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

September 17, 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.

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.

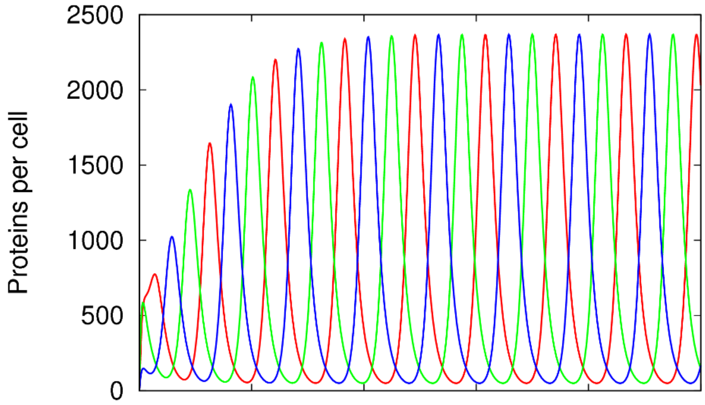

Figure 1: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI. The number of repressor proteins lacI, tetR and cI is shown. (taken from [Waltemath et al.])*

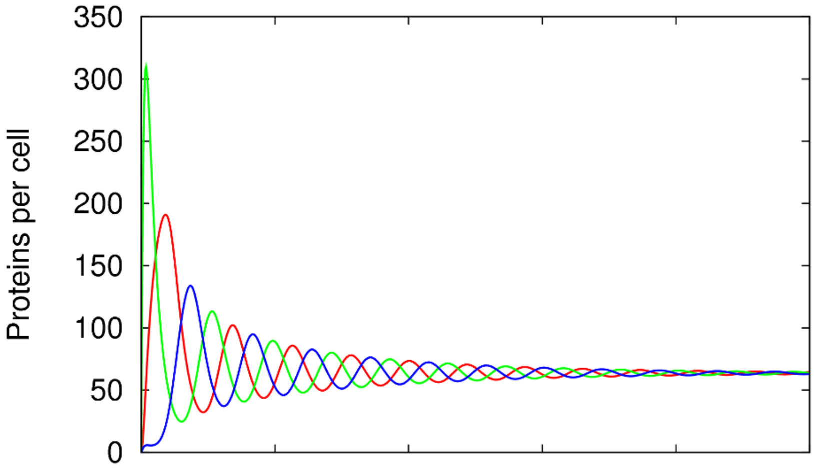

Figure 2: *Time-course simulation of the repressilator model, imported from BioModels Databas and simulated in COPASI after modification of the initial values of the* protein copies per promoter *and the* leakiness in protein copies per promoter. *The number of repressor proteins lacI, tetR and cI is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

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 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).

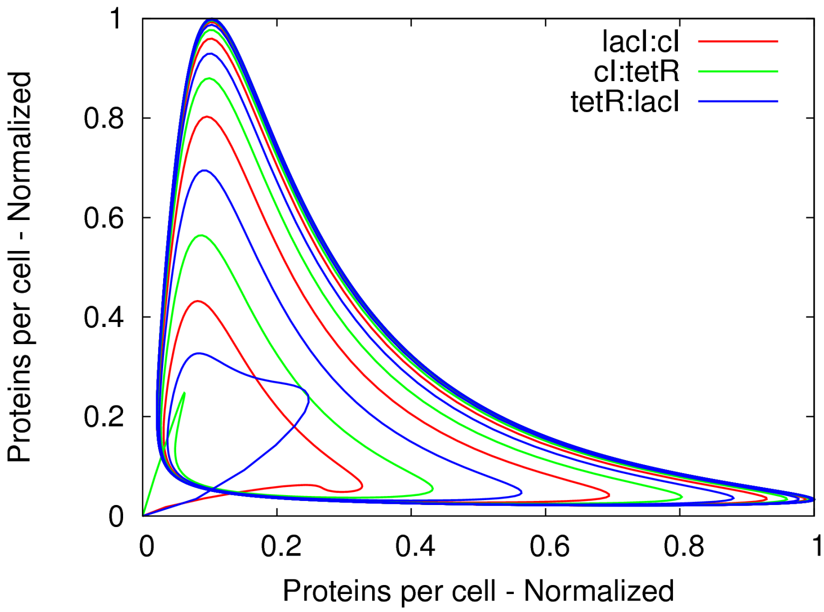

Figure 3: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI, showing the normalized temporal evolution of repressor proteins lacI, tetR and cI in phase-plane. (taken from [Waltemath et al.])*

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 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`.

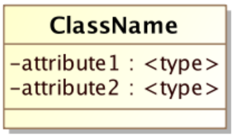

Figure 4: *SED-ML UML Class with class names and attributes*

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*

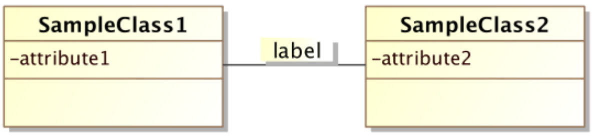

Figure 5: *UML Class connectors*

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) 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).

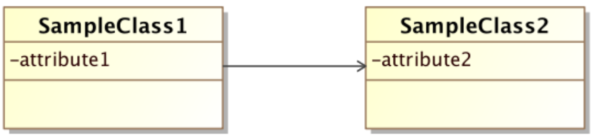

Figure 6: *UML Association*

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.

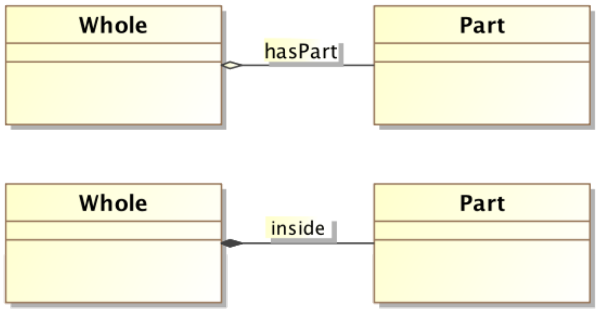

Figure 7: *UML Aggregation*

The composite aggregation (Figure 7, 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) allows to extend classes (`BaseClass`) by additional properties. The derived class (`DerivedClass`) inherites all properties of the base class and defines additional ones.

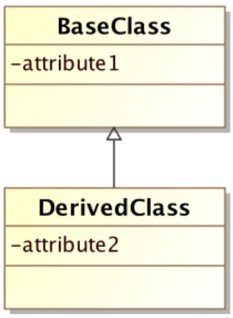

Figure 8: *UML Generalisation*

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 expresses that a class is given by a professor, and a professor might give one to many classes.

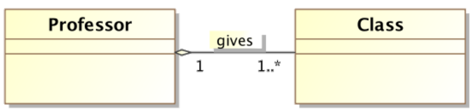

Figure 9: *UML Multiplicity in an Aggregation*

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:

```
<xs:element name="computeChange">
 <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>
```

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

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.

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.

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 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. Finally, annotations of SED-ML elements should be provided with a standardised annotation scheme.

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 adopted from 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 [Li et al., 2010]. 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 recommends to follow the above scheme for model references, if possible. 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 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 this issue 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

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

2.2.4 Annotations

When annotating SED-ML elements with semantic annotations, the MIRIAM URI Scheme should be used. In addition to providing the data type (e. g. PubMed) and the particular data entry inside that data type (e. g. `10415827`), the relation of the annotation to the annotated element should be described using the standardised biomodels.net qualifier. The list of qualifiers, as well as further information about their usage, is available from `http://www.biomodels.net/qualifiers/`.

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 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 declares the namespace for the SED-ML document. The pre-defined namespace for SED-ML documents 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 the encoding of mathematical expressions in MathML 2.0. SED-ML uses a subset of MathML as described in section 2.1 on page 10.

SED-ML notes use the **xmlns** of XHTML `http://www.w3.org/1999/xhtml`. The Notes class is described in section 3.4.2 on page 14.

Additional external namespaces might be used in Annotations.

