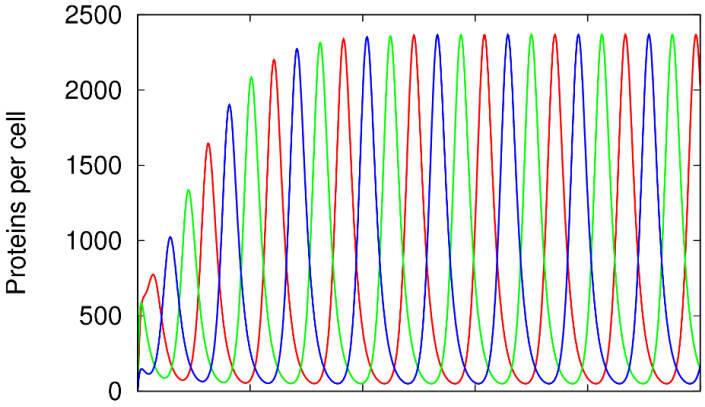

Figure 1: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI. The number of repressor proteins lacI, tetR and cI is shown. (taken from [Waltemath et al.])*

1.1.2 Applying pre-processing

The fine-tuning of the model can be shown by adjusting parameters before simulation. When changing the initial values of the parameters *protein copies per promoter* and *leakiness in protein copies per promoter* the system's behavior switches from sustained oscillation to asymptotic steady-state. The adjustments leading to that behavior may be described as:

1. Import the model as above.
2. Change the value of the parameter `tps_repr` from "0.0005" to "1.3e-05".
3. Change the value of the parameter `tps_active` from "0.5 " to " 0.013".
4. Select a deterministic method.
5. Run a uniform time course for the duration of 1000 min with an output interval of 1 min.
6. Plot the amount of lacI, tetR and cI against time in a 2D Plot.

Figure 2 on the next page shows the result of the simulation.

1.1.3 Applying post-processing

However, the raw numerical output of the simulation steps may be subjected to data post-processing before plotting or reporting. In order to describe the production of a normalized plot of the time-course in the first example (section 1.1.1), depicting the influence of one variable on another (in phase-planes), one could define the following further steps:

(Please note that the description steps 1 - 4 remain as above)

5. Collect PX(t) (lacI), PY(t) (tetR) and PZ(t) (cI).
6. Compute the highest value for each of the repressor proteins, max(PX(t)), max(PY(t)), max(PZ(t)) .
7. Normalize the data for each of the repressor proteins by dividing each time point by the maximum value, i.e., PX(t)/max(PX(t)), PY(t)/max(PY(t)) , and PZ(t)/max(PZ(t)).
8. Plot the normalized `lacI` protein in function of the normalized `cI`, the normalized `cI` in function of the normalized `tetR` protein, and the normalized `tetR` protein against the normalized `lacI` protein in a 2D plot.

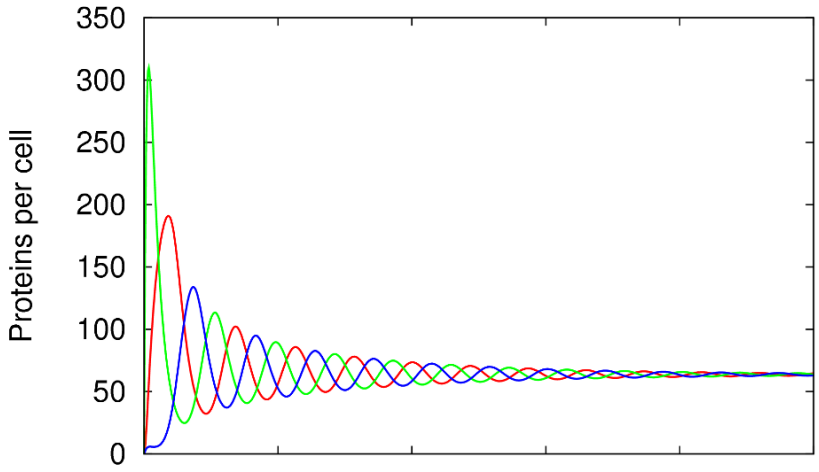

Figure 2: *Time-course simulation of the repressilator model, imported from BioModels Databas and simulated in COPASI after modification of the initial values of the* protein copies per promoter *and the* leakiness in protein copies per promoter. *The number of repressor proteins lacI, tetR and cI is shown. (taken from [Waltemath et al.])*

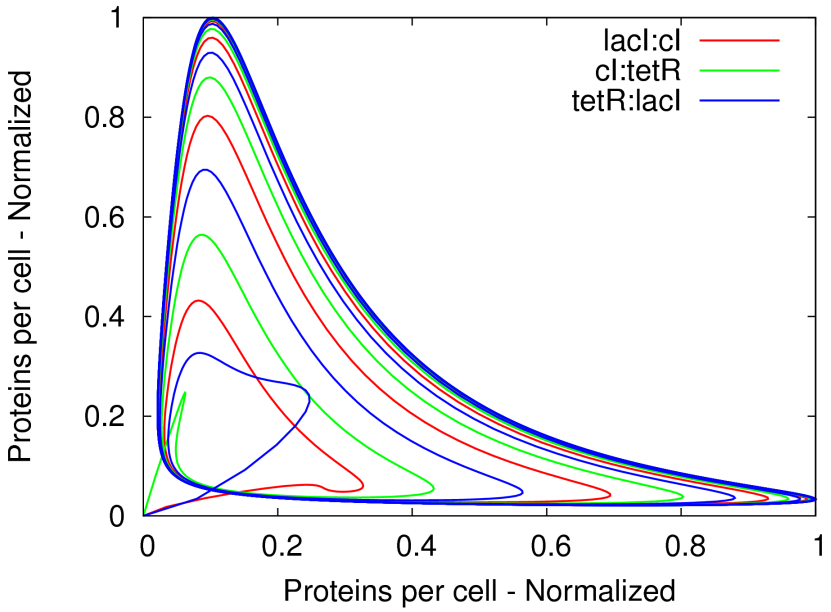

Figure 3: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI, showing the normalized temporal evolution of repressor proteins lacI, tetR and cI in phase-plane. (taken from [Waltemath et al.])*

Figure 3 illustrates the result of the simulation after post-processing of the output data.

1.2 Conventions used in this document

SED-ML is specified as an XML Schema [W3C, 2004]. We also provide a UML Class diagram representation of that XML Schema (refer to appendix A). UML class diagrams are a subset of the *Unified Markup Language* notation (UML, [OMG, 2009]). Sample experiment descriptions are given as XML snippets that comply with the XML Schema.

1.2.1 UML Classes

A SED-ML UML class (Figure 4) consists of a class name (`ClassName`) and a number of attributes (`attribute`) each of a specific data type (`type`). The SED-ML UML specification does not make use of UML `operations`.

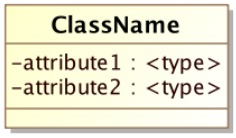

Figure 4: *SED-ML UML Class with class names and attributes*

SED-ML class names always begin with upper case letters, if they are composed of different words, the camel case style is used, as in e. g. `DataGenerator`.

1.2.2 UML Relationships

1.2.2.1 UML Relation Types

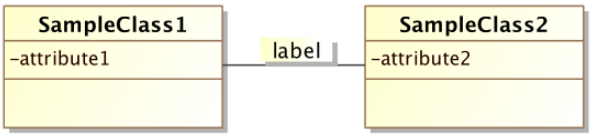

Figure 5: *UML Class connectors*

Links between classes specify the connection of objects with each other (Figure 5). The different relation types used in the SED-ML specification include aggregation, composite aggregation, and generalisation. The label on the line is called symbol (`label`) and describes the relation of the objects of both classes.

The association (Figure 6 on the next page) indicates the existence of a connection between the objects of the participating classes. Often associations are directed to show how the label should be read (in which direction). Associations can be uni-directional (one arrowhead), or bidirectional (zero or two arrowheads).

The aggregation (Figure 7 on the following page, top) indicates that the objects of the participating classes are connected in a way that one class (`Whole`) consists of several parts (`Part`). In an aggregation, the parts may be independent of the whole. For example, a car (`Whole`) has several parts called wheel (`Part`); however, the wheels can exist independently of the car while the car requires the wheels in order to function.

The composite aggregation (Figure 7 on the next page, bottom) indicates that the objects of the participating classes are connected in a way that one class (`Whole`) consists of several parts (`Part`). In contrast to the aggregation, the subelements (`Part`) are dependent on the parent class (`Whole`). An example is that a university (`Whole`) consists of a number of departments (`Part`) which have a so-called "lifetime responsibility" with the university, e. g. if the university vanishes, so will with it the departments Bell [2003].

The generalisation (Figure 8 on page 9) allows to extend classes (`BaseClass`) by additional properties. The derived class (`DerivedClass`) inherites all properties of the base class and defines additional ones.

Figure 6: *UML Association*

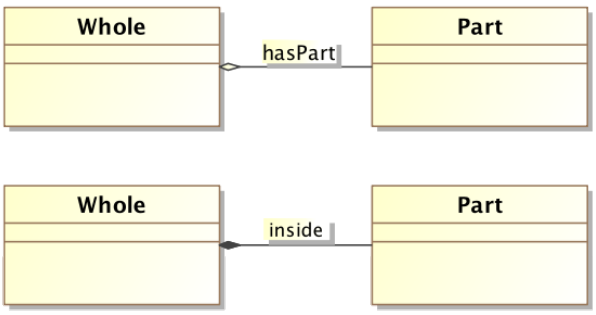

Figure 7: *UML Aggregation*

1.2.2.2 UML multiplicity

UML multiplicity defines the number of objects in one class that can be related to one object in the other class (also known as cardinality). Possible types of multiplicity include values (1), ranges (1..4), intervals (1,3,9), or combinations of ranges and intervals. The standard notation for "many" is the asterix (*).

Multiplicity can be defined for both sides of a relationship between classes. The default relationship is "many to many". The example in Figure 9 on the next page expresses that a class is given by a professor, and a professor might give one to many classes.

1.2.3 XML Schema language elements

The main building blocks of an XML Schema specification are

- simple and complex types
- element specifications
- attribute specifications

XML Schema definitions create new types, declarations define new elements and attributes. The definition of new (simple and complex) types can be based on a number of already existing, prefedined types (string, boolean, float). Simple types are restrictions or extensions of predefined types. Complex types describe how attribues can be assigned to elements and how elements can contain further elements. The SED-ML XML Schema only makes use of *complex type definitions*. An example for a complex type definition is given in listing 1:

```
1 <xs:element name="computeChange">
2  <xs:complexType>
3   <xs:sequence>
4    <xs:element ref="listOfVariables" />
5    <xs:element ref="listOfParameters" />
6    <xs:element ref="math:math" />
7   </xs:sequence>
8   <xs:attribute name="target" type="xs:token" />
9  </xs:complexType>
10 </xs:element>
```

Listing 1: *Complex Type definition of the SED-ML* `computeChange` *element*

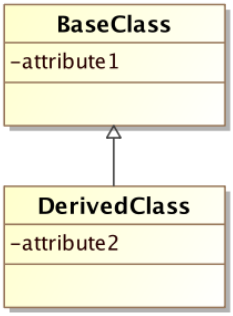

Figure 8: *UML Generalisation*

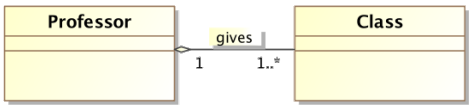

Figure 9: *UML Multiplicity in an Aggregation*

It shows the declaration of an element called `computeChange` that is used in SED-ML to change mathematical expressions. The element is defined using an *unnamed* complex type which is build of further elements called `listOfVariables`, `listOfParameters`, and `math`. Additionally, the element `computeChange` has an attribute `target` declared. Please note that the definition of the elements inside the complex type are only referred to and will be found elsewhere in the schema. The nesting of elements in the schema can be expressed using the `xs:sequence` (a sequence of elements), `xs:choice` (an alternative of elements to choose from), or `xs:all` (a set of elements that can occur in any order) concepts. The SED-ML XML Schema only uses the *sequence* of elements.

1.2.3.1 Multiplicities

The standard multiplicity for each defined `element` is 1. Explicit multiplicity is to be defined using the `minOccurs` and `maxOccurs` attributes inside the complex type definition, as shown in listing 2.

```
<xs:element name=" dataGenerator ">
 <xs:complexType >
  <xs:sequence >
   <xs:element ref=" listOfVariables " minOccurs ="0" />
   <xs:element ref=" listOfParameters " minOccurs ="0" />
   <xs:element ref=" math:math " />
  </xs:sequence >
  [..]
 </xs:complexType >
</xs:element >
```

Listing 2: *Multiplicity for complex types in XML Schema*

The `dataGenerator` type is build of a sequence of three elements: The `listOfVariables` element is not necessary for the definition of a valid `dataGenerator` XML structure (it may occur 0 times or once). The same is true for the `listOfParameters` element (it may as well occur 0 times or once). The `math` element, however, uses the implicit standard multiplicity – it must occur exactly 1 time in the `dataGenerator` specification.

1.2.3.2 Type extensions

XML Schema offers mechanics to restrict and extend previously defined complex types. Extensions add element or attribute declarations to existing types, while restrictions restrict the types by adding further characteristics and requirements (facets) to a type. An example for a

type extension is given in listing 3.

```
<xs:element name="sedML">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="SEDBase">
    <xs:sequence>
     <xs:element ref="listOfSimulations" />
     <xs:element ref="listOfModels" />
     <xs:element ref="listOfTasks" />
     <xs:element ref="listOfDataGenerators" />
     <xs:element ref="listOfOutputs" />
    </xs:sequence>
    <xs:attribute name="version" type="xs:decimal" use="required" fixed="0.1" />
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
```

Listing 3: *Definition of the sedML type through extension of SEDBase in SED-ML*

The `sedML` element is an extension of the previously defined `SEDBase` type. It extends `SEDBase` by a sequence of five additional elements (`listOfSimulations`, `listOfModels`, `listOfTasks`, `listOfDataGenerators`, and `listOfOutputs`) and a new attribute `versions`.

2 Concepts used in SED-ML

2.1 The MathML Subset used in SED-ML

The SED-ML specification allows for the encoding of pre-processing applied to the compuational model, as well as for the encoding of post processing applied to the raw simulation data before output. The corresponding mathematical expressions are encoded using MathML 2.0 [Carlisle et al., 2001]. MathML is an international standard for encoding mathematical expressions using XML. It is also used as representation of mathematical expressions in other formats, such as SBML and CellML, two of the languages supported by SED-ML.

FRANK: I have distinguished operations, functions and symbols now – could you check the whole section again? I am not very familiar with mathML terminology ;-)

2.1.1 MathML operations

In order to make the SED-ML format easier to adopt, at the beginning we restrict the MathML subset to the following operations:

- *token*: `cn`, `ci`, `csymbol`, `sep`
- *general*: `apply`, `piecewise`, `piece`, `otherwise`, `lambda`
- *relational operators*: `eq`, `neq`, `gt`, `lt`, `geq`, `leq`
- *arithmetic operators*: `plus`, `minus`, `times`, `divide`, `power`, `root`, `abs`, `exp`, `ln`, `log`, `floor`, `ceiling`, `factorial`
- *logical operators*: `and`, `or`, `xor`, `not`
- *qualifiers*: `degree`, `bvar`, `logbase`
- *trigonometric operators*: `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arcsec`, `arccsc`, `arccot`, `arcsinh`, `arccosh`, `arctanh`, `arcsech`, `arccsch`, `arccoth`
- *constants*: `true`, `false`, `notanumber`, `pi`, `infinity`, `exponentiale`
- *MathML annotations*: `semantics`, `annotation`, `annotation-xml`

2.1.2 MathML Symbols

All the operations listed above only operate on singular values. However, as one of SED-ML's aim is to provide post processing on the results of simulation experiments, we need to enhance this basic set of operations by some aggregate functions. Therefore a defined set of MathML symbols that represent vector values are supported by SED-ML Level 1 Version 1. To simplify things for SED-ML L1V1 the only symbols to be used are the identifiers of variables defined in the listOfVariables of DataGenerators. These variables represent the data collected from the simulation experiment with the associated task. FRANK/NLN: and the implicit variables as well, right?

2.1.3 MathML functions

The following aggregate functions are available for use in Level 1 Version 1.

- *min*: Where the minimum of a variable represents the smallest value the simulation experiment yielded. Example:

  ```
  <min> <ci> variableId </ci></min>
  ```

- *max*: Where the maximum of a variables represents the largest value the simulation experiment yielded. Example:

  ```
  <max> <ci> variableId </ci></max>
  ```

- *sum*: All values of the variable returned by the simulation experiment are added up. Example:

  ```
  <sum> <ci> variableId </ci></sum>
  ```

- *product*: All values of the variable returned by the simulation experiment are multiplied. Example:

  ```
  <product> <ci> variableId </ci></product>
  ```

These represent the only exceptions. At this point SED-ML does not define a complete algebra of vector values. For more information see the description of the DataGenerator class.

2.2 URI Scheme in SED-ML

URIs are needed at three different points in SED-ML Level 1 Version 1: Firstly, they are the preferred mechanism to refer to model encodings. Secondly, they are used to specify the language of the referenced model. Thirdly, they enable addressing implicit model variables.