3.2 The `id` attribute

Most objects in SED-ML carry an id attribute. The id attribute, if existent for an object, is always required and identifies SED-ML constituents unambiguously. It is used to refer to a constituent from other constituents. The **id** data type is **String**. All **id**s have a global scope, i. e. the **id** must be unambiguous throughout a whole SED-ML document. As such it identifies the constituent it is related to. An example for a defined **id** is given in listing 4.

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

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

The defined model carries the ID **m00001**. If the model is used somewhere else in the SED-ML document, it is referred to by that ID.

3.3 The `name` attribute

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

Listing 5 extends the model definition in listing 4 by a model name.

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

Listing 5: *SED-ML name definition, e. g. for a model*

3.4 The SEDBase Class

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

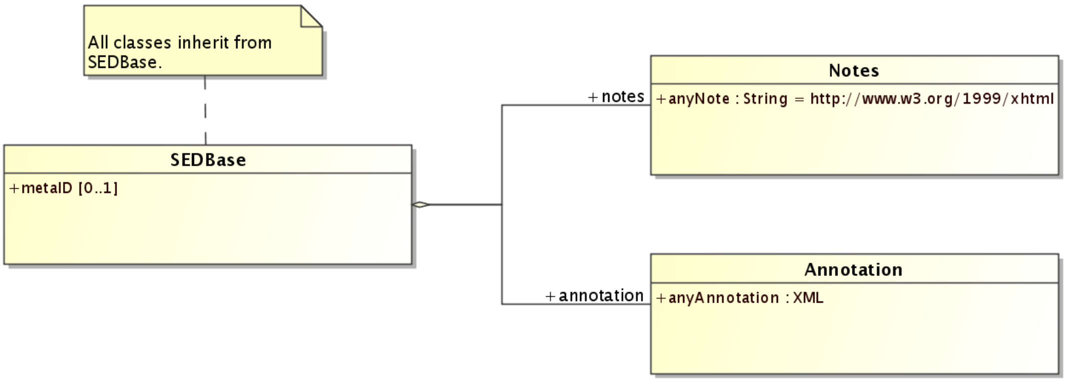

Figure 10: *The SEDBase class*

Table 3.4 shows all attributes and sub-elements for the SEDBase element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	**description**
metaID^o	page 14
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

3.4.1 *`metaid` Attribute*

The main purpose of the `metaid` attribute is to attach semantic annotations in form of the Annotation class to SED-ML elements. The type of `metaid` is XML ID and as such the `metaid` attribute is globally unique throughout the whole SED-ML document.

For an example showing how to link a semantic annotation to a SED-ML object via the `metaid` is given in the Annotation description.

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. Instances of the Notes class may contain any valid XHTML [Pemberton et al., 2002], ranging from short comments to whole HTML pages for display in a Web browser. 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).

`Notes` has a mandatory attribute xmlns to declare the `XHTML` namespace. It does not have any further sub-elements nor attributes associated to it.

Listing 6 shows the use of the `notes` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<sedML [..]>
 <notes http://www.w3.org/1999/xhtml>
```

```
  The enclosed simulation description shows the oscillating behaviour of the Repressilator model using
       deterministic and stochastic simulators.
 </notes>
</sedML>
```

Listing 6: *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. 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). The style of annotations in SED-ML is briefly described in section 2.2.4 on page 12.

Table 3.4.3 shows all attributes and sub-elements for the Annotation element as defined by the SED-ML Level 1 Version 1 XML Schema.

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

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

Listing 7 shows the use of the `annotation` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<sedML>
  [..]
  <model id="model1" metaID="001" 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 7: *The annotation element*

In that example, a SED-ML model element is annotated with a reference to the original publication. The `model` contains an `annotation` that uses the biomodels.net 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 which defines the document's structure and content (Figure 11). A SED-ML document needs to have the SED-ML namespace defined

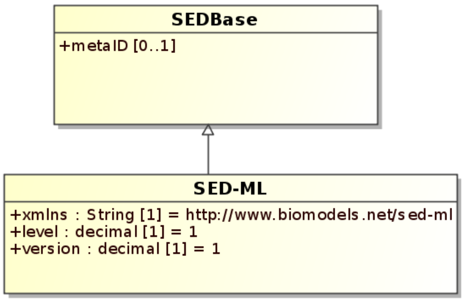

Figure 11: *The SED-ML class*

through the mandatory xmlns attribute. In addition, the SED-ML level and version attributes are mandatory.

The SED-ML document consists 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 sub-elements for the SED-ML element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaID^o	page 14
xmlns	page 13
level	page 17
version	page 17
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
model^o	page 27
simulation^o	page 33
task^o	page 35
dataGenerator^o	page 36
output^o	page 38

Table 3: *Attributes and nested elements for SED-ML.* elements^o *denotes optional elements.*

The basic XML structure of a SED-ML file is shown in listing 8.

```
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns:math="http://www.w3.org/1998/Math/MathML"
       xmlns="http://www.biomodels.net/sed-ml" level="1" version="1">
 <listOfModels />
  [MODEL REFERENCES AND APPLIED CHANGES]
 <listOfSimulations />
  [SIMULATION SETUPS]
 <listOfTasks />
  [MODELS LINKED TO SIMULATIONS]
 <listOfDataGenerators />
  [DEFINITION OF POST-PROCESSING]
 <listOfOutputs />
  [DEFINITION OF OUTPUT]
```

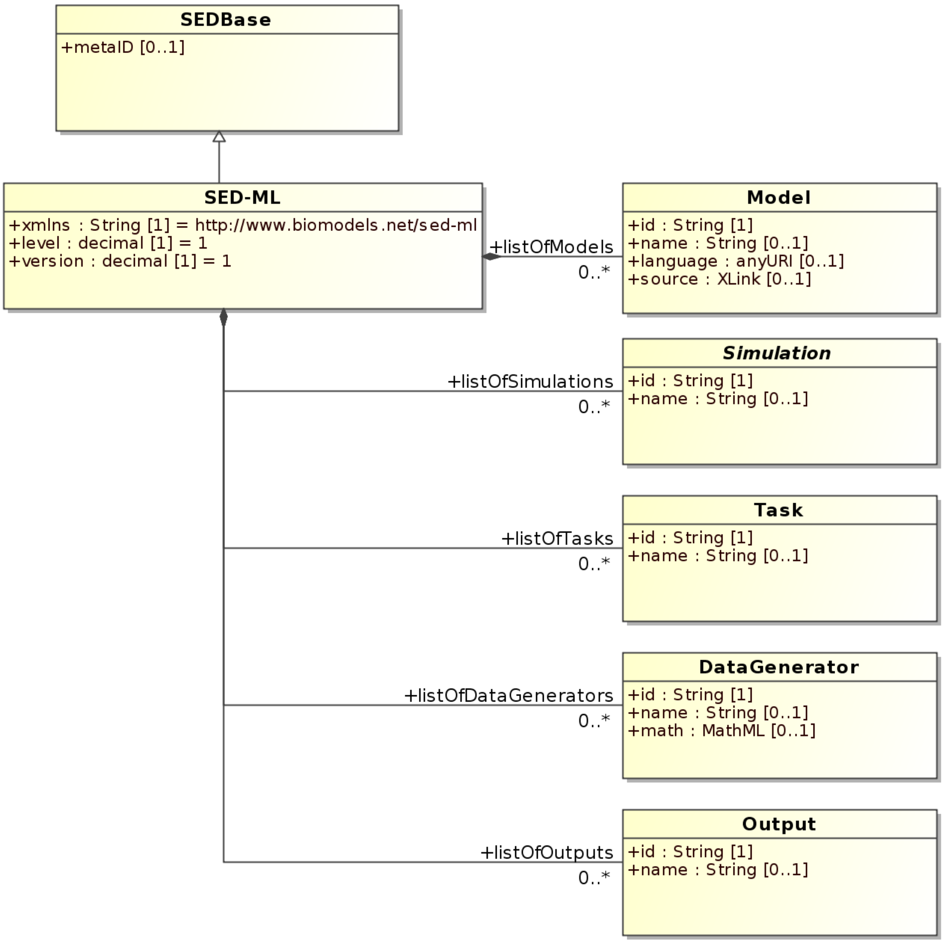

Figure 12: *The sub-classes of SED-ML*