The use of a standardised URI Scheme ensures long-time availability of a particular information that can unambiguously be identified.

2.2.1 Model references

The preferred way for referencing a model from a SED-ML file is the MIRIAM URI Scheme. MIRIAM allows to identify a data resource by a predefined URN. A data entry inside that resource is identified by an ID. That way each single model in a particular model repository can be unambiguously referenced. To become part of MIRIAM resources, a model repository must ensure permanent and consistent model references, that is stable IDs.

One model repository that is part of MIRIAM resources is the BioModels Database. It's data resource name in MIRIAM is `urn:miriam:biomodels.db`. To refer to a particular model, a standardised identifier scheme is defined in MIRIAM Resources. The ID entry maps to a particular model in the model repository. That model is never deleted. A sample BioModels Database ID is `BIOMD0000000048`. Together with the data resource name it becomes unambiguously referrable by the URN `urn:miriam:biomodels.db:BIOMD0000000048` (in this case referring to the 1999 Kholodenko model on EGFR signaling).

SED-ML does not specify how to resolve the URNs. However, MIRIAM Resources offers web services to do so [1]. For the above example of the `urn:miriam:biomodels.db:BIOMD0000000048` model, the resolved URL may look like:

- `http://biomodels.caltech.edu/BIOMD0000000048`
- `http://www.ebi.ac.uk/biomodels-main/BIOMD0000000048`

depending on the physical location of the resource chosen to resolve the URN.

Further information on the source attribute referencing the model location is provided in section 4.1.2.

2.2.2 Language references

To specify the language a model is encoded in, a set of pre-defined SED-ML URNs can be used. The structure of SED-ML language URNs is `urn:sedml:language:`*name.version*. SED-ML allows to specify a model representation format very generally as "XML", if no standardised representation format has been used to encode the model. On the other hand, one can be as

[1] http://www.ebi.ac.uk/miriam

specific as defining a model being in a particular version of a language, as "SBML Level 2, Version 2, Revision 1".

The list of URNs is available from `http://www.biomodels.net/sed-ml/#sedmlLanguage`. Further information on the language attribute is provided in section 4.1.1.

2.2.3 *Implicit variables*

Some variables used in an experiment are not explicitely defined in the model, but may be implicitely contained in it. For example, to plot a variable's behaviour over time, that variable is defined in an SBML model, while time is not explicitely defined.

To overcome that shortness and allow SED-ML to refer to such variables in a common way, the notion of *implicit variables* is used. Those variables are called `symbols` in SED-ML. They are defined following the idea of MIRIAM URNs and using the SED-ML URN scheme. The structure of the URNs is `urn:sedml:symbol:`*implicit variable*. To refer from a SED-ML file to the definition of *time*, for example, the URN is `urn:sedml:symbol:time`.

The list of predefined symbols is available from the SED-ML site on `http://biomodels.net/sed-ml`. From that source, also a mapping of SED-ML symbols on possibly existing concepts in the single languages supported by SED-ML is provided.

FRANK: we have to define a symbols section on the web site, anything apart time for the moment?

2.3 KiSAO

An important aspect of a simulation experiment is the simulation algorithm used to solve the system.

The sole reference of a simulation algorithm through its name in form of a string is error prone and unambiguous. Firstly, typing mistakes or language differences may make the identification of the intended algorithm difficult. Secondly, many algorithms exist with more than one name, having synonyms or various abbriviations that are commonly used.

These problems can be solved by using controlled vocabulary to refer to a particular simulation algorithm. One attempt to provide such a vocabulary is the *Kinetic Simulation Algorithm Ontology* (KiSAO, `http://www.ebi.ac.uk/compneur-srv/kisao/`). KiSAO is a community-driven approach of classifying and structuring simulation approaches by model characteristics and numerical characteristics. Model characteristics include, for instance, the type of variables used for the simulation (such as discrete or continuous variables) and the spatial resolution (spatial or non-spatial descriptions). Numerical characteristics specify whether the system's behavior can be described as deterministic or stochastic, and whether the algorithms use fixed or adaptive time steps. Related algorithms are grouped together, producing classes of algorithms [Courtot et al., to be submitted].

Although work is still at an early stage, the use of KiSAO is recommended when referring to a simulation algorithm from a SED-ML description. However, the use of KiSAO for the moment is limited. One may look up the algorithm that was used in the simulation experiment (through resolving the KiSAO ID) and then try and use one algorithm that is as similar to the original one as possible. KiSAO will become more supportive for SED-ML as soon as the ontology contains a wider range of relationships between different algorithms, as well as extended descriptions of the algorithm characteristics.

2.4 SED-ML resources

SED-ML is part of the biomodels.net initiative http://www.biomodels.net. Information on SED-ML can be found on www.biomodels.net/sed-ml.

The SED-ML XML Schema, the UML schema and related implementations, libraries, validators and so on can be found on the SED-ML sourceforge project page http://sed-ml.svn.sourceforge.net/.

3 Preliminary definitions and general attributes and classes

In this section we introduce attributes and concepts used repeatedly throughout the SED-ML specification.

3.1 The `xmlns` attribute

The xmlns attribute is used in several places of the SED-ML document. First of all, it declares the namespace for the SED-ML document. The pre-defined standard namespace is `http://www.biomodels.net/sed-ml`.

In addition, SED-ML makes use of the MathML namespace `http://www.w3.org/1998/Math/MathML` to enable encoding mathematical expressions in MathML 2.0. SED-ML uses a subset of MathML as described in section 2.1 on page 11.

For the Notes class, the standard value of `xmlns` is `http://www.w3.org/1999/xhtml`.

3.2 The `id` attribute

Most objects in SED-ML carry the id and name attributes. The id attribute, if existent for an object, is always required and can be used to identify SED-ML constituents unambiguously. The id attribute can be used to refer to a constituent from other constituents. The `id` data type is `String`. All `id`s have a global scope, meaning that throughout a whole SED-ML document, the `id` should be unambiguous and as such identifying the constituent it is related to. An example for a defined id is given in listing 4.

```
1 <model id="m00001" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.
      db:BIOMD0000000012">
2  [MODEL DEFINITION]
3 </model>
```

Listing 4: *SED-ML identifier definition, e. g. for a model*

3.3 The `name` attribute

Besides an id, a SED-ML constituent may carry an optional name attribute. However, names do not have identifying character and so severeal SED-ML constituents may carry the same name. The purposes of the name attribute is to keep a human-readible name of the constituent, e. g. for display to the user. Names are of the data type `String` in the XML Schema representation.

3.4 The SEDBase Class

SEDBase represents the base class for all elements of the SED-ML Level 1 Version 1 language. That is, all elements are derived from it. It provides means for additional information to be attached on all other classes (Figure 10 on the following page). That information can be specified in form of human readable Notes or custom Annotation classes.

Table 3.4 shows all attributes and derived classes for the SEDBase element as defined in the SED-ML XML Schema.

attribute	description
metaID^o	page 16
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 1: *Attributes and nested elements for SEDBase.* elements^o *denotes optional elements.*

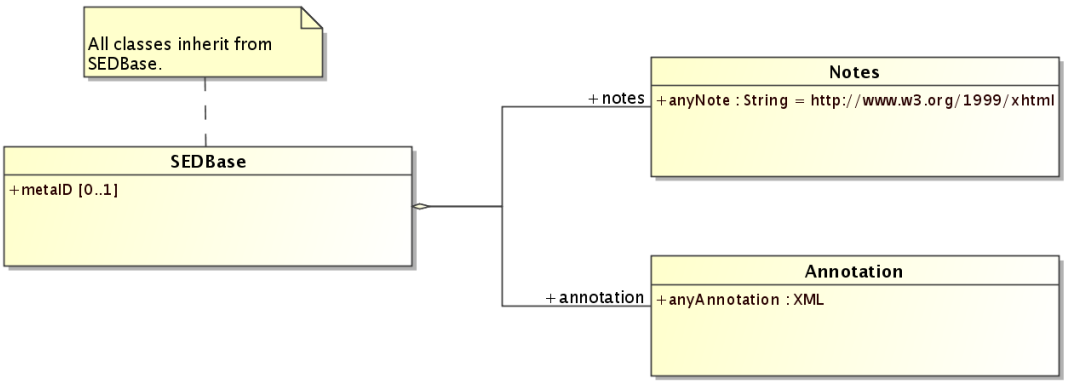

Figure 10: *The SEDBase class*

3.4.1 `metaid` Attribute

The main purpose of the `metaid` attribute is to attach Annotations to SED-ML elements. Thus, the `metaID` attribute has to be globally unique throughout the whole SED-ML document.

`metaID` is of type XML ID.

3.4.2 The Notes Class

A note is considered a human-readable description of the element it is assigned to. It serves to display information to the user. Notes contains any valid XHTML [Pemberton et al., 2002], ranging from short comments to whole HTML pages for display in a Web browser.

Notes has a mandatory attribute xmlns to declare the `XHTML` namespace. It does not have any further subclasses nor attributes associated to it (see Table 3.4.2).

Table 3.4.2 shows all attributes and derived classes for the Notes element as defined in the SED-ML XML Schema.

attribute	**description**
xmlns	page 15
sub-elements	**description**
none	

Table 2: *Attributes and nested elements for Notes.* elements^o *denotes optional elements.*

The namespace URL for `XHTML` content inside the Notes class is `http://www.w3.org/1999/xhtml`. It may either be declared in the `sedML` XML element, or directly for use in each `notes` element of the XML file. For further options of how to set the namespace and detailed examples, please refer to ([Hucka et al., 2010], p. 14).

Listing 5 shows the use of the `notes` element in a SED-ML file.

```
1 <sedML version="0.1">
2  <notes http://www.w3.org/1999/xhtml>
3   The enclosed simulation description shows the oscillating behaviour of the Repressilator
        model using deterministic and stochastic simulators.
4  </notes>
5 </sedML>
```

Listing 5: *The `notes` element*

In this example the namespace declaration is inside the `notes` element and the note is related to the `sedML` root element of the SED-ML file. A note may, however, occur inside *any* SED-ML

XML element, except `note` itself and annotation.

3.4.3 The Annotation Class

An annotation is considered a computer-processible piece of information. Annotations may contain any valid XML content. No additional namespace declaration is needed for use of the annotation element. However, for the type of XML inside annotation, the according namespaces should be declared, if necessary. For further guidelines on how to use annotations, we would like to encourage the reading of the according section in the SBML specification ([Hucka et al., 2010], pp. 14-16).

Table 3.4.3 shows all attributes and derived classes for the Annotation element as defined in the SED-ML XML Schema.

attribute	description
none	
sub-elements	**description**
none	

Table 3: *Attributes and nested elements for Annotation.* elements^o *denotes optional elements.*

Listing 6 shows the use of the `annotation` element in a SED-ML file.

```
<sedML>
  [..]
  <model id="model1" metaID="001" name="Circadian Oscillations" language="
      urn:sedml:language:cellml"
   source="http://models.cellml.org/workspace/leloup_gonze_goldbeter_1999/@@rawfile/
       d6613d7e1051b3eff2bb1d3d419a445bb8c754ad/leloup_gonze_goldbeter_1999_a.cellml" >
   <annotation>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:bqmodel="http://biomodels.net/model-qualifiers/">
     <rdf:Description rdf:about="#001">
      <bqmodel:isDescribedBy>
       <rdf:Bag>
        <rdf:li rdf:resource="urn:miriam:pubmed:10415827"/>
       </rdf:Bag>
      </bqmodel:isDescribedBy>
     </rdf:Description>
    </rdf:RDF>
   </annotation>
  </model>
  [..]
</sedML>
```

Listing 6: *The annotation element*

In the given example a SED-ML model element is annotated by referencing the original publication the model is build upon. The `model` contains an `annotation` that uses the model-qualifier `isDescribedBy` to link to the external resource `urn:miriam:pubmed:10415827`. In natural language the annotation content could be interpreted as "The model *is described by* the published article available from *pubmed* under ID *10643740*". The example annotation follows the proposed URI Scheme suggested by MIRIAM. The MIRIAM URN can be resolved to the PubMED (`http://pubmed.gov`) publication with ID 10415827, namely the article "Alternating oscillations and chaos in a model of two coupled biochemical oscillators driving successive phases of the cell cycle." published by Romond et al. in 1999.

3.5 The SED-ML Class

Each SED-ML Level 1 Version 1 document has a main class called SED-ML (Figure 11) which defines the document's structure and content. A SED-ML document needs to have the SED-ML

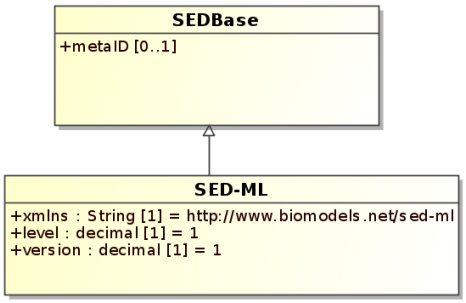

Figure 11: *The SED-ML class*

namespace defined through the mandatory xmlns attribute. In addition, the SED-ML level and version attributes are mandatory. All of them are explained in more detail in the following subsections.

The SED-ML document is build of several parts which are all connected to the SED-ML class through aggregation: the Model class (for model specification, see section 4.1), the Simulation class (for simulation setup specification, see section 4.3), the Task class (for the linkage of models and simulation setups, see section 4.4), the DataGenerator class (for the definition of post-processing, see section 4.5), and the Output class (for the output specification, see section 4.6). All of them are shown in Figure 12 on the following page and will be explained in more detail in the according sections of this document.

Table 3.5 shows all attributes and derived classes for the SED-ML element as defined in the SED-ML XML Schema.

attribute	**description**
metaID^o	page 16
xmlns	page 15
level	page 19
version	page 19
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
model^o	page 30
simulation^o	page 36
task^o	page 39
dataGenerator^o	page 40
output^o	page 41

Table 4: *Attributes and nested elements for SED-ML.* elements^o *denotes optional elements.*

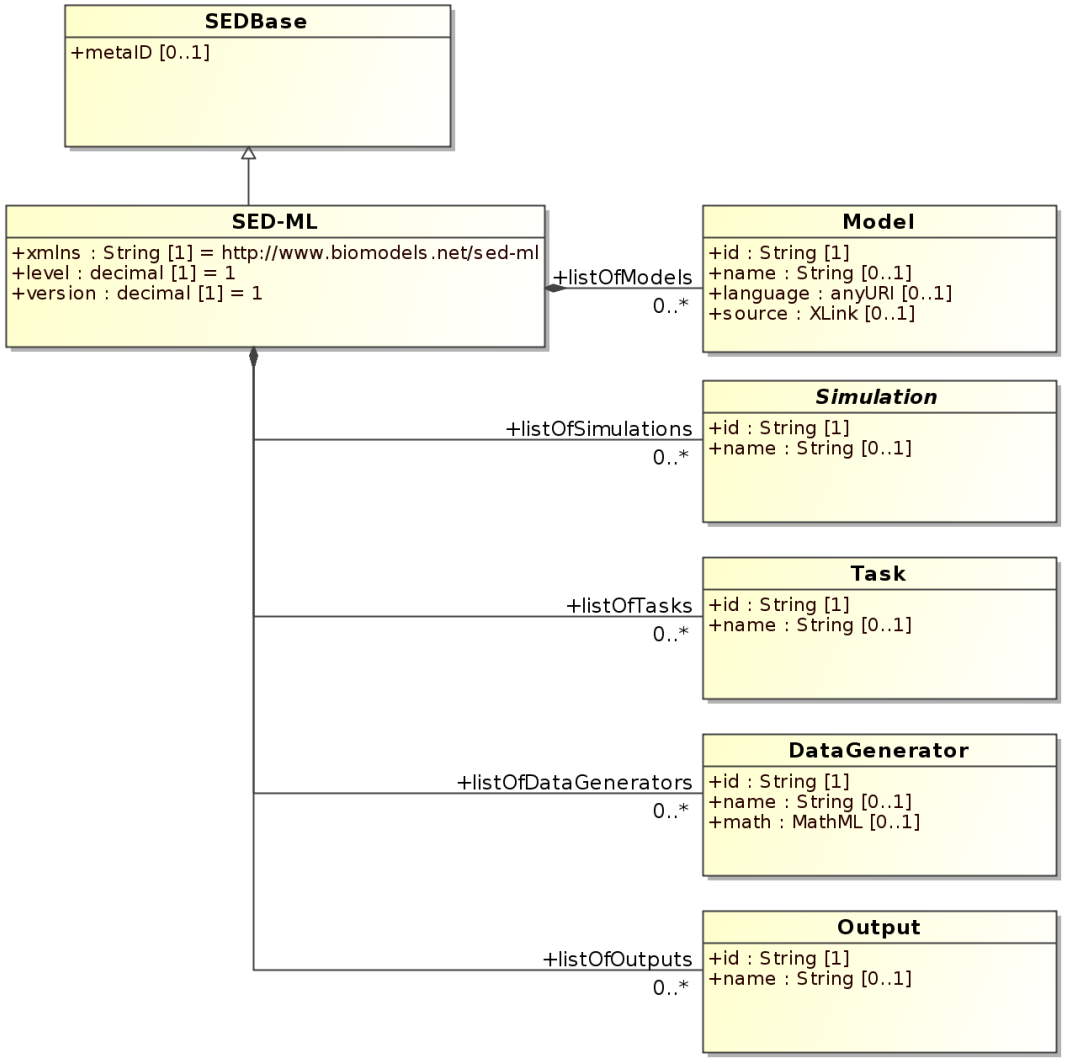

Figure 12: *The sub-classes of SED-ML*

3.5.1 The `level` attribute

The current SED-ML level is Level 1. Major revisions containing substantial changes will lead to the definition of forthcoming levels.