```
14 </sedML>
```

Listing 8: *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 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 definitions, and listOfOutputs for all output definitions).

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, as shown in the example in listing 8.

3.5.2 The `version` attribute

The current SED-ML version is *version 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, as shown in the example in listing 8.

3.6 The reference Relations

The reference concept is used to refer to a particular element inside the SED-ML document. 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 a Task and the associated Model (modelRereference) or
4. as an association between a Task and the Simulation (simulationReference)

Depending on the use of the reference relation in connection with a Variable object, it may take different roles:

a. The reference association might occur between a Variable object and a Model object, if the variable is to define a Change. In that case the **variable** element contains a modelReference to refer to the particular model that contains the variable used to define the change (see section 3.6.1 on page 18).

b. If the reference is used as an association between a Variable object and a Task object inside the dataGenerator class, then the **variable** element contains a taskReference to unambiguously refer to an observable in a given task (see section 3.6.2 on page 19).

The definition of a Task object demands a reference to a particular Model object (modelReference, see 3.6.1 on page 18); furthermore, the Task object must be associated with a particular Simulation object (simulationReference, see 3.6.3 on page 19).

3.6.1 model Reference

The modelReference represents a relation between a Variable object and a Model object, or a relation between a Task object and a Model object.

If pre-processing needs to be applied to a model before simulation, then the model update can be specified by creating a Change object. In the particular case that a change must be calculated with a mathematical function, variables need to be defined. To refer to an existing entity in a defined Model, the modelReference is used.

The `modelReference` attribute of the `variable` element contains the id of a model that is defined in the document.. Listing 9 shows the use of the `modelReference` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<model id="m0001" [..]>
 <listOfChanges>
   <computeChange>
    <listOfVariables>
     <variable id="v1" modelReference="cellML" target="/cellml:model/cellml:component[@cmeta:id='MP']/
          cellml:variable[@name='vsP']/@initial_value" />
    </listOfVariables>
    <listOfParameters [..] />
    <math>
     [CALCULATION OF CHANGE]
    </math>
   </computeChange>
 </listOfChanges>
 [..]
</model>
```

Listing 9: *SED-ML* `modelReference` *attribute inside a variable definition of a* `computeChange` *element*

In the example, a change is applied on 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 10 shows how this can be done for a sample SED-ML document.

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

Listing 10: *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

DataGenerator objects are created to apply post-processing to the simulation results before simulation output.

For 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 11 gives an example.

```
<listOfDataGenerators>
 <dataGenerator id="tim3" name="tim mRNA (difference v1-v2+20)">
  <listOfVariables>
   <variable id="v1" taskReference="t1" [..] />
  </listOfVariables>
  <math />
 </dataGenerator>
</listOfDataGenerators>
```

Listing 11: *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:

```
<listOfTasks>
  <task id="t1" name="task definition" modelReference="model1"
        simulationReference="simulation1" />
</listOfTasks>
```

Task `t1` references the model `model1`. Therefore we can conclude that the variable `v1` defined in listing 11 targets an element of the model with ID `model1`. The targeting process itself will be explained in section 3.7.1 on page 21.

3.6.3 simulationReference

The simulationReference is used to refer to a particular Simulation in a Task. Listing 10 on page 19 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

In SED-ML variables are references to already existing entities, either existing in one of the defined models or externally defined symbols (Figure 13). If the variable is defined through a reference to a

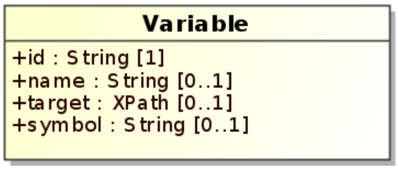

Figure 13: *The Variable class*

model constituent, such as an SBML species, then the reference is specified using the target attribute. If the variable is defined through a reference to an external entity, then the symbol attribute is used. It holds a SED-ML URI. A `variable` is always placed inside a listOfVariables. `Symbol` and `target` must not be used together in a single instance of Variable.

Table 3.7 shows all attributes and sub-elements for the Variable element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
target	page 21
symbol	page 21
taskReference	page 19
modelReference	page 18
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

Table 4: *Attributes and nested elements for Variable.* elements^o *denotes optional elements.*

A `variable` element must contain a taskReference if it occurs inside a `listOfVariables` inside a dataGenerator element. A `variable` element must contain a modelReference if it occurs inside a `listOfVariables` inside a computeChange element.

Listing 12 shows the use of the `variable` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<sedML>
 <listOfModels>
  <model [..]>
   <listOfChanges>
    <computeChange target="TARGET ELEMENT OR ATTRIBUTE">
     <listOfVariables>
       <variable id="v1" name="maximum velocity"
        target="XPath TO A MODEL ELEMENT OR ATTRIBUTE IN ANY SPECIFIED MODEL" />
      [FURTHER VARIABLE DEFINITIONS]
     </listOfVariables>
     [..]
    </computeChange>
   </listOfChanges>
   [..]
  </model>
  [..]
 </listOfModels>
 <listOfDataGenerators>
  <dataGenerator [..]>
   <listOfVariables>
    <variable id="v2" name="time" symbol="urn:sedml:symbol:time" />
      [FURTHER VARIABLE DEFINITIONS]
   </listOfVariables>
```

```
24   </dataGenerator>
25  </listOfDataGenerators>
26  [..]
27 </sedML>
```

Listing 12: *SED-ML* `variable` *definitions inside the* `computeChange` *element and inside the* `dataGenerator` *element*

Listing 12 defines a variable `v1` (line 7) to compute a change on a model constituent (referenced by the `target` attribute on `computeChange` in line 5). The value of `v1` corresponds with the value of the targeted model constituent references by the `target` attribute in line 8. The second variable, `v2` (line 21), is used inside a `dataGenerator`. As the variable is time, the `symbol` attribute is used to refer to the SED-ML URI for time (line 21).

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.

Listing 13 shows the use of the `target` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 13: *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 12, 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 12, but with different naming:

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

Listing 14: *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 15: *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 11.

Listing 16 shows the use of the `symbol` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 16: *SED-ML* `symbol` *definition*

3.8 The Parameter Class

The SED-ML Parameter class creates instances with a constant value (Figure 14). SED-ML uses param-

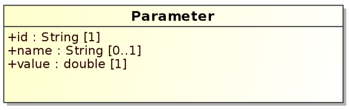

Figure 14: *The Parameter class*

eters in two ways: Firstly, parameters may be defined in the ComputeChange class for describing the mathematical computation of a change of a model's observable. Secondly, parameters may be part of a DataGenerator specification. In both cases the parameter definitions are local to the particular class defining them.

Table 3.8 shows all attributes and sub-elements for the parameter element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaID^o	page 14
id	page 13
name^o	page 13
value	page 22
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

A parameter can unambiguously be identified through it's given id. It may additionally carry an optional name. Each parameter has one associated value.

Listing 17 shows the use of the `parameter` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 17: *The definition of a parameter in SED-ML*

The listing shows the definition of a parameter `p1` with the `value="40"` assigned.

3.8.1 The `value` attribute

Each parameter defines a particular thing with a exactly one fixed value. The `value` attribute of XML data type `Double` is required for each `parameter` element.

3.9 The ListOf containers

SED-ML listOf* elements 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.

3.9.1 listOfVariables: The variable definition container

SED-ML uses the variable concept to refer to existing entities inside a model. The container for all variables is listOfVariable (Figure 15). It includes all variables that need to be defined to either describe a change in the model by means of mathematical equations (ComputeChange) or to set up a dataGeneratorClass.

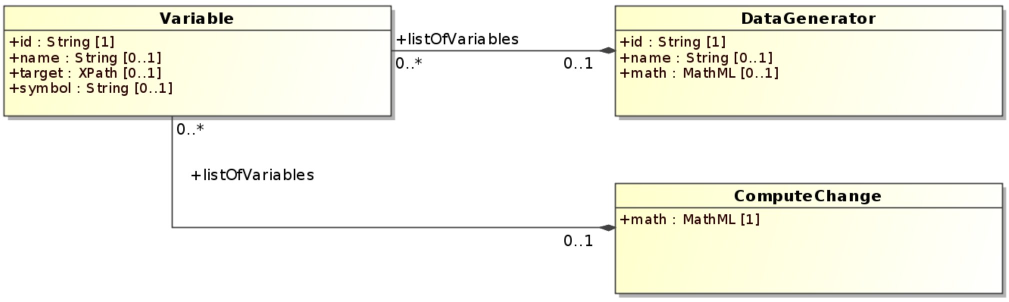

Figure 15: *The SED-ML listOfVariables container*

Listing 18 shows the use of the `listOfVariables` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 18: *SED-ML listOfVariables element*

The `listOfVariables` is optional and may contain zero to many models.

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).

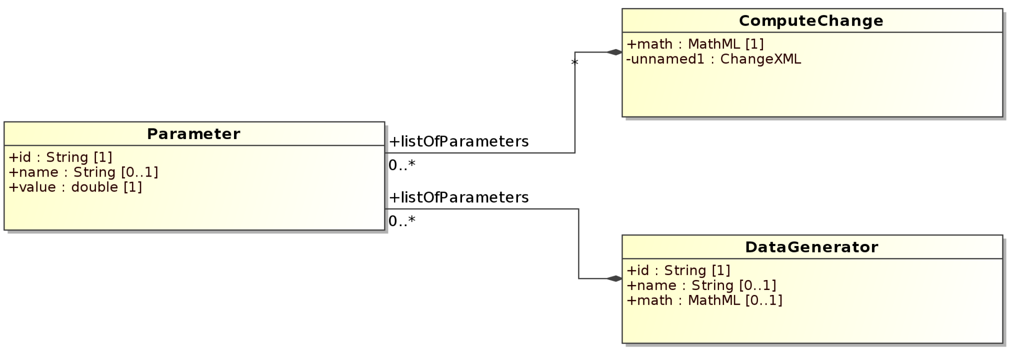

Figure 16: *The SED-ML `listOfParameters` container*

Listing 19 shows the use of the `listOfParameters` element in a SED-ML file as defined by the SED-ML

Level 1 Version 1 XML Schema. The element is optional and may contain zero to many parameters.

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

Listing 19: *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 listOfModels container for all necessary models (Figure 17).

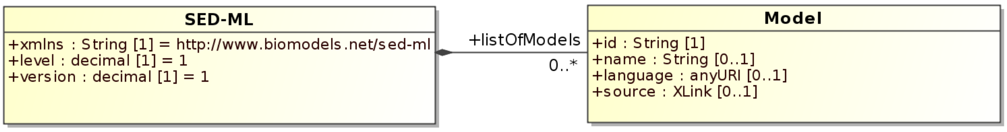

Figure 17: *The SED-ML listOfModels container*

Listing 20 shows the use of the `listOfModels` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema. The `listOfModels` is optional and may contain zero to many models.

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

Listing 20: *SED-ML listOfModels element*

3.9.4 listOfChanges: The change definition container

The listOfChanges contains the defined changes to be applied to a particular model (Figure 18). It

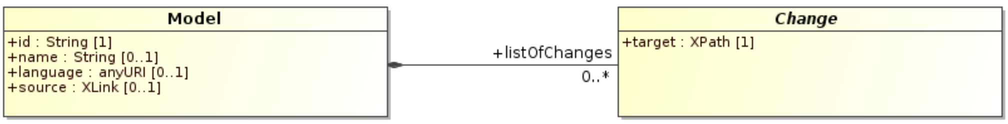

Figure 18: *The SED-ML listOfChanges container*

always occurs as an optional subelement of the `model` element.

Listing 21 shows the use of the `listOfChanges` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema. The `listOfChanges` is nested inside the `model` element.

```
<model id="m0001" [..]>
 <listOfChanges>
  [CHANGE DEFINITION]
 </listOfChanges>
</model>
```

Listing 21: *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, each stored in a single `change` element, 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 (Figure 19 on the following page).

Listing 22 shows the use of the `listOfSimulation` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

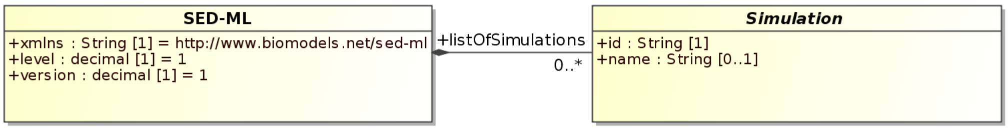

Figure 19: *The listOfSimulations container*

```
<listOfSimulations>
 <simulation id="s1" [..]>
  [UNIFORM TIMECOURSE DEFINITION]
 </simulation>
 <simulation id="s2" [..]>
  [UNIFORM TIMECOURSE DEFINITION]
 </simulation>
</listOfSimulations>
```

Listing 22: *The SED-ML* `listOfSimulations` *element, containing two simulation setups*

Listing 22 shows the definition of two simulation setups, each of them encoded in a single simulation element. 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

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

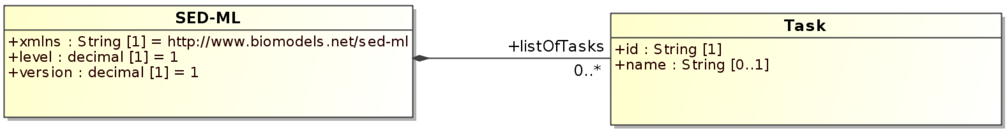

Figure 20: *The SED-ML listOfTasks container*

Listing ?? shows the use of the `listOfTasks` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<listOfTasks>
 <task id="t1" name="simulating v1" modelReference="m1" simulationReference="s1">
 [FURTHER TASK DEFINITIONS]
</listOfTasks>
```

Listing 23: *The SED-ML* `listOfTasks` *element, defining one task*

The example shows the definition of one task `t1` in the `listOfTasks` container.

3.9.7 listOfDataGenerators: The post-processing container

In SED-ML, all variable- and parameter values that shall be used in the Output class need to be defined as a dataGenerator beforehand. The container for those data generators is the listofDataGenerators (Figure 21).

Figure 21: *The SED-ML listOfDataGenerators container*

Listing 36 shows the use of the `listOfDataGenerators` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<listOfDatGenerators>
 <dataGenerator id="d1" name="time">
  [DATA GENERATOR DEFINITION FOLLOWING]
 </dataGenerator>
 <dataGenerator id="LaCI" name="LaCI repressor">
  [DATA GENERATOR DEFINITION FOLLOWING]
 </dataGenerator>
</listOfDataGenerators>
```

Listing 24: *The* `listOfDataGenerators` *element, defining two data generators* time *and* LaCI repressor

3.9.8 *listOfOutputs: The output specification container*

The listOfOutputs container holds the output specifications for a simulation experiment.

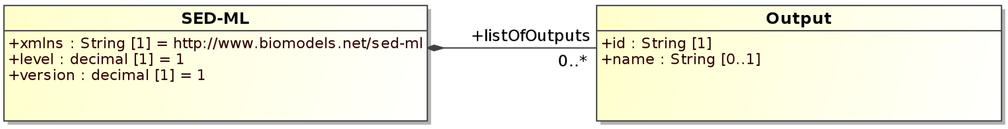

Figure 22: *The SED-ML listOfOutputs container*

The output can be defined as either a report, a plot2D or as a 3D plot.

Listing 25 shows the use of the `listOfOutputs` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<listOfOutputs>
 <report id="report1">
  [REPORT DEFINITION FOLLOWING]
 </report>
 <plot2D id="plot1">
  [2D PLOT DEFINITION FOLLOWING]
 </plot2D>
</listOfOutputs>
```

Listing 25: *The* `listOfOutput` *element*

The example shows the definition of two different outputs, one being a data table (report), the other one a 2D plot.

4 SED-ML Components

In this section we describe the major components of SED-ML. In the following we use the UML notation presented in section 1.2.1. In addition, we provide a detailed BNMP diagram with explanation of the SED-ML workflow in Appendix B and an XML Schema in C.