The level attribute is **`required`** and its value is a **`fixed`** decimal. For SED-ML Level 1 Version 1 the value is set to `1`.

3.5.2 The `version` attribute

The current SED-ML version is 1. Minor revisions containing corrections and refinements of SED-ML elements will lead to the definition of forthcoming versions.

The version attribute is **`required`** and its value is a **`fixed`** decimal. For SED-ML Level 1 Version 1 the value is set to `1`.

The basic XML structure of a SED-ML file is shown in listing 7.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <sedML xmlns:math="http://www.w3.org/1998/Math/MathML"
3        xmlns="http://www.biomodels.net/sed-ml"
```

```
4         level="1"
5         version="1">
6   <listOfModels />
7    [MODEL REFERENCES AND APPLIED CHANGES]
8   <listOfSimulations />
9    [SIMULATION SETUPS]
10  <listOfTasks />
11   [MODELS LINKED TO SIMULATIONS]
12  <listOfDataGenerators />
13   [DEFINITION OF POST-PROCESSING]
14  <listOfOutputs />
15   [DEFINITION OF OUTPUT]
16 <\sedML>
```

Listing 7: *The SED-ML root element*

The root element of each SED-ML XML file is the `sedML` element, encoding version and level of the file, and setting the necessary namespaces. Nested inside the `sedML` element are the five aforementioned lists serving as containers for the encoded data (listOfModels for all models, listOfSimulations for all simulations, listOfTasks for all tasks, listOfDataGenerators for all post-processing definition, and listOfOutputs for all output definitions).

3.6 The reference Relations

The reference association is used to refer to a particular source. It may occur in four different ways in the SED-ML document:

1. as an association between a Variable and a Model (modelReference)
2. as an association between a Variable and a Task (taskReference)
3. as an association between the Task and the associated Model (modelRereference)
4. or as an association between the Task and the Simulation (simulationReference)

Depending on the use of the reference relation in connection with a Variable object, it may take different roles: The reference association might occur between a Variable and a Model, when the Variable is to define a Change. In that case the `variable` element contains a modelReference, i. e. a reference to the model that must be used to execute the XPath expression given in the target arribute of the Variable. An example is shown in listing 8

If the reference is used as an association between a Variable object and a inside the dataGenerator class, then the `variable` contains a taskReference to unambiguously refer to an observable in a given task.

If the reference is used as an association between a Task and a Simulation then the Task contains a simulationReference.

3.6.1 model Reference

The modelReference might occur as a relation between a Variable object and a Model object, or as a relation between a Task object and a Model object.

If pre-processing needs to be applied to a model before simulation the model update can be specified by creating a Change object. If the change is calculated using mathematics, variables need to be defined. To refer to an existing entity in a defined Model, the modelReference is used. Listing 8 shows the definition of a model reference.

```
1 <model id="m0001" [..]>
2  <listOfChanges>
3    <computeChange>
4     <listOfVariables>
```

```
5       <variable id="v1" modelReference="cellML" target="/cellml:model/cellml:component[
            @cmeta:id='MP']/cellml:variable[@name='vsP']/@initial_value" />
6     </listOdVariables>
7     <listOfParameters [..] />
8     <math>
9       [CALCULATION OF CHANGE]
10    </math>
11   </computeChange>
12  </listOfChanges>
13  [..]
14 </model>
```

Listing 8: *SED-ML* `modelReference` *definition inside a* `computeChange` *element*

In the example, a change using mathematical functions is defined for model `m0001`. In the `computeChange` a list of variables is defined. One of those variable is `v1` which is defined in another model, namely `cellML`. To identify the variable in model `cellML` the XPath expression given in the target attribute.

The modelReference is as well used to define that a Model object is used in a particular Task. Listing 9 shows how this can be done for a sample SED-ML document.

```
1 <listOfTasks>
2  <task id="t1" name="Baseline" modelReference="model1" simulationReference="simulation1" />
3  <task id="t2" name="Modified" modelReference="model2" simulationReference="simulation1" />
4 </listOfTasks>
```

Listing 9: *SED-ML* `modelReference` *definition inside a* `task` *element*

The example defines two different tasks, the first one applies the simulation settings of `simulation1` on `model1`, the second one applies the same simulation settings on `model2`.

3.6.2 taskReference

In order to define the output of a simulation so-called DataGenerator objects are created that take the simulation results and apply post-processing to them. To define certain types of post-processing, Variable objects need to be created. Those link to a defined Task from which the model that contains the variable of interest can be inferred. A taskReference association is used to realise that link from a Variable object inside a DataGenerator to a Task object. Listing 10 shows an example for a taskReference inside a DataGenerator.

```
1 <listOfDataGenerators>
2  <dataGenerator id="tim3" name="tim mRNA (difference v1-v2+20)">
3   <listOfVariables>
4    <variable id="v1" taskReference="t1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
         sbml:species[@id='Mt']" />
5   </listOfVariables>
6   <math />
7  </dataGenerator>
8 </listOfDataGenerators>
```

Listing 10: *SED-ML* `taskReference` *definition inside a* `dataGenerator` *element*

The example shows the definition of a variable `v1` in a `dataGenerator` element. The variable appears in the model that is used in task `t1`. The task definition of `t1` might look as follows:

```
1 <listOfTasks>
2   <task id="t1" name="task definition" modelReference="model1"
3         simulationReference="simulation 1" />
4 </listOfTasks>
```

In the above task definition the variable `v1` is part of the model `model1` as it is the model referred to from `t1`. As such, the variable defined in the data generator in example 10 is equivalent to the SBML species `Mt` in `model1`.

3.6.3 *simulationReference*

The simulationReference is used to refer to a particular Simulation in a Task. Listing 9 shows how the reference to a defined simulation for a sample SED-ML document. In the example, both tasks `t1` and `t2` use the simulation settings defined in `simulation1` to run the experiment.

3.7 The Variable Class

Variables in SED-ML are references to already existing constituents in one of the defined models. A variable always is placed inside a listOfVariables. Each instance of the Variable class (see

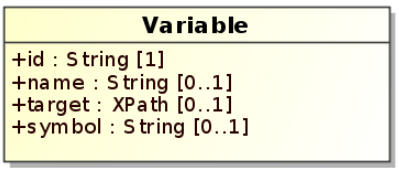

Figure 13: *The Variable class*

Figure 13) has a required id and an optional name. Variables are used to either refer to a model constituent, i. e. a model observable, such as an SBML species, or to refer to am implicit variable. The referenced constituent is specified through the mandatory target attribute in the first case, and through a symbol holding a MIRIAM URI in the second case.

Listing 11 shows a model with a listOfVariables declared that holds the variable definitions.

```
<model [..]>
 <listOfVariables>
   [VARIABLE DEFINITIONS FOLLOWING]
 </listOfVariables>
 [..]
</model>
```

Listing 11: *SED-ML* `variable` *definition*

Table 3.7 shows all attributes and derived classes for the Variable element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
target	page 23
symbol	page 24
taskReference	page 21
modelReference	page 20
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 5: *Attributes and nested elements for Variable.* elements^o *denotes optional elements.*

The reference to an object of the Variable class may occur in two different places of the SED-ML document:

First, it may occur in the Change class where it is used for describing the mathematical computation of a change of a model's observable, using other observables existing in a defined model.

The second use of the Variable class is for defining a DataGenerator. Here, a variable in an existing model or an implicit variable might be used to define the post-processing of the simulation.

3.7.1 The `target` attribute

An instance of Variable refers to a model constituent inside a particular model through an XPath expression stored in the required target attribute.

XPath allows to unambiguously identify an element or attribute in an XML file.

An example for a variable definition is given in listing 11.

```
<model id="m0001" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.
    db:BIOMD0000000012">
 <listOfChanges>
  <computeChange>
   <listOfVariables>
    <variable id="v1" name="Tet Repressor protein" taskreference="t1"  target="/sbml/
        listOfSpecies/species[@id="PY"]" />
   </listOfVariables>
   [CHANGE DEFINITION FOLLOWING]
  </computeChange>
 </listOfChanges>
</model>
```

Listing 12: *SED-ML `target` definition*

Please note that the identifier and names inside the SED-ML document do not have to comply with the identifiers and names that the model and its constituents carry in the model definition. In the above example 11, the variable with ID `v1` is defined. It is described as the `TetR protein`. The reference points to a species in the referenced SBML model. The particular species can be identified through its ID in the SBML model, namely `PY`. However, SED-ML does not forbid to use identical identifiers and names as in the referenced models neither. The following is the same valid SED-ML example for the specification of a variable as the above in listing 11, but with different naming:

```
<model id="m0001" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.
    db:BIOMD0000000012">
 <listOfVariables>
  <variable id="PY" name="TetR protein" target="/sbml/listOfSpecies/species[@id="PY"]" />
 </listOfVariables>
 [..]
</model>
```

Listing 13: *SED-ML variable definition using the original model identifier and name in SED-ML*

The XPath expression used in the `target` attribute unambiguously leads to the particular place in the XML SBML model – the species is to be found in the *sbml* element, and there inside the *listOfSpecies*:

```
<sbml [..]>
 <listOfSpecies]
  <species metaid="PY" id="PY" name="TetR protein" [..]>
   [..]
  </species>
 </listOfSpecies>
 [..]
</sbml>
```

Listing 14: *Species definition in the referenced model (extracted from `urn:miriam:biomodels.db:BIOMD0000000012`)*

3.7.2 The `symbol` attribute

Symbols are predefined, implicit variables that can be called in a SED-ML file by referring to the defined URNs representing that variable's concept. The notion of implicit variables is explained in section 2.2.3 on page page 13.

An example for a symbol definition is given in listing 15.

```
<model id="m0001" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.
    db:BIOMD0000000012">
 <listOfVariables>
  <variable id="t1" name="time" symbol="urn:sedml:symbol:time" />
 </listOfVariables>
 [..]
</model>
```

Listing 15: *SED-ML `symbol` definition*

3.8 The Parameter Class

An instance of the Parameter class in SED-ML defines a symbol associated with a constant value (see Figure 14). A parameter can unambiguously be identified through it's given id. It

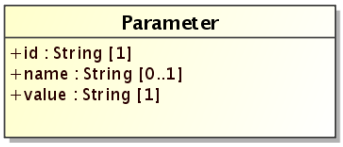

Figure 14: *The Parameter class*

may additionally carry an optional name. Each parameter has one associated value. Table 3.8 shows all attributes and derived classes for Parameter.

Table 3.8 shows all attributes and derived classes for the Parameter element as defined in the SED-ML XML Schema.

attribute	description
metaID^o	page 16
id	page 15
name^o	page 15
value	page 25
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 6: *Attributes and nested elements for Parameter.* elements^o *denotes optional elements.*

SED-ML uses parameters in two ways: First, they may occur in the Change class for describing the mathematical computation of a change of a model's observable through predefined parameters. The second use of the Parameter class is to refer from a DataGenerator definition to a constant value.

In both cases the parameter definitions are local to the according particular class defining them.

3.8.1 The `value` attribute

Each instance of Parameter defines a particular thing with a fixed value. The value in the XML representation is of the data type `String`. The `value` attribute is required for each `parameter` element. A parameter gets exactly one constant value assigned, meaning that the value is fixed and cannot be changed.

An example for a parameter definition is given in listing 16.

```
<listOfParameters>
  <parameter id="p1" name="KM" value="40" />
</listOfParameters>
```

Listing 16: *The definition of a parameter in SED-ML*

3.9 The ListOf containers

listOf* elements in SED-ML serve as containers for a collection of objects of the same type. For example, the listOfModels contains all Model objects of a SED-ML document. Lists do not carry any further semantics, nor do they add additional attributes to the language. They might, however, be annotated with Notes and Annotations as they are derived from SBase. All listOf* elements are optional in a SED-ML document.

SED-ML uses the following listOf* elements:

3.9.1 listOfVariables: The variable definition container

In order to refer to existing entities inside a model, these variables have to be referrable to from the simulation experiment description. SED-ML uses the XML listOfVariable element as a container for all necessary variables that need to be defined in order to either describe a change in the mathematics of a model (ComputeChange) or to define a data generator using existing variables from a given task (dataGeneratorClass). Figure 15 shows the use of the listOfVariables container. The listOfVariables is optional and may contain zero to many models.

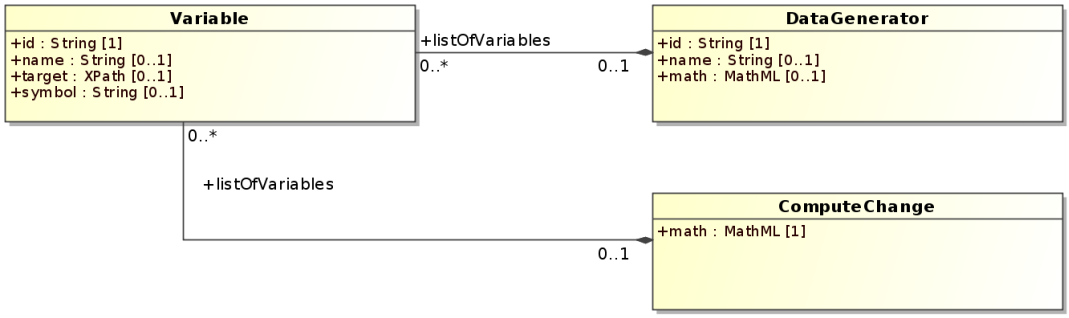

Figure 15: *The SED-ML listOfVariables container*

An XML code snippet for the definition of a `listOfVariables` element is shown in listing 17.

```
<listOfVariables>
  <variable id="v1" name="maximum velocity" target="/cellml:model/cellml:component[@cmeta:id
      ='MP']/cellml:variable[@name='vsP']/@initial_value" />
  <variable id="v2" target="SBML" />
</listOfVariables>
```

Listing 17: *SED-ML listOfVariables element*

3.9.2 listOfParameters: The parameter definition container

All parameters needed throughout the simulation experiment, either to apply a Change on a model prior to simulation or to set up a DataGenerator, are defined inside a listOfParameters.

Figure 16 shows the use of the listOfParameters container. It is optional and may contain zero to many models.

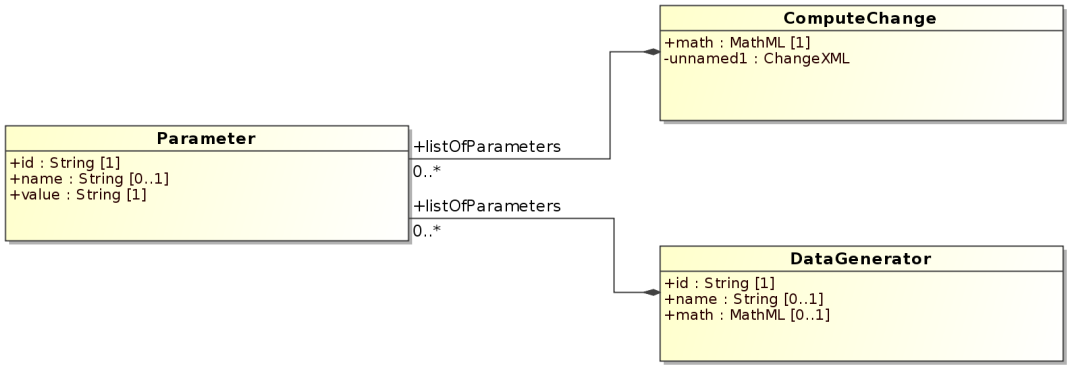

Figure 16: *The SED-ML* `listOfParameters` *container*

An XML code snippet for the definition of a `listOfParameters` element is shown in listing 18.

```
<listOfParameters>
  <parameter id="p1" value="1" />
  <parameter id="p2" name="Kadp_2" value="0.23" />
</listOfParameters>
```

Listing 18: *SED-ML* `listOfParameters` *element*

3.9.3 listOfModels: The model description container

In order to specify a simulation experiment, the participating models have to be defined. SED-ML uses the XML listOfModels element as a container for all necessary models (see Figure Figure 17. The listOfModels is optional and may contain zero to many models.

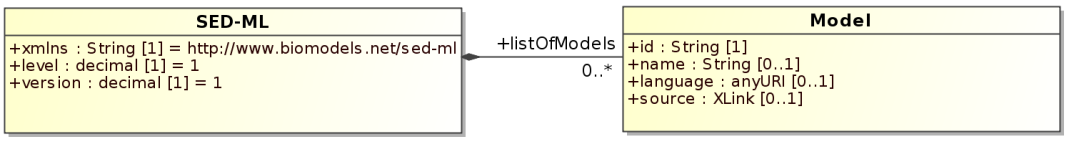

Figure 17: *The SED-ML listOfModels container*

An XML code snippet for the `listOfModels` element is shown in listing 19.