4.1 Model

The Model defines the models to be used in the simulation experiment (Figure 23).

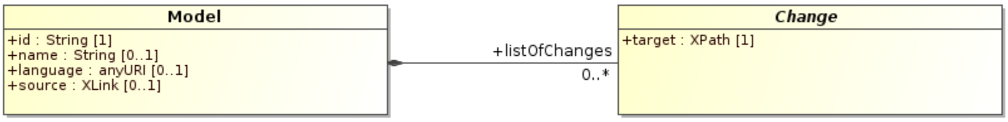

Figure 23: *The SED-ML Model class*

Each instance of the Model class has an unambiguous and mandatory id. An additional, optional name may be given to the model.

The language may be specified, defining the format the model is encoded in, if such a format exists. Example formats are SBML or CellML.

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 (language attribute).
- To refer to a particular model in an external resource, an unambiguous reference must be given (source attribute).

Table 4.1 shows all attributes and sub-elements for the model element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
language^o	page 28
source	page 28
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
change^o	page 29

Table 6: *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.

Listing 26 shows the use of the `model` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<listOfModels>
 <model id="m0001" language="urn:sedml:language:sbml"
  source="urn:miriam:biomodels.db:BIOMD0000000012">
  <listOfChanges>
```

```
5     <change>
6      [MODEL PRE-PROCESSING]
7     </change>
8     </listOfChanges>
9   </model>
10  <model id="m0002" language="urn:sedml:language:sbml" source="m0001">
11   [MODEL PRE-PROCESSING]
12  </model>
13  <model id="m0003" language="urn:sedml:language:cellml" source="http://www.cellml.org/models/
        leloup_gonze_goldbeter_1999_version02">
14   [MODEL PRE-PROCESSING]
15  </model>
16 </listOfModels>
```

Listing 26: *SED-ML* **`model`** *element*

The above `listOfModels` contains three models: The first model `m0001` is the Repressilator model taken from BioModels Database. The original model is available from `urn:miriam:biomodels.db:BIOMD0000000012`. For the SED-ML simulation, the model might undergo pre-processings, described in the change element (lines 5-7). Based on the description of the first model `m0001`, the second model is build. It refers to the model `m001` in the `source` attribute, that is the modified version of the Repressilator model. `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 (lines 13-15) is a different model in CellML representation. `m0003` is the model available from the given URL in the `source` attribute. Again, it might have additional pre-processing applied to it before used in the simulation.

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. 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. SED-ML also defines a set of standard URIs to refer to particular language definitions (see again section 2.2.2 on page 11). 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`. To specify language and version, following the idea of MIRIAM URNs, the SED-ML URN scheme `urn:sedml:language:`*language name* is used. A model's language being "SBML Level 2 Version 2" can be referred to, for example, through the URN `urn:sedml:language:sbml.level-2.version-2`.

The `language` attribute is optional in the XML representation of a SED-ML file. If it is not explicitely defined 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 11.

4.1.2 The `source` attribute

To make the model available for the execution of a SED-ML file, the model `source` must 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. References to curated, open model bases are recommended, such as the BioModels Database. However, any resource registered with MIRIAM resources[2] can easily be referenced. Even without a MIRIAM URN, SED-ML can be used (see again section 2.2.1 on page 11).

An example for the definition of a model, and using the URI scheme is given in listing 27.

```
<model id="m1" name="repressilator" language="urn:sedml:language:sbml"
 source="urn:miriam:biomodels.db:BIOMD0000000012">
```

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

```
  <listOfChanges>
   [DEFINE MODEL PRE-PROCESSING HERE]
  </listOfChanges>
</model>
```

Listing 27: *The SED-ML* `source` *element, using the URI scheme*

The example defines one model `m1`. `urn:miriam:biomodels.db:BIOMD0000000012` defines the source of the model code. The MIRIAM URN can be resolved into the SBML model stored in BioModels Database under ID `BIOMD0000000012`.

An example for the definition of a model and using a URL is given in listing 28.

```
<model id="m1" name="repressilator" language="urn:sedml:language:cellml"
  source="http://models.cellml.org/exposure/bba4e39f2c7ba8af51fd045463e7bdd3/aguda_b_1999.cellml">
  <listOfChanges />
</model>
```

Listing 28: *The SED-ML* `source` *element, using a URL*

In the example one model is defined. The `language` of the model is `CellML`. As the CellML model repository currently does not provide a MIRIAM URI 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 (Figure 24). Changes can be of three different types:

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. Changes based on mathematical calculations (ComputeChange)

The Change class is abstract and serves as the container for the different types of changes. Therefore, a SED-ML document will only contain the derived classes, i. e. ChangeAttribute, ChangeXML, or ComputeChange.

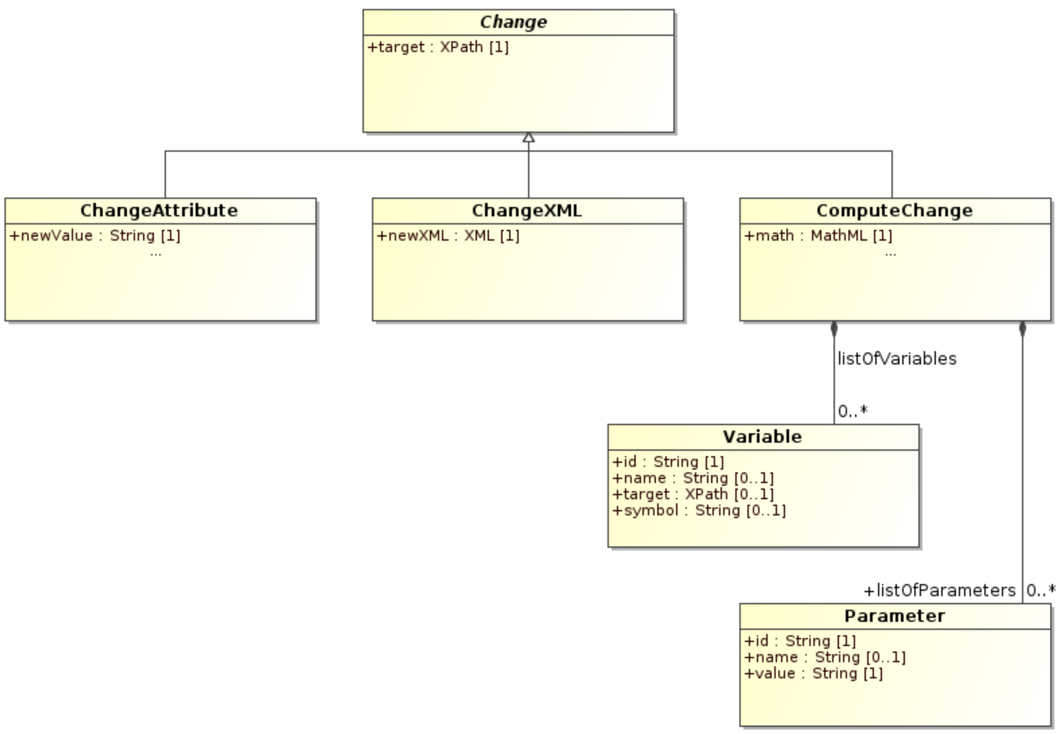

Figure 24: *The SED-ML Change class*

Table 4.2 shows all attributes and sub-elements for the change element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
target	page 21
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
changeXML^o	page 30
changeAttribute^o	page 31
computeChange^o	page 31

Table 7: *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.

4.2.1 The ChangeXML Class

The ChangeXML class defines changes of any XML element in the model that can be addressed by a valid XPath expression. The XPath is specified in the required target attribute (see again section 3.7.1 on page page 21).

Table 4.2.1 shows all attributes and sub-elements for the changeXml element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
target	page 21
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
newXML	page 30

Table 8: *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 29)*

4.2.2 The newXML element

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

An example for a change that adds an additional parameter to a model is given in listing 29.