```
<listOfModels>
  <model id="m0001" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.
      db:BIOMD0000000012">
  [MODEL PRE-PROCESSING]
  </model>
  <model id="m0002" language="urn:sedml:language:sbml" source="m0001">
  [MODEL PRE-PROCESSING]
  </model>
  <model id="m0003" language="urn:sedml:language:cellml" source="http://www.cellml.org/
      models/leloup_gonze_goldbeter_1999_version02">
  [MODEL PRE-PROCESSING]
  </model>
```

```
11 </listOfModels>
```

Listing 19: *SED-ML listOfModels element*

The above `listOfModels` references three models: The first model (`m0001`) is the Repressilator model taken from BioModels Database. The model itself is available from `urn:miriam:biomodels.db:BIOMD0000000012`. For the SED-ML simulation, the model might undergo pre-processings, described in the Change class. Based on the description of the first model `m0001`, the second model is build. It refers to the model that was originally the `urn:miriam:biomodels.db:BIOMD0000000012` model, but had changes applied to it. `m0002` might then have even further changes applied to it on top of the changes defined in the pre-processing of `m0001`. The third model in the code example above is a different model in CellML representation. `m0003` is the model available from `urn:miriam:biomodels.db:BIOMD0000000012`, and might have additional pre-processing applied to it before used in the simulation. Such pre-processings might include a (new) parametrisation of model constituents, or a model update in terms of revised reaction equations, as well as the substitution of whole model constituents. Further details will be given in the description of the Change class.

A SED-ML description can be a sole storage container for a *general* simulation setting, comparible to an experiment procudure description. In that case, no particular model needs to be related to that description to store the settings for later use in SED-ML format:

```
1 <sedML>
2  <listOfModels />
3  [SIMULATION SETTINGS FOLLOWING]
4 </sedML>
```

3.9.4 listOfChanges: The change definition container

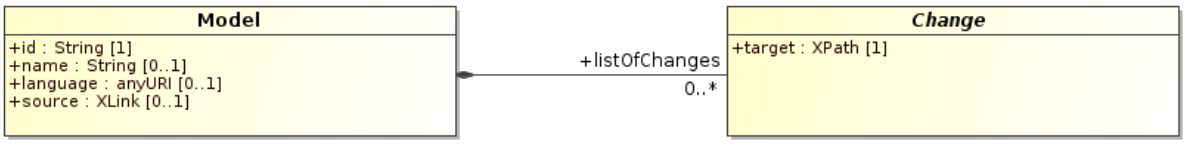

Figure 18: *The SED-ML listOfChanges container*

The listOfChanges container contains the defined changes to be applied to a particular model (see Figure 18). listOfChanges always occurs within a `model` element. It is an optional subelement of the `model` element.

Example 20 shows the definition of a `listOfChanges` container inside a `model` element.

```
1 <model id="m0001" [..]>
2  <listOfChanges>
3   [CHANGE DEFINITION]
4  </listOfChanges>
5 </model>
```

Listing 20: *The SED-ML `listOfChanges` element, defining a change on a model*

In the example a change is defined on the model `m0001`. To encode the change, a `listOfChanges` is created that contains all changes in single `change` elements, as explained in the definition of the Change class.

3.9.5 listOfSimulations: The simulation description container

The listOfSimulation is the container for simulation descriptions. The listOfSimulations is optional (see Figure 19 on the next page).

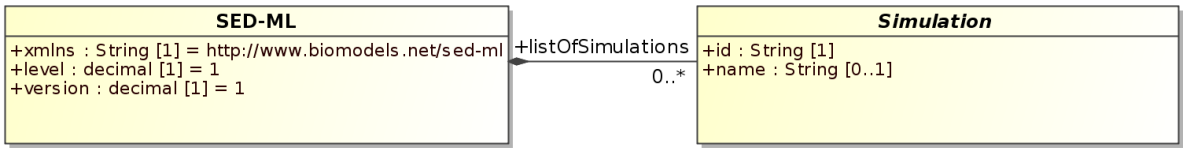

Figure 19: *The listOfSimulations container*

SED-ML version 1 offers the predefined simulation class UniformTimeCourse. Further simulation classes are planned for future versions of SED-ML, including simulation classes for bifurcation analysis and parameter scans.

Example 21 shows the definition of a `listOfSimulation` container inside a `sedML` element.

```
1 <sedML>
2  <listOfSimulations>
3   [UNIFORM TIMECOURSE DEFINITION]
4  </listOfSimulations>
5  [..]
6 </sedML>
```

Listing 21: *The SED-ML `listOfSimulations` element, defining a change on a model*

In the example a `listOfSimulations` element is created that contains all simulations, encoded in single simulation elements. For all SED-ML Level 1 Version 1 documents, the encoded simulation definitions are instances of the Uniform Timecourse class.

3.9.6 listOfTasks: The task specification container

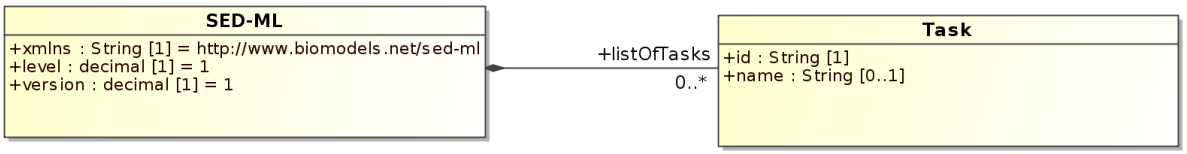

Figure 20: *The SED-ML listOfTasks container*

The listOfTasks container contains the defined tasks for the simulation experiment (see Figure 20).

An example for the definition of a task inside the listOfTasks is given in the XML snippet in listing 22.

```
1 <listOfTasks>
2  <task id="t1" name="simulating v1" modelReference="m1" simulationReference="s1">
3 </listOfTasks>
```

Listing 22: *The SED-ML listOfTasks element, defining one task*

In the given example, a task is created that makes use of one of the previously defined models and runs it with the simulation settings of one of the previously defined simulations.

3.9.7 listOfDataGenerators: The post-processing container

In SED-ML, all variables/parameters/values that shall be used in the Output class need to be defined as a dataGenerator beforehand, in the listofDataGenerators (see Figure Figure 21 on the next page).

Each dataGenerator is identifiable within the experiment by its unambiguous id. It can be further characterised by an optional name. The math element contains a mathML expression

Figure 21: *The SED-ML listOfDataGenerators container*

for the calculation of the data generator. Within the mathematical expression, variables defined in the listOfVariables and parameters defined in the listOfParameters can be used.

3.9.8 listOfOutputs: The output specification container

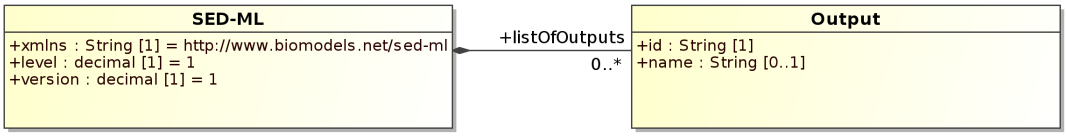

Figure 22: *The SED-ML listOfOutputs container*

The listOfOutputs container holds the output specifications for a simulation experiment. In SED-ML the output can be defined as either a report, a plot2D or as a 3D plot. The listing 23 shows the definition of two different outputs, one being a data table (report), the other one a 2D plot.

```
<listOfOutputs>
 <report id="report1" name="sample report">
  <listOfDataSets>
   [..]
  </listOfDataSets>
 </report>
 <plot2D id="plot1" name="sample 2D plot">
  <listOfCurves>
   [..]
  </listOfCurves>
 </plot2D>
</listOfOutputs>
```

Listing 23: *The listOfOutput element*

4 SED-ML Components

In this section we describe the major components of SED-ML. In the following we use the UML notation described in section 1.2.1. For an overview we provide a BNMP diagram in Appendix B and an XML Schema in C.

4.1 Model

The Model class is the container for a model reference (see Figure 23).

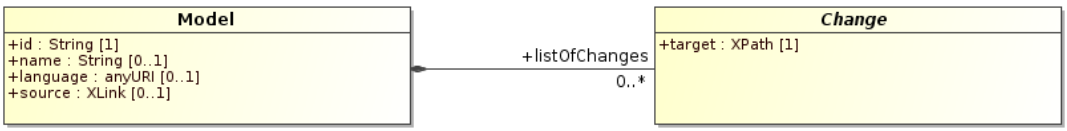

Figure 23: *The SED-ML Model class*

A model is defined through an unambiguous and mandatory id. An additional, optional name may be given to the model. For each model, the language may be specified, defining the format the model is encoded in, if such a format exists. Example formats are SBML, CellML, or myFancyLanguageML. The Model class refers to the particular model of interest through the source attribute. The restrictions on the model reference are

- The model must be encoded in an XML format.
- To refer to the model encoding language, a reference to a valid definition of that XML format must be given.
- To refer to a particular model in an external resource, an unambiguous reference must be given.

Table 4.1 shows all attributes and derived classes for the Model element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
language^o	page 31
source	page 31
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
change^o	page 32

Table 7: *Attributes and nested elements for Model.* elements^o *denotes optional elements.*

A model might need to undergo pre-processings before simulation. Those pre-processings are specified in the SED-ML Change class.

4.1.1 The `language` attribute

The evaluation of a SED-ML file will decide whether or not it can be used for a particular simulation environment. One crucial criterium is the particular model representation language used to encode the model. A simulation software usually only supports a small subset of the representation formats available to model biological systems computationally.

To help a software decide whether or not it supports a SED-ML description file, the information on the model encoding for each referenced model can be provided through the language attribute. As the description of a language name and version through an unrestricted `String` is error-prone, SED-ML provides a set of standard URIs to refer to particular language definitions. A prerequisite for a language to be fully supported by SED-ML is that a formalised language definition, e. g. an XML Schema, is provided online. The notion of URNs is described in section 2.2 on page 12.

To refer to a specific language and it's version, following the idea of MIRIAM URNs, the SED-ML URN scheme `urn:sedml:language:`*language name* is used. A model in a SED-ML file of type "SBML Level 2 Version 2" can be referred to, for example, through the URN `urn:sedml:language:sbml.level-2.version-2`.

The list of URNs for languages so far associated with SED-ML is available from the SED-ML web site on `http://biomodels.net/sed-ml`.

The `language` attribute is optional for the XML representation of a SED-ML file. If it is not explicitely defined otherwise in the SED-ML file, the default value for the `language` attribute is `urn:sedml:language:xml`, referring to any XML based model representation.

However, the use of the `language` attribute is strongly encouraged. Not only does it help a user decide whether or not he is able to run the simulation, that is to parse the model referenced in the SED-ML file. The language attribute is also needed to decide how to handle a particular implicit variable in the Variable class. The interpretation of implicit variables depends on the language of the representation format. The concept of implicit variables has been introduced in section 2.2.3 on page 13.

4.1.2 The `source` attribute

To make the model available during execution of a SED-ML file, the model `source` should be specified through an XLink. The XLink should preferably point to a public, consistent URI that contains the model description file and follows the proposed URI Scheme. Consistent URIs ensure the long-term availability of models used in a SED-ML simulation description file. Therefore, reference to curated, open model bases are recommended. An example for such a resource is BioModels Database [Le Novère et al., 2006]. However, any resource registered with MIRIAM resources[2] can easily be used in SED-ML. Even without a MIRIAM URN, SED-ML can be used. The long-term availability of a model, and thus the validity of a simulation experiment cannot be assured in those cases though.

An example for the definition of a model inside the listOfModels and using the MIRIAM URI scheme is given in the XML snippet in listing 24.

```
<listOfModels>
 <model id="m1" name="repressilator" language="urn:sedml:language:sbml"
  source="urn:miriam:biomodels.db:BIOMD0000000012">
  <listOfChanges>
   [DEFINE MODEL PRE-PROCESSING HERE]
  </listOfChanges>
 </model>
</listOfModels>
```

Listing 24: *The SED-ML model element, using the URI scheme*

[2] `http://www.ebi.ac.uk/miriam/main/`

In the example one model is defined. An `id` and a `name` are given. The `language` the model is described in is `SBML`, and the model source code is available from `urn:miriam:biomodels.db:BIOMD0000000012`. The MIRIAM URN can be resolved into the SBML model stored in BioModels Database under ID `BIOMD0000000012`.

An example for the definintion of a model inside the listOfModels and using a URL is given in the XML snippet in listing 25.

```
<listOfModels>
 <model id="m1" name="repressilator" language="urn:sedml:language:cellml"
  source="http://models.cellml.org/exposure/bba4e39f2c7ba8af51fd045463e7bdd3/aguda_b_1999.
      cellml">
  <listOfChanges>
   [DEFINE MODEL PRE-PROCESSING HERE]
  </listOfChanges>
 </model>
</listOfModels>
```

Listing 25: *The SED-ML model element, using a URL*

In the example one model is defined. An `id` and a `name` are given. The `type` of the model is `CellML`. As CellML currently does not provide a MIRIAM URI scheme for model reference, the URL pointing to the model code is used to refer to the model. The URL is given in the `source` element.

4.2 The Change Class

SED-ML not only allows to use the sole model for simlation, but on the contrary enables the description of changes to be made on the model before simulation using the listOfChanges element. Changes can be of three different types (see Figure 24 on the following page):

1. Changes on attributes of the model's XML representation (ChangeAttribute)
2. Changes on any XML snippet of the model's XML representation (ChangeXML)
3. Applying changes based on mathematical calculations (ComputeChange)

The Change class is abstract and serves as the container for all different types of changes.

Table 4.2 shows all attributes and derived classes for the Change element as defined in the SED-ML XML Schema.

attribute	**description**
metaid^o	page 16
id	page 15
name^o	page 15
target	page 23
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
changeXML^o	page 33
changeAttribute^o	page 34
computeChange^o	page 35

Table 8: *Attributes and nested elements for Change.* elements^o *denotes optional elements.*

Each Change has a target attribute that holds a valid XPath expression pointing to the XML element or XML attribute that is to undergo the defined changes.

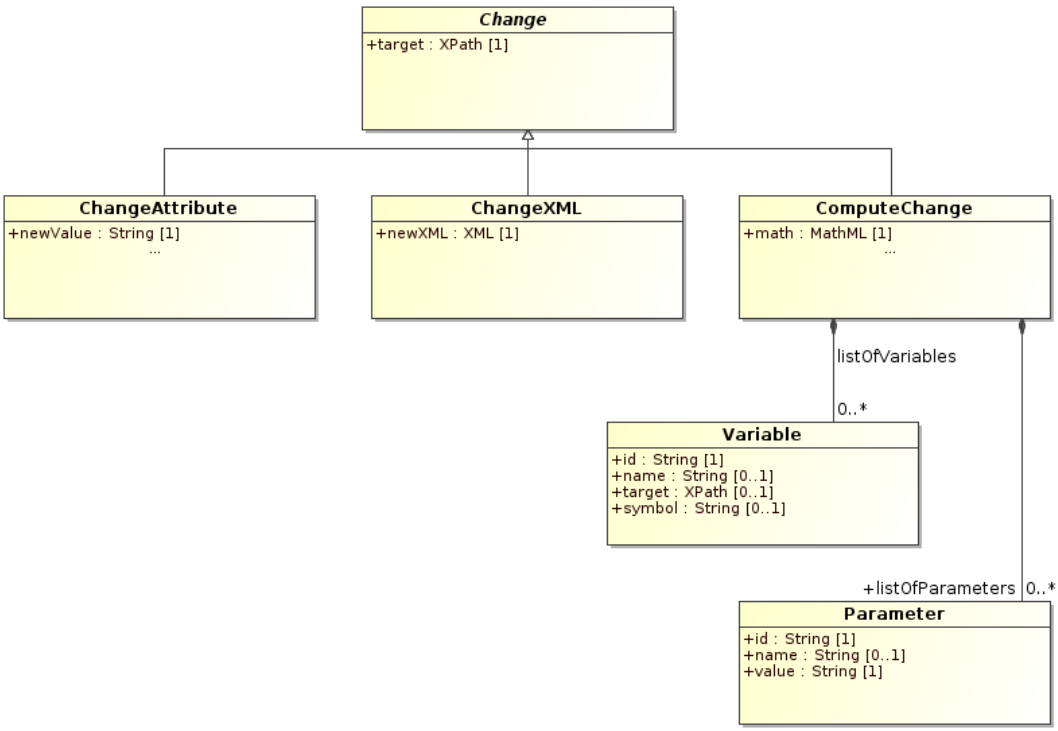

Figure 24: *The SED-ML Change class*

The Change class itself is abstract, a SED-ML document therefor will always contain one of the classes derived from Change; those currently are the aforementioned classes ChangeAttribute, ChangeXML, or ComputeChange.

4.2.1 The ChangeXML Class

The ChangeXML element is used to change any XML element in the model that can be addressed by a valid XPath expression.

Table 4.2.1 shows all attributes and derived classes for the changeXml element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
target	page 23
newXML	page 34
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 9: *Attributes and nested elements for ChangeXML.* elements^o *denotes optional elements. , newXML is an attribute, probably should be made an element!? (see problems in listing 26)*

The XPath is specified in the required target (see definition on page 23).

4.2.2 The newXML attribute

The new piece of XML code that is to substitute the XML element addressed by the XPAth is provided in the required **newXML** attribute.

An example that adds an additional parameter to a model is given in listing 26.