```
<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"> />
  </newXML>
 </listOfChanges>
</model>
```

Listing 29: *The* **changeXML** *element with its* **newXML** *sub-element*

The code of the model is changed in the way that its parameter with ID `V_mT` is substituted by another two parameters `V_mT` and `V_mT_2`.

4.2.3 The ChangeAttribute Class

The ChangeAttribute class allows to define updates on the XML attribute values of the corresponding model. 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.

Table 4.2.3 shows all attributes and sub-elements for the changeAttribute element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
target	page 21
newValue	page 31

sub-elements	description
notes^o	page 14
annotation^o	page 15

Table 9: *Attributes and nested elements for ChangeAttribute.* elements^o *denotes optional elements.*

One required attribute of the `changeAttribute` element is the target of change, i. e. the location of the addressed XML attribute.

4.2.4 The `newValue` attribute

The second required attribute in the `changeAttribute` element is `newValue`, which assignes a new value to the targeted XML attribute.

An example for an SBML model is the update of the initial concentration of a certain parameter, as shown in listing 30.

```
<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>
</model>
```

Listing 30: *The* `changeAttribute` *element and its* `newValue` *attribute*

4.2.5 The ComputeChange Class

The ComputeChange class allows to make changes on any element of the XML file addressable by an XPath expression. The changes are described by mathematical expressions using a subset of MathML.

Table 4.2.5 on the next page shows all attributes and sub-elements for the computeChange element as defined by the SED-ML Level 1 Version 1 XML Schema.

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 addressed by their ID.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
target	page 21

sub-elements	description
notes^o	page 14
annotation^o	page 15
listOfVariables^o	page 23
listOfParameters^o	page 23
math	page 32

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

4.2.5.1 *The* `math` *element*

The `math` element encodes mathematical functions. If used as an element of the ComputeChange class, it computes the change of the element or attribute addressed by the target attribute. Level 1 Version 1 supports the subset of MathML 2.0 shown in section 2.1.

Listing 31 shows the use of the `computeChange` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<model [..]>
    <computeChange target="/sbml/model/listOfParameters/parameter[@id='w']">
      <listOfVariables>
        <variable id="camkii" name="active calcium/calmoduline kinase II"
                  target="/sbml/model[@id='calcium']/listOfSpecies/species[@id='KII']" />
        <variable id="w" name="synaptic weight"
                  target="/sbml/model[@id='synapse']/listOfParameters/parameter[@id='w']" />
      <listOfVariables/>
      <listOfParameters>
        <parameter id="w0" name="synaptic weight change" value="1">
        <parameter id="n" name="utrasensitivity to calcium" value="2">
        <parameter id="K" name="sensitivity to calcium" value="1e-6">
      <listOfParameters/>
      <math>
         <apply>
           <plus />
           <ci>w</ci>
           <apply>
             <times />
             <ci>w0</ci>
             <apply>
               <divide />
               <apply>
                 <power />
                 <ci>camkii</ci>
                 <ci>n</ci>
               </apply>
               <apply>
                 <plus />
                 <apply>
                   <power />
                   <ci>K</ci>
                   <ci>n</ci>
                 </apply>
                 <apply>
                   <power />
                   <ci> camkii </ci>
                   <ci>n</ci>
                 </apply>
               </apply>
             </apply>
           </apply>
         </apply>
      </math>
    </computeChange>
  </listOfChanges>
</model>
```

Listing 31: *The computeChange element*

The example in listing 31 computes a change on the model parameter w. To do so, it defines two variables. `camkii` corresponds to the value of the species `KII` defined in the model with ID `calcium`. `w` corresponds to the value of the parameter `w` defined in the model with ID `synapse`. In addition, three further parameter are defined: `w0=1`, `n=2`, and `K=1e-6`. The mathematical expression in the mathML then computes the change of w using the encoded equation:

$w + wo \times \frac{camkii^n}{camkii^n + K^n}$

4.3 The Simulation Class

A simulation is the execution of some defined algorithm(s). Simulations are described differently depending on the type of simulation experiment to be performed (Figure 25). Simulation is an abstract class

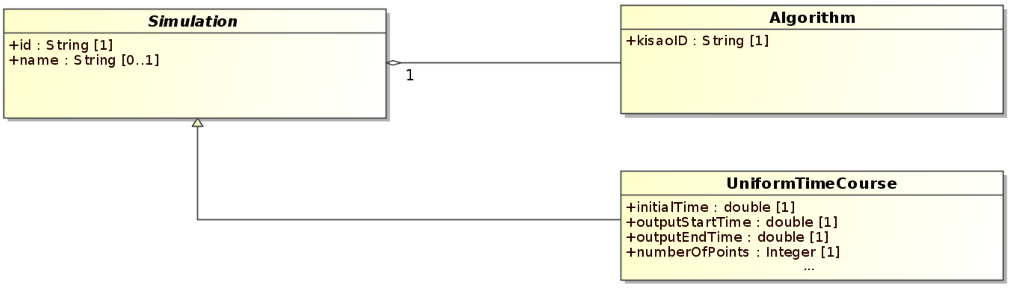

Figure 25: *The SED-ML Simulation class*

and serves as the container for the different types of simulation experiments. SED-ML Level 1 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. Simulation algorithms used for the execution of a simulation setup are defined in the Algorithm class.

Table 4.3 shows all attributes and sub-elements for the simulation element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
algorithm	page 34

Table 11: *Attributes and nested elements for simulation.* elements^o *denotes optional elements.*

Listing 32 shows the use of the `simulation` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<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 32: *The SED-ML* `listOfSimulations` *element, defining two different simulations*

Two timcourses with uniform range are defined.

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.

Each instance of the Simulation class must contain one reference to a simulation algorithm. And each 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 sub-elements for the Algorithm element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	**description**
metaid^o	page 14
kisaoID	page 12
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