```
<model [..]>
 <listOfChanges>
  <changeXML target="/sbml:sbml/sbml:model/sbml:listOfParameters
                     /sbml:parameter[@id='V_mT']"
   newXML="<parameter metaid="metaid_0000010" id="V_mT" value="0.7">
           <parameter metaid="metaid_0000050" id="V_mT_2" value="4.6">" />
 </listOfChanges>
</model>
```

Listing 26: *The changeXML element*

4.2.3 The ChangeAttribute Class

The ChangeAttribute allows to update the value of an XML attribute in the model. One required attribute for ChangeAttribute is the target of change, i. e. the location of the addressed XML attribute.

Table 4.2.3 lists all attributes of the ChangeAttribute class.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
target	page 23
newValue	page 34
derived class	**description link**
notes^o	page 16
annotation^o	page 17

Table 10: *Attributes and nested elements for ChangeAttribute.* elements^o *denotes optional elements.*

The ChangeXML class covers the possibilities provided by the ChangeAttribute class. That is, everything that can be expressed by a ChangeAttribute construct can also be expressed by a ChangeXML. However, both concepts exist to allow for being very specific in defining changes. It is recommended to use the ChangeAttribute for any changes of an XML attribute, and to use the more general ChangeXML for all other cases.

4.2.4 The newValue attribute

The second required attribute is the **newValue**, which assignes a new value to the targeted XML attribute. tbw

For SBML, an example is the update of the initial concentration of a certain parameter, as shown in listing 27.

```
<model id="model1" name="Circadian Chaos" language="urn:sedml:language:sbml" source="
    urn:miriam:biomodels.db:BIOMD0000000021">
 <listOfChanges>
  <changeAttribute target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='
      V_mT']/@value" newValue="0.28"/>
  <changeAttribute target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='
      V_dT']/@value" newValue="4.8"/>
 </listOfChanges>
```

```
6 </model>
```

Listing 27: *The changeAttribute element*

4.2.5 The ComputeChange Class

The ComputeChange is used to make changes on any element of the XML file addressable by an XPath expression, where the changes are described by mathematical expressions through MathML.

Table 4.2.5 shows all attributes and derived classes for the ComputeChange element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
target	page 23
math	page 35
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
variable^o	page 22
parameter^o	page 24

Table 11: *Attributes and nested elements for ComputeChange.* elements^o *denotes optional elements.* , *math currently is an attribute, probably should be turned into an element (see listing 28)*

The `target` attribute contains the XPath addressing the piece of XML that is to be changed. It is possible to introduce additional parameters for the mathematics. Therefore, the parameters first need to be defined in the listOfParameters. They are then referenced through their ID. To use model variables for the definition of a mathematical expression, those variables need to be defined in the listOfVariables first, and can then be incorporated through their ID.

4.2.5.1 The `math` attribute

The `math` element is used to define mathematical functions. If used as an attribute of the ComputeChange class, it computes the change of the element or attribute addressed by the target attribute.

An example is given in listing 28.

```
1  <model [..]>
2      <computeChange target="/sbml/model/listOfParameters/parameter[@id='w']">
3        <listOfVariables>
4          <variable id="camkii" name="active calcium/calmoduline kinase II"
5                    target="/sbml/model[@id='calcium']/listOfSpecies/species[@id='KII']" />
6          <variable id="w" name="synaptic weight"
7                    target="/sbml/model[@id='synapse']/listOfParameters/parameter[@id='w']" /
                        >
8        <listOfVariables/>
9        <listOfParameters>
10         <parameter id="w0" name="synaptic weight change" value="1">
11         <parameter id="n" name="utrasensitivity to calcium" value="2">
12         <parameter id="K" name="sensitivity to calcium" value="1e-6">
13       <listOfParameters/>
14       <math>
15          <apply>
```

```
16            <plus />
17            <ci>w</ci>
18            <apply>
19              <times />
20              <ci>w0</ci>
21              <apply>
22                <divide />
23                <apply>
24                  <power />
25                  <ci>camkii</ci>
26                  <ci>n</ci>
27                </apply>
28                <apply>
29                  <plus />
30                  <apply>
31                    <power />
32                    <ci>K</ci>
33                    <ci>n</ci>
34                  </apply>
35                  <apply>
36                    <power />
37                    <ci> camkii </ci>
38                    <ci>n</ci>
39                  </apply>
40                </apply>
41              </apply>
42            </apply>
43          </apply>
44        </math>
45      </computeChange>
46    </listOfChanges>
47  </model>
```

Listing 28: *The computeChange element*

Level 1 Version 1 supports the subset of MathML 2.0 shown in section 2.1. A problem arises, because the individual supported model exchange languages allow different subsets of MathML. Thus, when an instance of ComputeChange replaces a mathematical expression of an SBML reaction, only the MathML subset allowed by SBML should be used here. FRANK/NLN: Should we propose a solution here or leave it as this?

4.3 The Simulation Class

A simulation is the execution of some defined algorithm(s). Therefore, a simulation is defined through it's `id`, an optional `name`, and the simulation algorithm used to run the simulation. Simulations are described differently depending on the type of simulation experiment to be performed. SED-ML Level 1 Version 1 does only support UniformTimeCourse simulations.

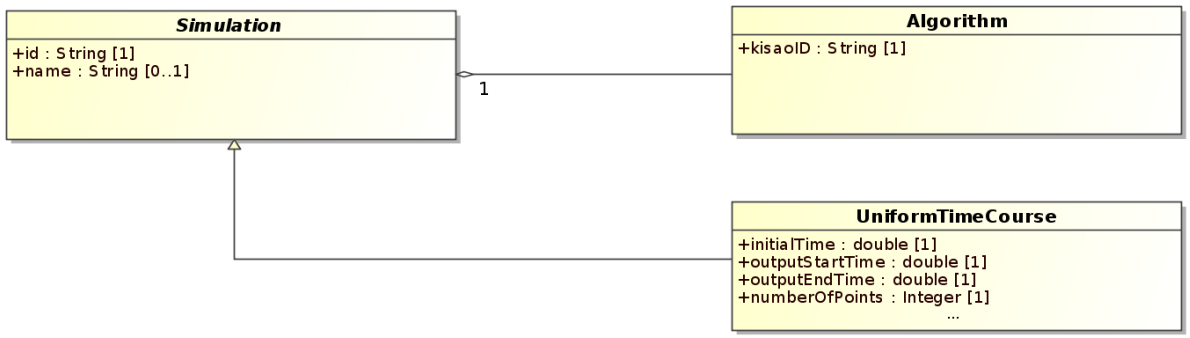

Figure 25: *The SED-ML Simulation class*

Table 4.3 on the next page shows all attributes and derived classes for the Simulation element

as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
algorithm	page 37

Table 12: *Attributes and nested elements for Simulation.* elements^o *denotes optional elements.*

An example for the definition of two different simulations is given in the XML snippet in listing 29.

```
<listOfSimulations>
  <uniformTimeCourse id="s1" name="time course simulation of variable v1 over 100 minutes"
      [..]>
    <algorithm>
      [ALGORITHM DEFINITION FOLLOWING]
    </algorithm>
  </uniformTimeCourse>
  <uniformTimeCourse id="s2" name="time course definition for concentration of p" [..]>
    [..]
  </uniformTimeCourse>
</listOfSimulations>
```

Listing 29: *The SED-ML* `listOfSimulations` *element, defining three different simulations*

Two timcourses with uniform range are defined. How to define the concrete algorithm is specified on page page 37.

4.3.1 The Algorithm Class

SED-ML makes use of the KiSA ontology to refer to a term in the controlled vocabulary identifying the particular simulation algorithm to be used in the simulation.

One algorithm must be defined for each simulation setup. The instance of the Algorithm class must contain a KiSAO reference to a simulation algorithm. The reference should define the simulation algorithm to be used in the simulation as precisely as possible.

Table 4.3.1 shows all attributes and derived classes for the Algorithm element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
kisaoID	page 13
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 13: *Attributes and nested elements for Algorithm.* elements^o *denotes optional elements.*

The example given in code snipped 29, completed by algorithm definitions looks as in listing 30.

```
<listOfSimulations>
  <uniformTimeCourse id="s1" name="time course simulation of variable v1 over 100 minutes"
      [..]>
      <algorithm kisaoID="KiSAO:0000030" />
  </uniformTimeCourse>
  <uniformTimeCourse id="s2" name="time course definition for concentration of p" [..]>
      <algorithm kisaoID="KiSAO:0000021" />
  </uniformTimeCourse>
</listOfSimulations>
```

Listing 30: *The SED-ML* `algorithm` *element, defining the two different algorithms in the two defined simulations*

For both simulations, one algorithm is defined. In the first simulation `s1` a deterministic simulation algorithm is used (Euler forward method), in the second simulation `s2` a stochastic one is used (Next reaction method).

4.3.2 The UniformTimeCourse Class

SED-ML Level 1 Version 1 so far only supports uniform time courses. Table 4.3.2 shows all attributes and derived classes for the UniformTimeCourse element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
initialTime	page 38
outputStartTime	page 39
outputEndTime	page 39
numberOfPoints	page 39
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
algorithm	page 37

Table 14: *Attributes and nested elements for UniformTimeCourse.* elements^o *denotes optional elements.*

An example for the definition of a uniform time course simulation is given in the XML snippet in listing 31.

```
<listOfSimulations>
 <uniformTimeCourse id="s1"  name="time course simulation of variable v1 over 100 minutes"
      initialTime="0" outputStartTime="0" outputEndTime="2500" numberOfPoints="1000">
    <algorithm kisaoID="KiSAO:0000030" />
 </uniformTimeCourse>
</listOfSimulations>
```

Listing 31: *The SED-ML* `uniformTimeCourse` *element, defining a uniform time course simulation over 2500 time units with 1000 simulation points, using the CVODE solver.*

4.3.3 The `initialTime` attribute

tbw

4.3.4 The `outputStartTime` attribute

tbw

4.3.5 The `outputEndTime` attribute

tbw

4.3.6 The `numberOfPoints` attribute

tbw

4.4 The Task Class

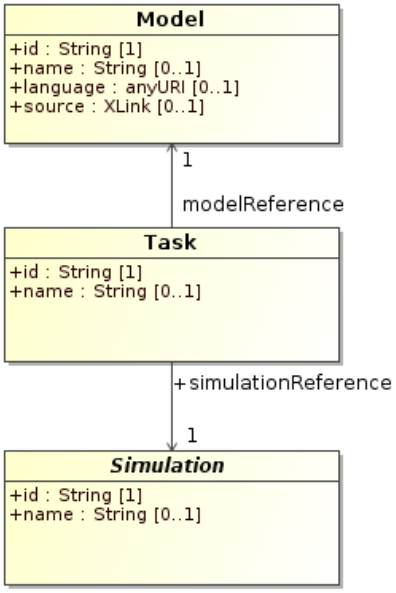

Figure 26: *The SED-ML Task class*

A task in SED-ML links a model as defined in the listOfModels to a certain simulation description as defined in the listOfSimulations via the two according IDs (model ID and simulation ID).

Table 4.4 shows all attributes and derived classes for the Task element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
modelReference	page 20
simulationReference	page 22
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 15: *Attributes and nested elements for Task.* elements^o *denotes optional elements.*

Each task does have its own task id for later reference and an optional name.

An example linking a simulation experiment to two different models using two task definitions within the list of tasks is given in listing 32.

```
<listOfTasks>
  <task id="t1" name="task definition" modelReference="model1"
        simulationReference="simulation 1" />
  <task id="t2" name="another task definition" modelReference="model2"
        simulationReference="simulation 1" />
</listOfTasks>
```

Listing 32: *The listOfTasks element*

In the example, a simulation setting *simulation1* is applied first to *model1* and then is applied to *model2*. Please note, that the tasks may be executed in any order, as XML does not have an ordering concept.

In SED-ML 1 it is only possible to link one simulation description to one model at a time. However, one can define as many tasks as needed within one experiment description, i. e. one SED-ML file.

4.5 The DataGenerator Class

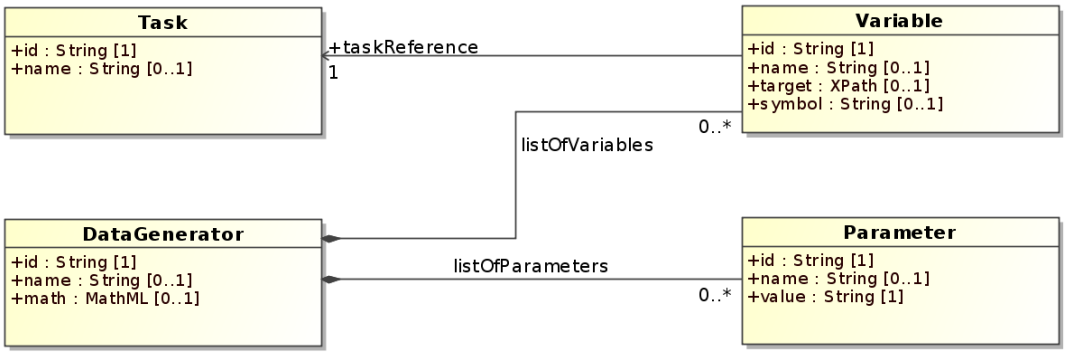

Figure 27: *The SED-ML DataGenerator class*

Table 4.5 shows all attributes and derived classes for the DataGenerator element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
math	page 35
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
variable^o	page 22
parameter^o	page 24

Table 16: *Attributes and nested elements for DataGenerator.* elements^o *denotes optional elements.*

The DataGenerator prepares the simulation data for usage in the output. Often, data needs to be post-processed before being returned to the user. Those post-processing steps can be simple normalisation of data, but also mathematical calculations. The data generator describes the rules to build the Output from simulation results and existing entities. The data Generator class is shown in Figure 27.

An example for a data generator is given in listing 33.

```
<listOfDatGenerators>
 <dataGenerator id="d1" name="time">
  <listOfVariables>
   <variable id="time" taskReference="task1" target="time" />
  </listOfVariables >
  <listOfParameters />
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <ci> time </ci>
  </math>
 </dataGenerator>
 <dataGenerator id="LaCI" name="LaCI repressor">
  <listOfVariables>
   <variable id="v1" taskReference="task1"
    target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
            sbml:species[@id='PX']" />
  </listOfVariables>
  <math:math>
   <math:ci>v1</math:ci>
  </math:math>
 </dataGenerator>
</listOfDataGenerators>
```

Listing 33: *The* `listOfDataGenerators` *element, defining two data generators* time *and* LaCI repressor

Mathematical functions available for the specification of DataGenerator variables are given in section 2.1.

4.6 The Output Class

The ouput container describes how the results of a simulaion should be presented to the user. It does not contain the data itself, but only the type of output and the data generators (data-Generator) which are to be used in the desired output type. Figure 28 shows the UML class definition of the Output class.

Table 4.6 shows all attributes and derived classes for the Output element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
plot2D^o	page 42
plot3D^o	page 43
report^o	page 43

Table 17: *Attributes and nested elements for Output.* elements^o *denotes optional elements.*

The types of output pre-defined in SED-ML are plots and reports, also referrible as data tables.

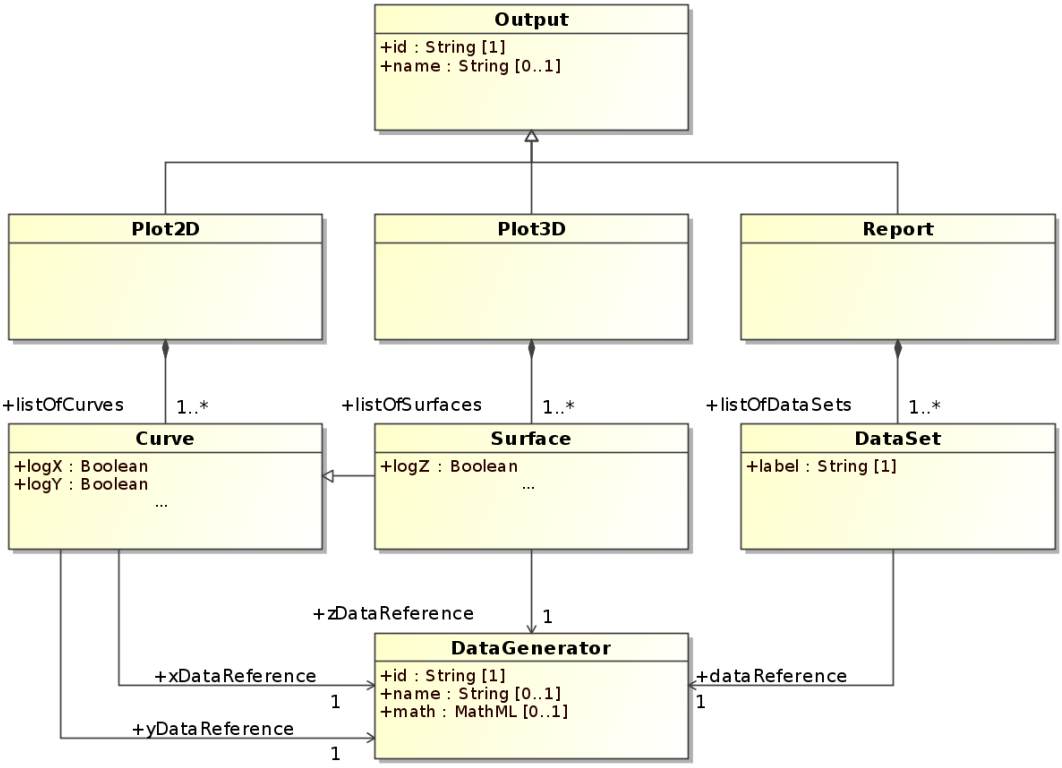

Figure 28: *The SED-ML Output class*

The output can be defined as a 2D plot or alternatively as a 3D plot.

4.6.1 The Plot2D Class

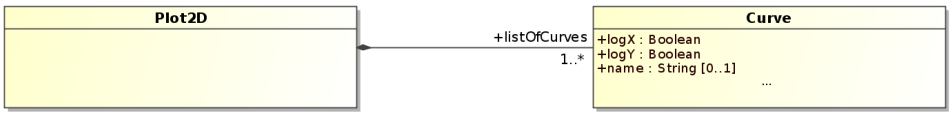

Figure 29: *The SED-ML Plot2D class*

Table 4.6.1 on the next page shows all attributes and derived classes for the Plot2D element as defined in the SED-ML XML Schema.

A 2D plot needs a data generator to be assigned to each of its two axes using the xDataReference to refer to a dataGenerator for the x-axis and using the yDataReference to refer to dataGenerator for the y-axis.

An example for the definition of a 2D-plot with 1 defined curve is given in listing 34.

```
<plot2D>
 <listOfCurves>
  <curve>
    [CURVE DEFINITION FOLLOWING]
  </curve>
 </listOfCurves>
</plot2D>
```

Listing 34: *The `plot2D` element with the nested `listOfCurves` element*

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
curve	page 44

Table 18: *Attributes and nested elements for Plot2D.* elements^o *denotes optional elements.*

Figure 30: *The SED-ML Plot3D class*

4.6.2 The Plot3D Class

Table 4.6.2 shows all attributes and derived classes for the Plot3D element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
surface	page 46

Table 19: *Attributes and nested elements for Plot3D.* elements^o *denotes optional elements.*

A 3D plot is defined similarly to a 2D plot, but having 3 axes. To every axis a data generator is assigned using the xDataReference, yDataReference, and the zDataReference. An example for a 3D plot with one defined surface is given in listing 35.

```
<plot3D>
 <listOfSurfaces>
  <surface>
   [SURFACE DEFINITION FOLLOWING]
  </surface>
 </listOfSurfaces>
</plot3D>
```

Listing 35: *The* `plot3D` *element with the nested* `listOfSurfaces` *element*

4.6.3 The Report Class

Table 4.6.3 on the next page shows all attributes and derived classes for the Report element as defined in the SED-ML XML Schema.

The Report class defines an output type that returns the simulation result in actual *numbers*. The particular columns of the report table are defined by creating a DataSet for each column.

Listing 36 provides an example for the definition of such a report with one column.

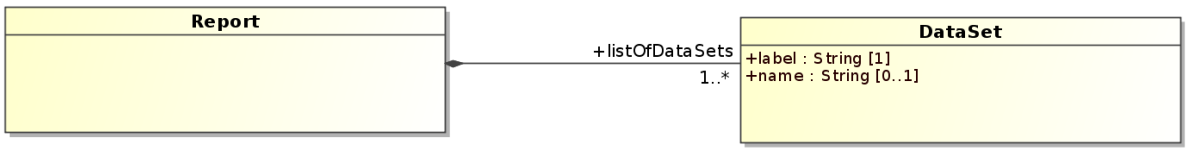

Figure 31: *The SED-ML Report class*

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
sub-elements	**description**
notes^o	page 16
annotation^o	page 17
dataSet	page 47

Table 20: *Attributes and nested elements for Report.* elements^o *denotes optional elements.*

```
<report>
 <listOfDataSets>
  <dataSet>
   [DATA REFERENCE FOLLOWING]
  </dataSet>
 </listOfDataSets>
</report>
```

Listing 36: *The* `report` *element with the nested* `listOfDataSets` *element*

The simulation result itself, i. e. concrete result numbers, are not stored in SED-ML, but the directive how to *calculate* them from the output of the simulator is provided through the dataGenerator.

4.7 The Curve Class

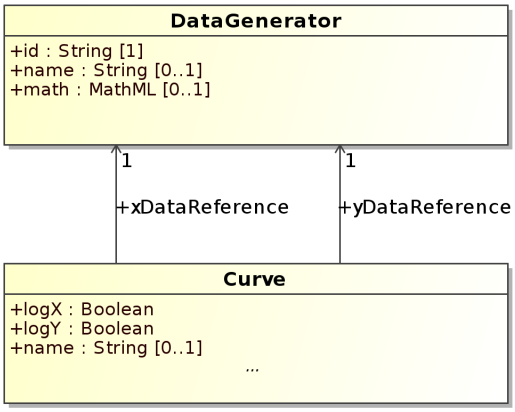

Figure 32: *The SED-ML Curve class*

Table 4.7 on the following page shows all attributes and derived classes for the Curve element as defined in the SED-ML XML Schema.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
logX	page 45
xDataReference	page 45
logY	page 45
yDataReference	page 46
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 21: *Attributes and nested elements for Curve.* elements^o *denotes optional elements.*

4.7.1 The logX attribute

logX is an optional attribute of the Curve class and defines whether or not the data output on the x-axis is logarithmic or not. The data type of logX is boolean, the standard value is "false". To make the output on the x-axis of a plot logarithmic, logX must be set to "true", as shown in the sample listing 37:

```
<listOfCurves logX="true">
  <curve id="c1" [..]>
</listOfCurves>
```

Listing 37: *The SED-ML logX attribute, defining a logarithmic output on the x-axis of the according output*

logX is also used in the definition of a Surface output.

4.7.2 The logY attribute

logY is an optional attribute of the Curve class and defines whether or not the data output on the x-axis is logarithmic or not. The data type of logY is boolean, the standard value is "false". To make the output on the y-axis of a plot logarithmic, logY must be set to "true", as shown in the sample listing 38:

```
<listOfCurves logY="true">
  <curve id="c1" [..]>
</listOfCurves>
```

Listing 38: *The SED-ML logY attribute, defining a logarithmic output on the y-axis of the according output*

logY is also used in the definition of a Surface output.

4.7.3 The xDataReference attribute

The xDataReference is a mandatory attribute of the Curve object. It's content refers to a dataGenerator ID which denotes the DataGenerator object that is used to generate the output on the x-axis of a Curve in a 2D Plot. xDataReference is also used in the definition of the x-axis of a Surface in a 3D Plot. The xDataReference data type is string. However, the number of valid values for the xDataReference is restricted to the IDs of already defined DataGenerator objects.

An example for the definition of a curve is given in listing 39.

4.7.4 The yDataReference attribute

The yDataReference is a mandatory attribute of the Curve object. It's content refers to a dataGenerator ID which denotes the DataGenerator object that is used to generate the output on the y-axis of a Curve in a 2D Plot. yDataReference is also used in the definition of the y-axis of a Surface in a 3D Plot. The yDataReference data type is string. However, the number of valid values for the yDataReference is restricted to the IDs of already defined DataGenerator objects.

An example for the definition of a curve is given in the XML snippet in listing 39.

```
1 <listOfCurves>
2   <curve id="c1" name="v1 / time" xDataReference="dg1" yDataReference="dg2" logX="true" />
3 </listOfCurves>
```

Listing 39: *The SED-ML* `curve` *element, defining the output curve showing the result of simulation for the referenced dataGenerators*

Here, only one curve is created, results shown on the x-axis are generated by the data generator `dg1`, results shown on the y-axis are generated by the data generator `dg2`. Both `dg1` and `dg2` need to be already defined in the listOfDataGenerators. The x-axis is plotted logarithmically.

4.8 The Surface Class

A surface is a three-dimensional Figure representing a simulation result. Figure 33 shows the definition of the Surface class used to encode the information needed for the creation of such Figures in SED-ML.

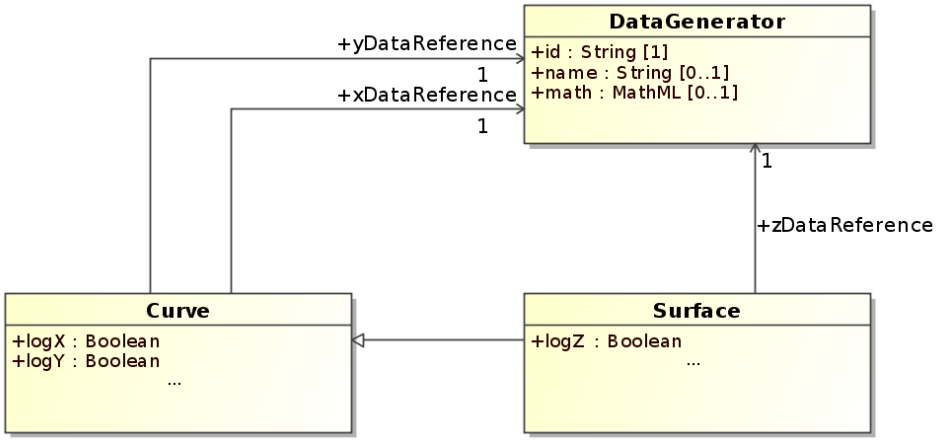

Figure 33: *The SED-ML Surface class*

Table 4.8 on the following page shows all attributes and derived classes for the Surface element as defined in the SED-ML XML Schema. To define a surface, the three different axes have to be defined, that is which data to plot on which axis and in which way. The aforementioned xDataReference and yDataReference attributes define the according dataGenerators for both the x- and y-axis of a surface. In addition, the zDataReference attribute defines the output for the z-axis. All axes might be logarithmic or not. This can be specified through the logX, logY, and the logZ attributes in the according dataReference elements.

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
logX	page 45
xDataReference	page 45
logY	page 45
yDataReference	page 46
logZ	page 47
zDataReference	page 47
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 22: *Attributes and nested elements for Surface.* elements^o *denotes optional elements.*

4.8.1 The logZ attribute

logZ is an optional attribute of the Surface class and defines whether or not the data output on the z-axis is logarithmic or not. The data type of logZ is boolean, the standard value is "false". To make the output on the z-axis of a surface plot logarithmic, logZ must be set to "true", as shown in the sample listing 40:

```
<listOfSurfaces logZ="true">
  <surface id="s1" [..]>
</listOfSurfaces>
```

Listing 40: *The SED-ML logZ attribute, defining a logarithmic output on the z-axis of the according output*

4.8.2 The zDataReference attribute

The zDataReference is a mandatory attribute of the Surface object. It's content refers to a dataGenerator ID which denotes the DataGenerator object that is used to generate the output on the z-axis of a 3D Plot. The zDataReference data type is string. However, the number of valid values for the zDataReference is restricted to the IDs of already defined DataGenerator objects.

An example for the definition of a surface is given in the XML snippet in listing 41.

```
<listOfSurfaces>
  <surface id="s1" name="surface" algorithm ="KiSAO:ID" xDataReference="dg1"
   yDataReference="dg2" zDataReference="dg3">
</listOfSurfaces>
```

Listing 41: *The SED-ML curve element, defining the output curve showing the result of the referenced task*

Here, only one curve is created, results shown on the x-axis are generated by the data generator `dg1`, results shown on the y-axis are generated by the data generator `dg2`, results shown on the z-axis are generated by the data generator `dg3`. All `dg1`, `dg2` and `dg3` need to be already defined in the listOfDataGenerators.

4.9 The DataSet Class

Table 4.9 on the next page shows all attributes and derived classes for the DataSet element as defined in the SED-ML XML Schema.

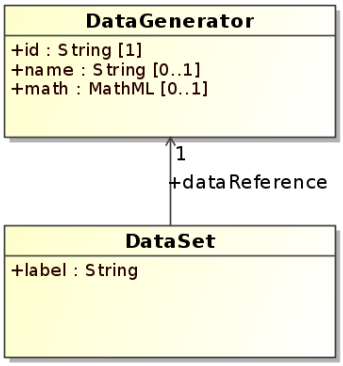

Figure 34: *The SED-ML DataSet class*

attribute	description
metaid^o	page 16
id	page 15
name^o	page 15
dataReference	page 48
label	page 48
sub-elements	**description**
notes^o	page 16
annotation^o	page 17

Table 23: *Attributes and nested elements for DataSet.* elements^o *denotes optional elements.*

4.9.1 The `dataReference` attribute

tbw

An example for the definition of a data set in form of a table is given in the XML snippet in listing 42.