The example given in code snipped 32, completed by algorithm definitions looks as in listing 33.

```
<listOfSimulations>
 <uniformTimeCourse id="s1" name="time course simulation 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 33: *The SED-ML* **`algorithm`** *element, defining 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 (Stochsim nearest neighbor).

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 sub-elements for the uniformTimeCourse element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	**description**
metaid^o	page 14
id	page 13
name^o	page 13
initialTime	page 35
outputStartTime	page 35
outputEndTime	page 35
numberOfPoints	page 35
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
algorithm	page 34

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

Listing 34 shows the use of the `uniformTimeCourse` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```xml
<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 34: *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

The attribute `initialTime` of type `double` represents the time from which to start the simulation. Usually this will be `0`. For an example, see listing 34.

4.3.4 The `outputStartTime` attribute

Sometimes a researcher is not interested in simulation results at the start of the simulation (i.e. the initial time). To accommodate this in SED-ML the `uniformTimeCourse` class uses the attribute `outputStartTime` of type `double`. To be valid the `outputStartTime` cannot be before `initialTime`. For an example, see listing 34.

4.3.5 The `outputEndTime` attribute

The attribute `outputEndTime` of type `double` marks the end time of the simulation. For an example, see listing 34.

4.3.6 The `numberOfPoints` attribute

When executed, the `uniformTimeCourse` simulation produces output on a regular grid starting with `outputStartTime` and ending with `outputEndTime`. The attribute `numberOfPoints` of type `integer`, describes the number of points expected in the result. Software interpreting the `uniformTimeCourse` is expected to produce a first outputPoint at time `outputStartTime` with the initial values of the model to be simulated, and then `numberOfPoints` output points with the results of the simulation. Thus a total of `numberOfPoints + 1` output points will be produced.

Just because the output points lie on the regular grid described above, this does not mean that the simulation algorithm has to work with the same step size. Usually the step size the simulator chooses will be adaptive and much smaller than the required output step size. On the other side a stochastic simulator might not have any new events occurring between two grid points. Nevertheless the simulator has to produce data on this regular grid. For an example, see listing 34.

4.4 The Task Class

A task in SED-ML links a model to a certain simulation description via the respective identifiers (Figure 26 on the following page), using the modelReference and the simulationReference. In SED-ML Level 1 Version 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. Please note, that the tasks may be executed in any order, as XML does not have an ordering concept.

Table 4.4 on the next page shows all attributes and sub-elements for the task element as defined by the SED-ML Level 1 Version 1 XML Schema.

Listing 35 shows the use of the `task` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```xml
<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 35: *The `task` element*

In the example, a simulation setting *simulation1* is applied first to *model1* and then is applied to *model2*.

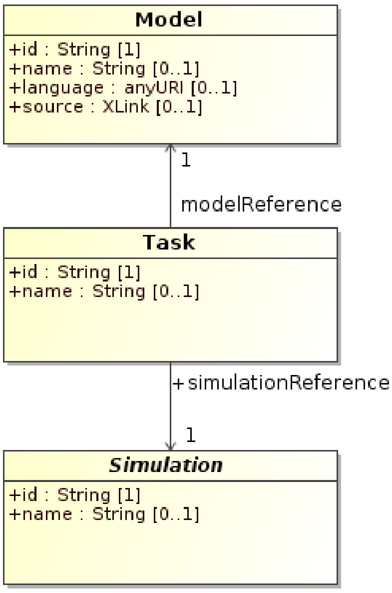

Figure 26: *The SED-ML Task class*

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
modelReference	page 18
simulationReference	page 19
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

4.5 The DataGenerator Class

The DataGenerator class prepares the raw simulation results for later output (Figure 27 on the following page). It encodes the post-processing the researcher intends to be applied to the simulation data. The post-processing steps could be anything, from simple normalisations of data to mathematical calculations.

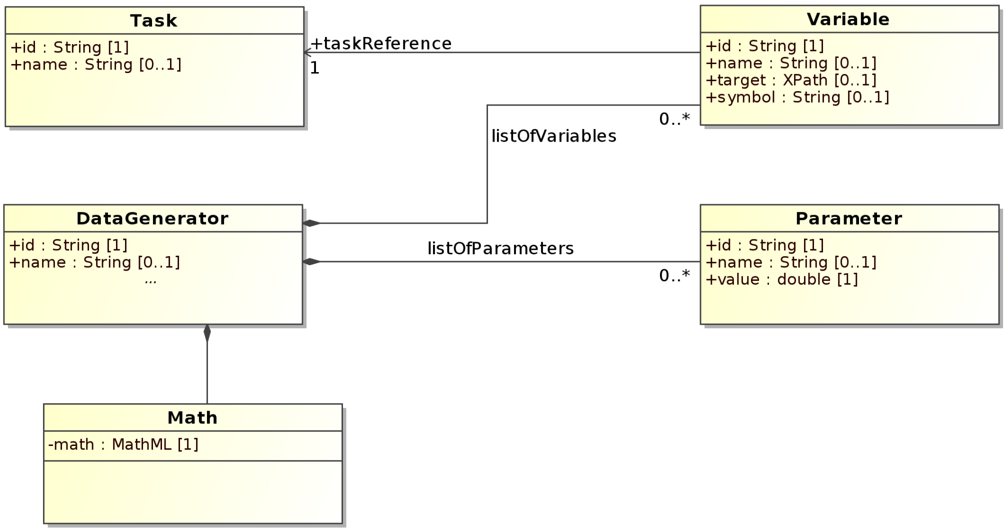

Figure 27: *The SED-ML DataGenerator class*

Table 4.5 shows all attributes and sub-elements for the dataGenerator element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13

sub-elements	description
math	page 32
notes^o	page 14
annotation^o	page 15
variable^o	page 20
parameter^o	page 22

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

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 for the calculation of the data generator. Mathematical functions available for the specification of DataGenerator variables are given in section 2.1 on page 10. Within the mathematical expression, variables defined in the listOfVariables and parameters defined in the listOfParameters can be used.

Listing 36 shows the use of the `listOfDataGenerators` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

```
<listOfDatGenerators>
 <dataGenerator id="d1" name="time">
  <listOfVariables>
   <variable id="time" taskReference="task1" symbol="urn:sedml:symbol:time" />
  </listOfVariables >
  <listOfParameters />
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <ci> time </ci>
  </math>
 </dataGenerator>
 <dataGenerator id="d2" 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 36: *The* `listOfDataGenerators` *element, defining two data generators* time *and* LaCI repressor

The `listOfDataGenerator` contains two `dataGenerator` elements. The first one, `d1`, refers to the task definition `t1` (which itself refers to a particular model), and from the corresponding model it reuses the symbol `time`. The second one, `d2`, references a particular species defined in the same model (and referred to via the `taskReference="t1"`). The model species with ID `PX` is reused for the data generator `d2` without further post-processing.

4.6 The Output Class

The Ouput class describes how the results of a simulation should be presented to the user (Figure 28). It does not contain the data itself, but the type of output and the data generators used with the output

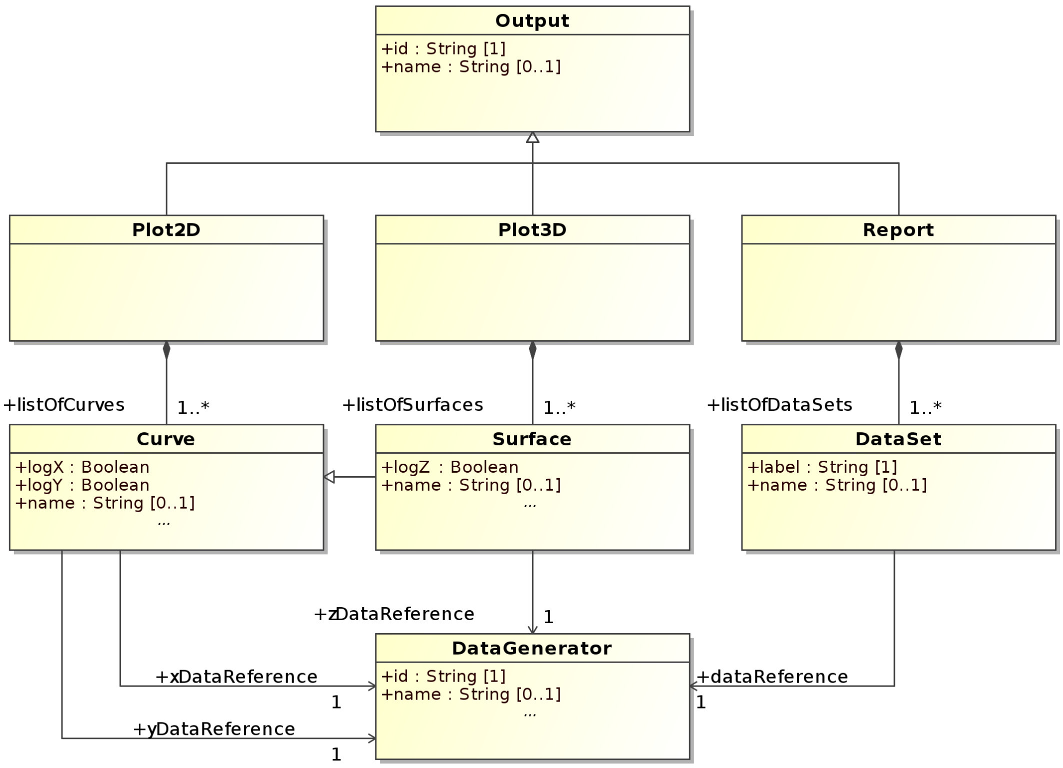

Figure 28: *The SED-ML Output class*

type.

Table 4.6 on the next page shows all attributes and sub-elements for the output element as defined by the SED-ML Level 1 Version 1 XML Schema.

The types of output pre-defined in SED-ML Level 1 Version 1 are plots and reports. The output can be defined as a 2D plot or alternatively as a 3D plot.

Note that even though the terms "2D plot" and "3D plot" are used, the exact type of plot is not specified. In other words, whether the 3D plot represents a surface plot, or three dimensional lines in space, cannot be distinguished by SED-ML alone. It is expected that applications use annotations for this purpose.

4.6.1 The Plot2D Class

A 2 dimensional plot (Figure 29 on the following page) contains a number of curve definitions.