```
1 <listOfDataSets>
2   <dataSet id="d1" name="v1 over time" dataReference="dg1" label="_1">
3 </listOfDataSets>
```

Listing 42: *The SED-ML* `dataSet` *element, defining a data set containing the result of the referenced task*

4.9.2 The `label` attribute

Each data set in a Report does have to carry an unambiguous label. The label is then used to refer to a particular data set in a report for later use.

Acknowledgements

The SED-ML specification has been developed with the input of many people. Main contributors of the current specification include Richard Adams, Frank Bergmann, Stefan Hoops, Nicolas Le Novère, Ion Moraru, Sven Sahle, Henning Schmidt and Dagmar Waltemath.

Moreover, we would like to thank all the participants of the meetings where SED-ML has been discussed a well as the subscribers of the sed-ml-discuss mailing list.

A SED-ML UML Overview

Figure 35 shows the complete UML diagram of the SED-ML. It gives the full picture of all implemented classes (see the XML Schema definition in 57).

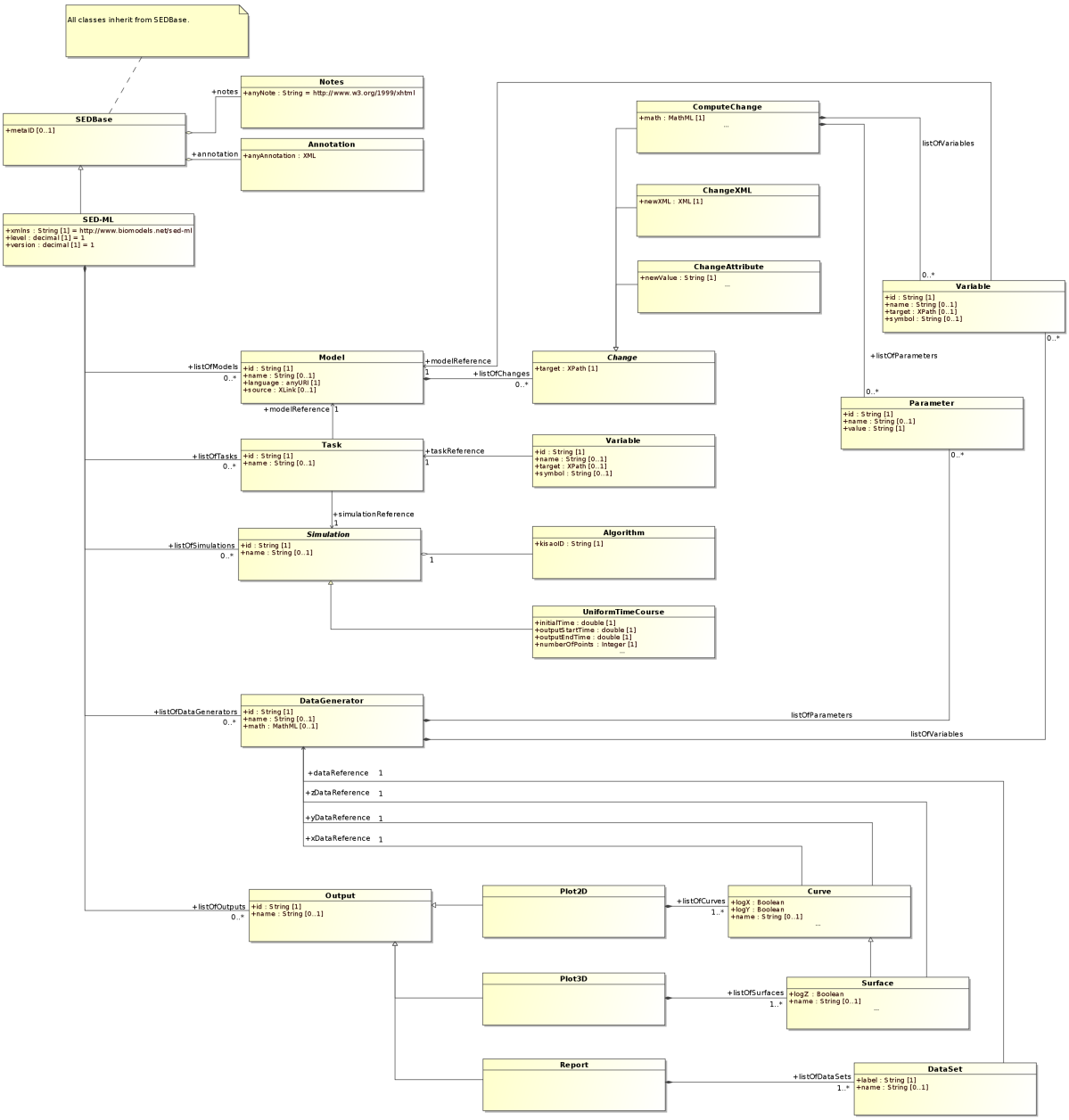

Figure 35: *The SED-ML UML class diagram*

B Overview of SED-ML

The *Simulation Experiment Description Markup Language* (SED-ML) is an XML-based format for the description of simulation experiments. It serves to store information about the simulation experiment performed on one or more models with a given set of outputs. Support for SED-ML compliant simulation descriptions will enable the exchange of simulation experiments across tools.

B.1 Conventions

The Busines Process Modeling Notation Version 1.2 (BPMN) was initially intended to describe internal business procedures (processes) in a graphical way. However, we will use BPMN to graphically describe the steps and processes of setting up a simulation experiment description. The major parts of BPMN that are used to specify SED-ML are activities, gateways, events, data, and documentation.

An *activity* is "work that is performed on a [..] process", for example "Specify the simulation settings". Activities may be atomic or non-atomic. SED-ML in particular makes use of the *task* activities, i.e., specific work units that need to be performed. Non-atomic tasks might be collapsed or expanded in the graphical representation (see Figure 36). Each collapsed subprocess has a corresponding expanded subprocess definition.

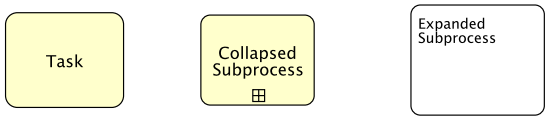

Figure 36: *BPMN activities: task, collapsed process, expanded subprocess*

Gateways serve as means to control the flow of sequence in the diagram. As the term already implies, a gateway needs some "mechanism that either allows or disallows passage through" [White et al., 2004]. The result of a gateway pass-through can be that processes are merged or splitted. Graphically, a gateway is represented as a diamond.

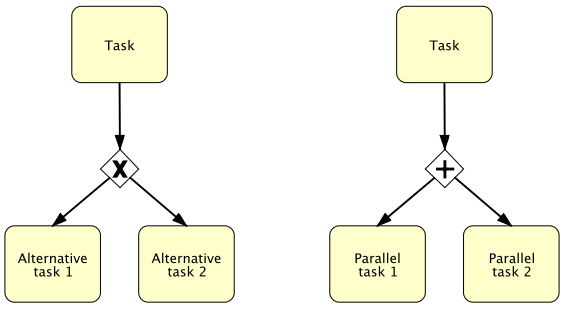

Figure 37: *BPML gateway types: Exclusive (left), parallel (right)*

While there exist a number of different gateway types (see [White et al., 2004], pp. 93), the SED-ML specification only uses the parallel and the exclusive gates (see Figure 37).

Exclusive gateways – also denoted as decisions – allow the sequence flow to take two or more

alternative paths (Figure 37 on the previous page, left hand side). However, *only one* of the paths may be chosen (not more). Sometimes two alternative branches need to be merged together again, in which case the exclusive gate must be used as well: The sequence flow continues as soon as *one* of the incoming processes send a signal. An exclusive gateways is marked by an `X` in the graphical notation.

Parallel gateways, "provide a mechanism to synchronize parallel flow and to create parallel flow" [White et al., 2004] (Figure 37 on the preceding page, right hand side). They are used to show parallel paths in the workflow; even if sometimes not required they might help in understanding the process. Synchronisation allows to start two processes in parallel at the same time in the sequence flow: The sequence flow will continue with *all* processes leaving the parallel gateway. Joining two processes with a parallel gateway is also possible: the process flow will only continue after a signal has arrived from *all* processes coming in the parallel gateway. A parallel gateway is marked by a + in the graphical notation.

Events mark everything happening during the execution of the sequence flow, usually they interrrupt the business process, having some cause or impact on the execution. From the broad range of events that BPMN offers, SED-ML only uses a small subset, namely the start event and the end event (Figure 38).

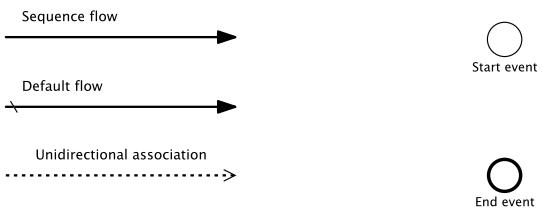

Figure 38: *BPML connectors (left) and events (right).*

All events are graphically drawn as small circles. A *start event* is drawn with a single thin line and mark the start of a process, it can not have any incoming sequence flow. Start event may be triggered by different mechanisms, for the case of SED-ML the untyped start event (no marker inside the circle) is used. The trigger to start the process is "Create new simulation experiment". The *end event* is marked with a thick line. It indicates the end of a process. SED-ML specification makes use of the untyped end event (no marker inside the circle). The end event is used to show the end of sub-processes as well as processes. If the end of a sub-process is reached, the sequence flow returns to the according parent process.

Connectors are used to combine different BPMN objects with each other ([White et al., 2004] page 30 shows the full list of valid connections). SED-ML uses only a subset of available connectors, namely sequence flow, default flow, and unidirectional associations (Figure 38). *Sequence flow* defines the execution order of activities. *Default flow* marks the default branch to be chosen if other conditions leave various possibilities for further execution of the sequence flow. A *unidirectional association* is used to indicate that a data object is modified, i. e. read and written during the execution of an activity [Business Process Technology group, 2009].

The rough SED-ML workflow is shown in Figure 39. The process of defining a SED-ML simulation experiment starts by initialising the experiment and creating a new sed-ml file. Afterwards, the models needed for the simulation are specified and stored into the existing sed-ml file (see section B.1.1). In a third step, the simulation experiment setups are defined and stored into the same file (see section B.1.2). To assign a setup to a number of models used in the experiment, these connections have to be defined and recorded (see section B.1.3), called task in SED-ML.

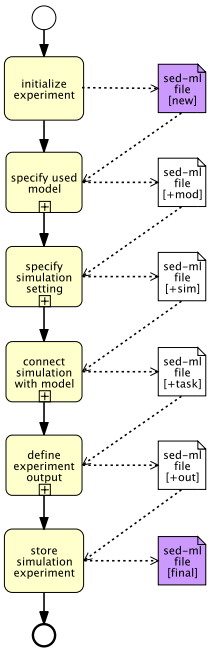

Figure 39: *The process of defining a simulation experiment in SED-ML (overview)*

After simulation, the output should befined, based on the specified tasks and performed simulation experiment. The information is added to the existing SED-ML file (see section B.1.4). In the end, the whole experiment is stored in the final SED-ML file. All collapsed processes are described in the following. Examples in XML are provided in the more technical description

B.1.1 Models

To define a simulation experiment, first a new SED-ML file is created. The models to be used in the experiment (zero or many) are referenced, using a link to a model description in some open, curated model base (such as Biomodels Database Le Novère et al. [2006], CellML Repository Beard et al. [2009], or alike). Changes that are necessary to simulate the model correctly are defined, e. g. assigning new parameter values or updating the mathematics of the model (Figure 40). The procedure is repeated until all models participating in the experiment

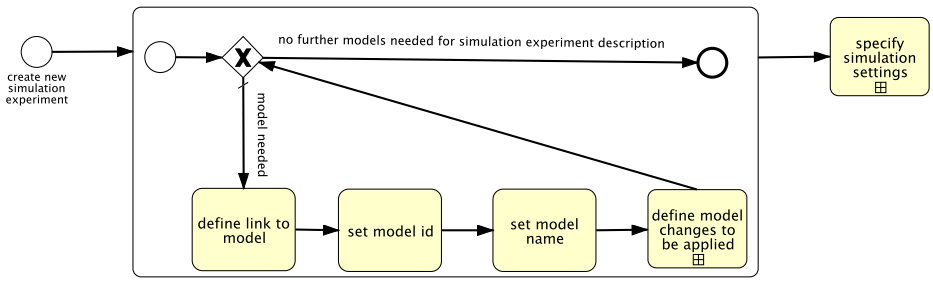

Figure 40: *The process of defining model(s) in SED-ML*

have been described. Each model used gets an internal SED-ML ID and an optional name.

B.1.2 Simulation setup

Secondly, the simulation setups (zero or many) used throughout the simulation experiment are described (Figure 41). Those may stem from various different types of simulation, e. g. steady

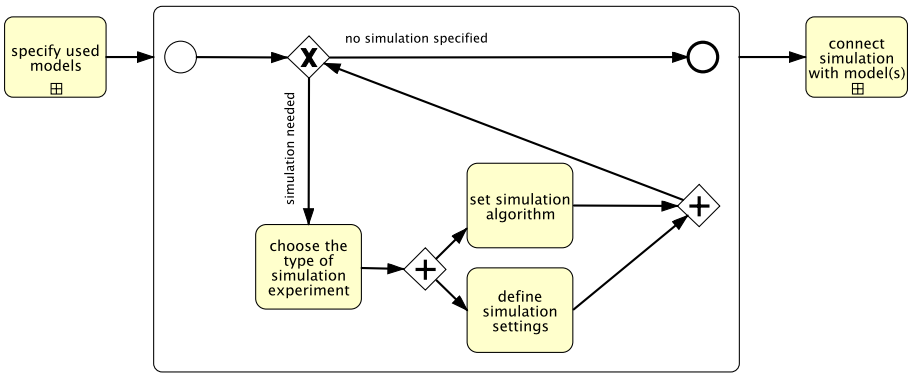

Figure 41: *The process of defining simulation(s) in SED-ML*

state analysis or bifurcation. Depending on the specific type of experiment, the information encoded for the simulation setup might differ. Thus, the definition of simulation settings is specific to the simulation experiment.

In a simple case the experiment consists of one simulation, but it can get far more comlex. For example, one might define a nested sequence of simulations, in which case every simulation has to be defined separately. Each simulation setup gets its own internal ID and an optional name. For each of the setups, the simulation algorithm to be used for that simulation is defined through a reference to a well-defined algorithm name, e. g. an ontology or controlled vocabulary. One approach to define such a controlled vocabulary of simulation algorihtms is the *Kinetic Simulation Algorithm Ontology* (KiSAO, Köhn and Le Novère [2008]). The setup definition is repeated until all different simulations have been described.

B.1.3 Task

SED-ML allows to apply one defined simulation setting to one defined model at a time. However, any number of tasks may be defined inside a simulation experiment description (Figure 42). To do so, each task refers to one of the formerly specified models and to one of the formerly specified simulation setups. Each task has its own ID and an optional name. The process of task definition is repeated until all tasks have been defined.

The current SED-ML does not allow to nest or order tasks. However, these features are evaluated for future versions of SED-ML.

B.1.4 Output

The SED-ML finally consists of output definitions that describe what kind of output the experiment uses to present the simulation result to the user, i. e. a plot or a data table (Figure 43), and also which data is part of the output. Therefore, SED-ML first defines a set of data generators (Figure 44), which are then used to specify a particular result, i. e. output (see section B.1.5).

The SED-ML specification comes with three pre-defined types of outputs: 2D- and 3D plots, and reports. All use the aforementioned data generators to specify the information to be plotted

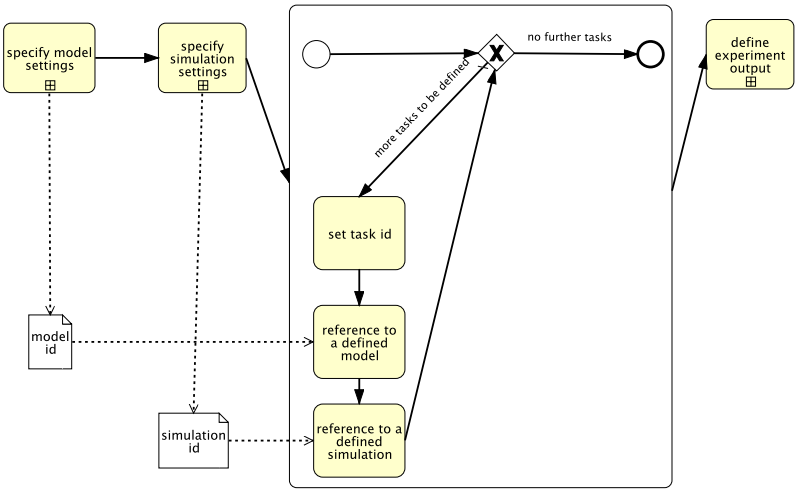

Figure 42: *The process of defining simulation task(s) in SED-ML*

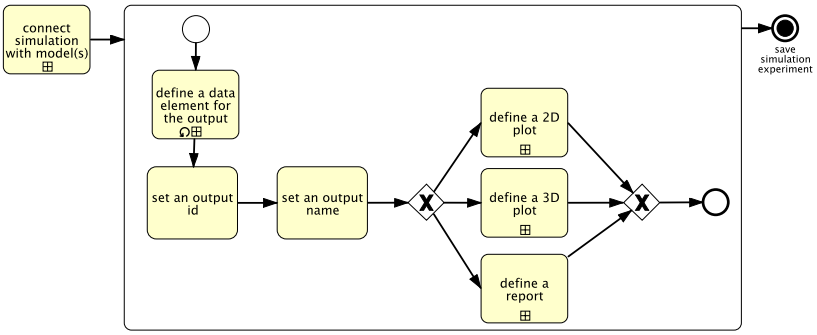

Figure 43: *The process of defining output(s) in SED-ML*

on the different axes, or in the table comlumns respectively.

B.1.5 Data Generator

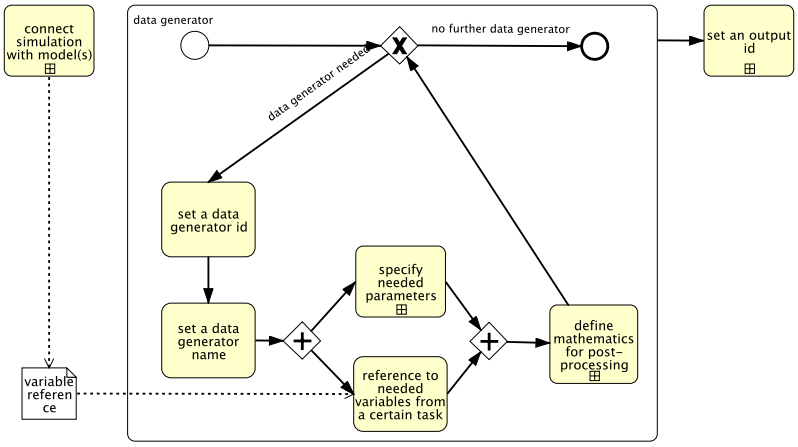

Figure 44: *The process of defining data generator(s) in SED-ML*

A data generator may use data elements, e. g. variables, or parameters, that either (1) have been taken directly from the model, or (2) have been generated in a post-processing step. If post-processing needs to be applied, variables and parameters from the various, previously defined models may be used, but also existing global parameters, such as *time*. If the variables are taken from existing models, a reference to the model and the particular variable needs to be given. If post-processing is necessary, a reference to an existing variable or parameter, including other data generators, has to be provided. Additional mathematical rules to be applied on the referred variable or parameter needs then to be specified. In a SED-ML file, any number of data generators can be created for later re-use in the output definition.

C XML Schema

Listing 43 shows the full SED-ML XML Schema. The code is commented inline.

```
<xs:schema targetNamespace="http://www.biomodels.net/sed-ml" xmlns="http://www.biomodels.net/sed-ml" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:math="http://www.w3.org/1998/Math/MathML">
<!-- mathML import -->
<xs:import namespace="http://www.w3.org/1998/Math/MathML" schemaLocation="sbml-mathml.xsd" />

<!-- global element declarations -->
<!-- variable definition -->
<xs:element name="variable">
 <xs:complexType>
  <xs:attribute name="taskReference" type="xs:string" use="optional" />
  <xs:attribute name="modelReference" type="xs:string" use="optional" />
  <xs:attribute name="name" type="xs:string" use="optional" />
  <xs:attribute name="target" type="xs:token" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- parameter definition -->
<xs:element name="parameter">
 <xs:complexType>
  <xs:attribute name="id" type="xs:string" />
  <xs:attribute name="name" type="xs:string" use="optional" />
  <xs:attribute name="value" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
<!-- general simulation class definition -->
<xs:element name="anySimulation">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="simulationProperty">
    <xs:complexType>
     <xs:attribute name="propertyName" type="xs:string" use="required" />
     <xs:attribute name="propertyValue" type="xs:string" use="required" />
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<!-- uniform time course simulation definition -->
<xs:element name="uniformTimeCourse">
 <xs:complexType>
  <xs:attribute name="id" type="xs:string" />
  <xs:attribute name="outputStartTime" type="xs:double" use="required" />
  <xs:attribute name="outputEndTime" type="xs:double" use="required" />
  <xs:attribute name="numberOfPoints" type="xs:integer" use="required" />
  <xs:attribute name="initialTime" type="xs:double" use="required" />
  <xs:attribute name="algorithm" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- task definition -->
<xs:element name="task">
 <xs:complexType>
  <xs:attribute name="simulationReference" type="xs:string" use="required" />
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="modelReference" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- notes element definition -->
<xs:element name="notes" type="xs:string" />
<!-- annotation element definition -->
<xs:element name="annotation" type="xs:string" />

<!-- complex type SEDBase definition -->
<xs:complexType name="SEDBase">
 <xs:annotation>
  <xs:documentation xml:lang="en">
   The SEDBase type is the base type of all main types in SED-ML. It serves as a container for the annotation of any part of the experiment description.
  </xs:documentation>
```

```
 </xs:annotation>
 <xs:sequence>
  <xs:element ref="notes" minOccurs="0" />
  <xs:element ref="annotation" minOccurs="0" />
 </xs:sequence>
</xs:complexType>

<!-- root element sedML definition -->
<xs:element name="sedML">
 <xs:complexType>
  <xs:complexContent>
   <xs:extension base="SEDBase">
    <xs:sequence>
     <xs:element ref="listOfSimulations" />
     <xs:element ref="listOfModels" />
     <xs:element ref="listOfTasks" />
     <xs:element ref="listOfDataGenerators" />
     <xs:element ref="listOfOutputs" />
    </xs:sequence>
    <xs:attribute name="version" type="xs:decimal" use="required" fixed="0.1" />
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>
</xs:element>
<!-- 2D plot definition -->
<xs:element name="plot2D">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="listOfCurves" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- 3D plot definitio -->
<xs:element name="plot3D">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="listOfSurfaces" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- report definition -->
<xs:element name="report">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="listOfDataSets" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- model element definition -->
<xs:element name="model">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="listOfChanges" minOccurs="0" />
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required" />
  <xs:attribute name="source" use="required" type="xs:string" />
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>

<!--  <xs:element name="math" type="math:Math" /> -->
<!-- list definitions -->
<xs:element name="listOfVariables">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="variable" />
  </xs:sequence>
 </xs:complexType>
```

```
</xs:element>
<xs:element name="listOfParameters">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="parameter" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfTasks">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="task" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfSimulations">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="uniformTimeCourse" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element ref="anySimulation" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfOutputs">
 <xs:complexType>
  <xs:sequence minOccurs="0">
   <xs:element ref="plot2D" minOccurs="0" />
   <xs:element ref="plot3D" minOccurs="0" />
   <xs:element ref="report" minOccurs="0" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfModels">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="model" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfDataGenerators">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="dataGenerator" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfCurves">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="curve" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfSurfaces">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="surface" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfDataSets">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="dataSet" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="listOfChanges">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="changeAttribute" minOccurs="0" maxOccurs="unbounded" />
   <xs:element ref="changeXML" minOccurs="0" maxOccurs="unbounded" />
   <xs:element ref="changeMath" minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
 </xs:complexType>
```

```
</xs:element>
<!-- data generator definition -->
<xs:element name="dataGenerator">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="listOfVariables" minOccurs="0" />
   <xs:element ref="listOfParameters" minOccurs="0" />
   <xs:element ref="math:math" />
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required" />
  <xs:attribute name="id" type="xs:string" />
 </xs:complexType>
</xs:element>
<!-- curve definition (2D plot) -->
<xs:element name="curve">
 <xs:complexType>
  <xs:attribute name="yDataReference" type="xs:string" use="required" />
  <xs:attribute name="xDataReference" type="xs:string" use="required" />
  <xs:attribute name="logY" use="required" type="xs:boolean" />
  <xs:attribute name="logX" use="required" type="xs:boolean" />
 </xs:complexType>
</xs:element>
<!-- surface definition (3D plot)
<xs:element name="surface">
 <xs:complexType>
  <xs:attribute name="yDataReference" type="xs:string" use="required" />
  <xs:attribute name="xDataReference" type="xs:string" use="required" />
  <xs:attribute name="zDataReference" type="xs:string" use="required" />
  <xs:attribute name="logY" use="required" type="xs:boolean" />
  <xs:attribute name="logX" use="required" type="xs:boolean" />
  <xs:attribute name="logZ" use="required" type="xs:boolean" />
 </xs:complexType>
</xs:element>
<!-- data set definition (report) -->
<xs:element name="dataSet">
 <xs:complexType>
  <xs:attribute name="dataReference" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
<!-- definition of attribute changes inside a model -->
<xs:element name="changeAttribute">
 <xs:complexType>
  <xs:attribute name="target" type="xs:token" />
  <xs:attribute name="newValue" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
<!-- definition of changes in the XML structure of the model -->
<xs:element name="changeXML">
 <xs:complexType>
  <xs:attribute name="target" type="xs:token" />
  <xs:attribute name="newValue" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
<!-- definition of changes in the mathematics of the model -->
<xs:element name="changeMath">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="listOfVariables" />
   <xs:element ref="listOfParameters" />
   <xs:element ref="math:math" />
  </xs:sequence>
  <xs:attribute name="target" type="xs:token" />
 </xs:complexType>
</xs:element>
</xs:schema>
```

Listing 43: *SED-ML XML Schema definition*

D Examples

D.1 Le Loup Model (CellML)

The following example provides a SED-ML description for the simulation of the model based on the publication by Leoup, Gonze and Goldbeter "Limit Cycle Models for Circadian Rhythms Based on Transcriptional Regulation in Drosophila and Neurospora" (PubMed ID: 10643740). The model source code is taken from the CellML Model Repository [Lloyd et al., 2008].

The original model used in the simulation experiment is referred to using a URL (http://models.cellml.org/workspace/leloup_gonze_goldbeter_1999/@@rawfile/d6613d7e1051b3eff2bb1d3d419a445bb8c754ad/leloup_gonze_goldbeter_1999_b.cellml, ll. 15-16). In order to st up the model some pre-processing needs to be applied: Those are defined in the `listOfChanges` from ll. 17-25. All changes defined update particular parameter values in the model.

A second model is defined in l. 28 of the example, using `model1` as a source and applying even further changes to it, in this case updating two more model parameters.

One simulation setup is defined in the `listOfSimulations`. It is a `uniformTimeCourse` over 180 time units, using 1000 simulation points. The algorithm used is the CVODE solver, as denoted by the KiSAO ID `KiSAO:0000019`.

A number of `dataGenerator`s are defined in ll. 42-92. Those are the prerequisite for defining the output of the simulation. The first dataGenerator named `tim1` in l. 45 maps on the `Mt` entity in the model that is used in `task1` which here is the model with ID `model1`. The second dataGenerator named `per-tim` in l. 57 maps on the `CN` entity in `model1`. Finally the third and fourth dataGenerators map on the `Mt` and `per-tim` entity respectively in the updated model with ID `model2`.

The `output` defined in the experiment constists of a 2D plot with two different curves (ll. 96-102). Both curves plot the `per-tim` concentration against the `tim` concentration. In the first curve the original parametrisation (as given in `model1`) is used, in the second curve the updated one is used (as given in `model2`).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <sedML version="0.1" xmlns="http://www.biomodels.net/sed-ml" xmlns:math="http://www.w3.org
       /1998/Math/MathML">
3   <!-- textual information about the experiment (optional) -->
4   <notes>Comparing Limit Cycles and strange attractors for oscillation in Drosophila
5   </notes>
6   <!-- definition of simulation setup -->
7   <listOfSimulations>
8    <!-- definition of a uniform time course over 180 time uints using the deterministic
         CVODE solver (KISAO:0000019) -->
9    <uniformTimeCourse id="simulation1" algorithm="KISAO:0000019" initialTime="0"
         outputStartTime="0" outputEndTime="180" numberOfPoints="1000" />
10  </listOfSimulations>
11  <!-- definition of models used during the experiment -->
12  <listOfModels>
13   <!-- reference to a cellML model -->
14   <model id="model1" name="Circadian Oscillations" type="CellML"
15    source="http://models.cellml.org/workspace/leloup_gonze_goldbeter_1999/@@rawfile/
         d6613d7e1051b3eff2bb1d3d419a445bb8c754ad/leloup_gonze_goldbeter_1999_b.cellml" >
16    <!-- definition of changes to be applied to the original model (changing initial
         conditions) -->
17    <listOfChanges>
18     <changeAttribute target="/cellml:model/cellml:component[@cmeta:id='MP']/cellml:variable
           [@name='vsP']/@initial_value" newValue="1"/>
19     <changeAttribute target="/cellml:model/cellml:component[@cmeta:id='MP']/cellml:variable
           [@name='vmP']/@initial_value" newValue="0.7"/>
20     <changeAttribute target="/cellml:model/cellml:component[@cmeta:id='P2']/cellml:variable
           [@name='vdP']/@initial_value" newValue="2"/>
21     <changeAttribute target="/cellml:model/cellml:component[@cmeta:id='T2']/cellml:variable
           [@name='vdT']/@initial_value" newValue="2"/>
22     <changeAttribute target="/cellml:model/cellml:component[@name='parameters']/
           cellml:variable[@name='k1']/@initial_value" newValue="0.6"/>
23     <changeAttribute target="/cellml:model/cellml:component[@name='parameters']/
           cellml:variable[@name='K4P']/@initial_value" newValue="1"/>
24     <changeAttribute target="/cellml:model/cellml:component[@name='parameters']/
           cellml:variable[@name='K4T']/@initial_value" newValue="1"/>
25    </listOfChanges>
26   </model>
27   <!-- reference to the above model (model1) with additional changes of initial values of
         MY and T2 -->
28   <model id="model2" name="Circadian Chaos" type="CellML" source="model1">
29    <listOfChanges>
```

```
30      <changeAttribute target="/cellml:model/cellml:component[@cmeta:id='MT']/cellml:variable
            [@name='vmT']/@initial_value" newValue="0.28"/>
31      <changeAttribute target="/cellml:model/cellml:component[@cmeta:id='T2']/cellml:variable
            [@name='vdT']/@initial_value" newValue="4.8"/>
32     </listOfChanges>
33    </model>
34   </listOfModels>
35   <!-- definition of tasks (combining simulation setup and model) -->
36   <listOfTasks>
37    <!-- limit cycle on model1 -->
38    <task id="task1" name="Limit Cycle" modelReference="model1" simulationReference="
          simulation1"/>
39    <!-- strange attractors on the further perturbated model model2 -->
40    <task id="task2" name="Strange attractors" modelReference="model2" simulationReference="
          simulation1"/>
41   </listOfTasks>
42   <!-- definition of the data generators needed to produce the output -->
43   <listOfDataGenerators>
44    <!-- definition of data generator for tim mRNA -->
45    <dataGenerator id="tim1" name="tim mRNA">
46     <listOfVariables>
47      <variable id="v1" taskReference="task1" target="/cellml:model/cellml:component[
            @cmeta:id='MT']" />
48     </listOfVariables>
49     <math:math>
50      <math:apply>
51       <math:plus />
52       <math:ci>v1</math:ci>
53      </math:apply>
54     </math:math>
55    </dataGenerator>
56    <!-- definition of data generator for the nuclear PER-TIM complex -->
57    <dataGenerator id="per-tim" name="nuclear PER-TIM complex">
58     <listOfVariables>
59      <variable id="v1" taskReference="task1" target="/cellml:model/cellml:component[
            @cmeta:id='CN']" />
60     </listOfVariables>
61     <math:math>
62      <math:apply>
63       <math:plus />
64       <math:ci>v1</math:ci>
65      </math:apply>
66     </math:math>
67    </dataGenerator>
68    <!-- definition of data generator for pertubated tim mRNA -->
69    <dataGenerator id="tim2" name="tim mRNA (changed parameters)">
70     <listOfVariables>
71      <variable id="v2" taskReference="task2" target="/cellml:model/cellml:component[
            @cmeta:id='MT']" />
72     </listOfVariables>
73     <math:math>
74      <math:apply>
75       <math:plus />
76       <math:ci>v2</math:ci>
77      </math:apply>
78     </math:math>
79    </dataGenerator>
80    <!-- definition of data generator for perturbated nuclear PER-TIM complex -->
81    <dataGenerator id="per-tim2" name="nuclear PER-TIM complex">
82     <listOfVariables>
83      <variable id="v1" taskReference="task2" target="/cellml:model/cellml:component[
            @cmeta:id='CN']" />
84     </listOfVariables>
85     <math:math>
86      <math:apply>
87       <math:plus />
88       <math:ci>v1</math:ci>
89      </math:apply>
90     </math:math>
91    </dataGenerator>
92   </listOfDataGenerators>
93   <!-- output definition -->
94    <listOfOutputs>
95     <!-- definition of a 2D plot to show the tim mRNA concentration with different initial
           conditions -->
```

```
96     <plot2D id="plot1" name="tim mRNA with Oscillation and Chaos">
97      <!-- definition of two output curves, both plotting per-tim (original and perturbated)
            against the tim concentration (original and perturbated) -->
98      <listOfCurves>
99       <curve logX="false" logY="false" xDataReference="per-tim" yDataReference="tim1" />
100      <curve logX="false" logY="false" xDataReference="per-tim2" yDataReference="tim2" />
101     </listOfCurves>
102    </plot2D>
103   </listOfOutputs>
104 </sedML>
```

Listing 44: *LeLoup Model Simulation Description in SED-ML*

References

D. A. Beard, R. Britten, M. T. Cooling, A. Garny, M. D. Halstead, P. J. Hunter, J. Lawson, C. M. Lloyd, J. Marsh, A. Miller, D. P. Nickerson, P. M. Nielsen, T. Nomura, S. Subramanium, S. M. Wimalaratne, and T. Yu. Cellml metadata standards, associated tools and repositories. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 367 (1895):1845–1867, May 2009. ISSN 1364-503X. doi: 10.1098/rsta.2008.0310. URL http://dx.doi.org/10.1098/rsta.2008.0310.

Donald Bell. Uml basics, part iii: The class diagram. IBM, the rational edge, 2003. http://download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/nov03/t_modelinguml_db.pdf.

T Berners-Lee, R Fielding, and L Masinter. Uniform resource identifier (URI): Generic syntax, 2005. URL http://www.ietf.org/rfc/rfc3986.txt.

T Bray, J Paoli, CM Sperberg-McQueen, E Maler, F Yergeau, and J Cowan. Extensible markup language (XML) 1.1 (second edition), 2006. URL http://www.w3.org/TR/xml11/.

Business Process Technology group. Bpmn – business process modeling notation. poster, 2009. URL http://bpt.hpi.uni-potsdam.de/Public/BPMNCorner.

D. Carlisle, P. Ion, R. Miner, and N. Poppelier. Mathematical markup language (mathml) version 2.0. *W3C Recommendation*, 21, 2001.

Mélanie Courtot, Dagmar Waltemath, Christian Knüpfer, et al. Controlled vocabularies and semantics in systems biology. to be submitted.

MB Elowitz and S Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, January 2000.

D.C. Fallside, P. Walmsley, et al. XML schema part 0: Primer. *W3C recommendation*, 2, 2001.

N Goddard, M Hucka, F Howell, H Cornelis, K Skankar, and D Beeman. Towards neuroml: Model description methods for collaborative modeling in neuroscience. *Phil. Trans. Royal Society series B*, 356:1209–1228, 2001.

Stefan Hoops, Sven Sahle, Christine Lee, Jurgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. COPASI - a COmplex PAthway SImulator. *Bioinformatics (Oxford, England)*, 22(24):3067–3074, December 2006. ISSN 1460-2059. doi: 10.1093/bioinformatics/btl485. URL http://dx.doi.org/10.1093/bioinformatics/btl485.

M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, March 2003. ISSN 1367-4803. doi: 10.1093/bioinformatics/btg015. URL http://dx.doi.org/10.1093/bioinformatics/btg015.

Michael Hucka, Michael Hucka, Frank T. Bergmann, Stefan Hoops, Sarah Keating, Sven Sahle, and Darren J. Wilkinson. The systems biology markup language (sbml): Language specification for level 3 version 1 core (release 1 candidate). *Nature Precedings*, January 2010. ISSN 1756-0357. doi: 10.1038/npre.2010.4123.1. URL http://dx.doi.org/10.1038/npre.2010.4123.1.

D. Köhn and N. Le Novère. The KInetic Simulation Algorithm Ontology (KiSAO)-A Proposal for the Classification of Simulation Algorithms in Systems Biology. Eingeladener Vortrag, 2008.

N. Le Novère, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J. L. Snoep, and M. Hucka. Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res*, 34(Database issue), January 2006. ISSN 1362-4962. URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=pubmed&dopt=Abstract&list_uids=16381960.

Catherine M. Lloyd, Matt D. B. Halstead, and Poul F. Nielsen. Cellml: its future, present and past. *Prog Biophys Mol Biol*, 85:433–450, 2004.

C.M. Lloyd, J.R. Lawson, P.J. Hunter, and P.F. Nielsen. The CellML model repository. *Bioinformatics*, 24(18):2122, 2008.

OMG. *UML 2.2 Superstructure and Infrastructure*, February 2009. URL http://www.omg.org/spec/UML/2.2/.

S. Pemberton et al. XHTML 1.0: The Extensible HyperText Markup Language—W3C Recommendation 26 January 2000. *World Wide Web Consortium (W3C)(August 2002)*, 2002.

W3C. Xml schema part 1: Structures second edition. W3C Recommendation, October 2004. URL http://www.w3.org/TR/xmlschema-1/.

D. Waltemath, R. Adams, D.A. Beard, F.T. Bergmann, U.S. Bhalla, R. Britten, V. Chelliah, M.T. Cooling, J. Cooper, E. Crampin, A. Garny, S. Hoops, M. Hucka, P. Hunter, E. Klipp, C. Laibe, A. Miller, i. Moraru, D. Nickerson, P. Nielsen, M. Nikolski, S. Sahle, H. Sauro, H. Schmidt, J.L. Snoep, D. Tolle, O. Wolkenhauer, and N. Le Novère. Minimum information about a simulation experiment (miase). *submitted to PLoS Comput Biol*.

S.A. White et al. Business process modeling notation (BPMN) version 1.0. *Business Process Management Initiative, BPMI. org*, 2004.