Table 4.6.1 on the next page shows all attributes and sub-elements for the plot2D element as defined by

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
plot2D^o	page 38
plot3D^o	page 39
report^o	page 40

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

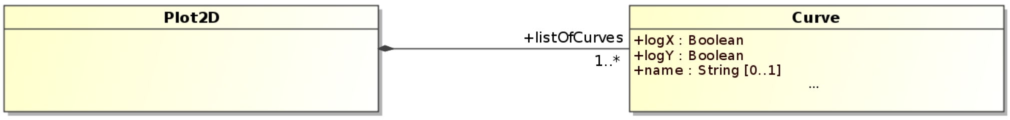

Figure 29: *The SED-ML Plot2D class*

the SED-ML Level 1 Version 1 XML Schema.

attribute	description
notes^o	page 14
annotation^o	page 15
curve	page 40

Table 17: *Attributes and nested elements for plot2D.* elements^o *denotes optional elements.*

Listing 37 shows the use of the `listOfCurves` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 37: *The `plot2D` element with the nested `listOfCurves` element*

The listing shows the definition of a 2 dimensional plot containing one curve element inside the `listOfCurves`. The curve definition follows in section 4.7 on page 40.

4.6.2 The Plot3D Class

A 3 dimensional plot (Figure 30 on the following page) contains a number of surface definitions.

Table 4.6.2 on the next page shows all attributes and sub-elements for the plot3D element as defined by the SED-ML Level 1 Version 1 XML Schema.

Listing 38 shows the use of the `plot3D` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 38: *The `plot3D` element with the nested `listOfSurfaces` element*

Figure 30: *The SED-ML Plot3D class*

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
surface	page 42

Table 18: *Attributes and nested elements for plot3D.* elements^o *denotes optional elements.*

The example defines one surface for the 3 dimensional plot. The surface definition follows in section 4.8 on page 42.

4.6.3 The Report Class

A report defines da data table consisting of several single instances of the DataSet class (Figure 31).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.

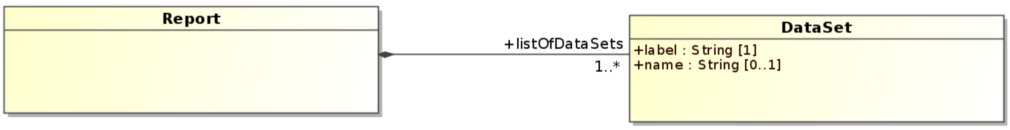

Figure 31: *The SED-ML Report class*

Table 4.6.3 on the next page shows all attributes and sub-elements for the report element as defined by the SED-ML Level 1 Version 1 XML Schema.

Listing 39 shows the use of the `listOfDataSets` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 39: *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

A curve needs a data generator reference to refer to the data that will be plotted on the x-axis, using the xDataReference. A second data generator reference is needed to refer to the data that will be plotted on the y-axis, using the yDataReference.

Table 4.7 on page 42 shows all attributes and sub-elements for the curve element as defined by the SED-ML Level 1 Version 1 XML Schema.

Listing 40 shows the use of the `curve` element in a SED-ML file as defined by the SED-ML Level 1

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
sub-elements	**description**
notes^o	page 14
annotation^o	page 15
dataSet	page 44

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

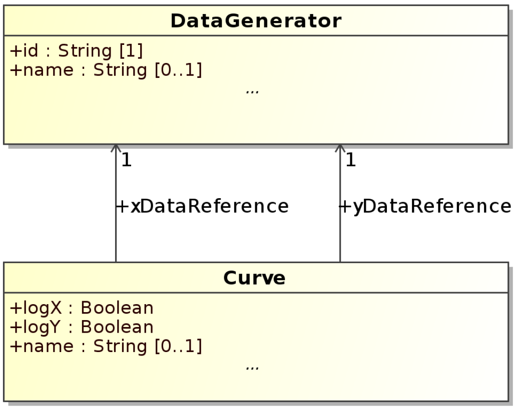

Figure 32: *The SED-ML Curve class*

Version 1 XML Schema.

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

Listing 40: *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.

4.7.1 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 41:

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

Listing 41: *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 42:

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

Listing 42: *The SED-ML* `logY` *attribute, defining a logarithmic output on the y-axis of the according output*

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
logX	page 41
xDataReference	page 42
logY	page 41
yDataReference	page 42
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

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. The xDataReference data type is string. However, the valid values for the xDataReference are restricted to the IDs of already defined DataGenerator objects.

An example for the definition of a curve is given in listing 40. xDataReference is also used in the definition of the x-axis of a Surface in a 3D Plot.

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. 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 listing 40. yDataReference is also used in the definition of the y-axis of a Surface in a 3D Plot.

4.8 The Surface Class

A surface is a three-dimensional figure representing a simulation result (Figure 33).

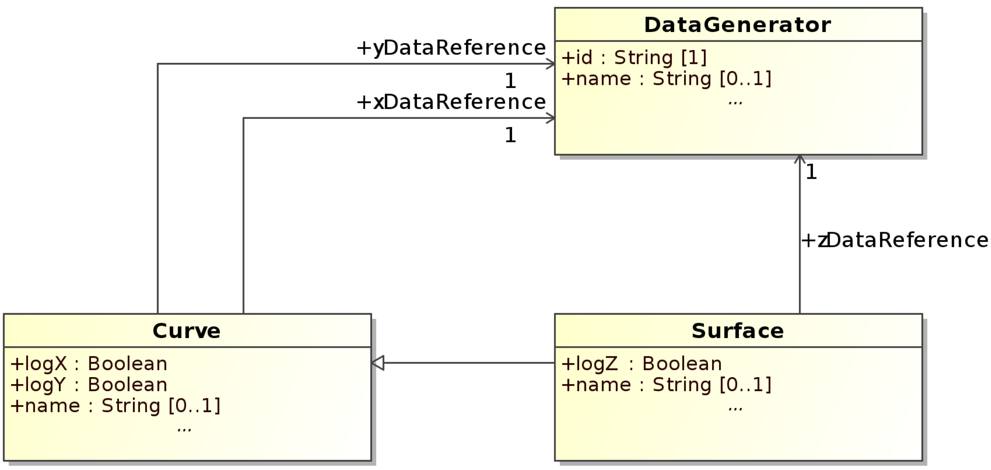

Figure 33: *The SED-ML Surface class*

Creating an instance of the Surface class demands the definition of three different axes, that is which data to plot on which axis and in which way. The aforementioned xDataReference and yDataReference attributes define the according data generators 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.

Table 4.8 shows all attributes and sub-elements for the surface element as defined by the SED-ML Level 1 Version 1 XML Schema. Listing 43 shows the use of the `surface` element in a SED-ML file as defined

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
logX	page 41
xDataReference	page 42
logY	page 41
yDataReference	page 42
logZ	page 43
zDataReference	page 43
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 43: *The SED-ML* `surface` *element, defining the output showing the result of the referenced task*

Here, only one surface 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`, and 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.

4.8.1 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. 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 44:

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

Listing 44: *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 valid values for the zDataReference are restricted to the IDs of already defined DataGenerator objects.

An example using the `zDataReference` is given in listing 43 on page 43.

4.9 The DataSet Class

The DataSet class holds definitions of data to be used in the Report class (Figure 34). Data sets are

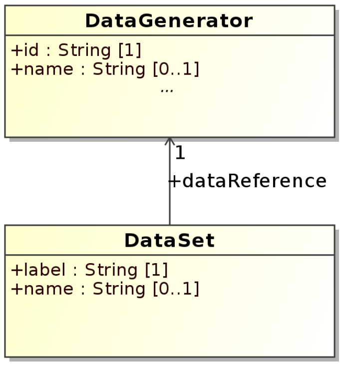

Figure 34: *The SED-ML DataSet class*

labeled references to instances of the DataGenerator class.

Table 4.9 shows all attributes and sub-elements for the dataSet element as defined by the SED-ML Level 1 Version 1 XML Schema.

attribute	description
metaid^o	page 14
id	page 13
name^o	page 13
dataReference	page 44
label	page 44
sub-elements	**description**
notes^o	page 14
annotation^o	page 15

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

4.9.1 The `dataReference` attribute

The dataReference attribute contains the ID of a dataGenerator element. As such it represents a link to that data generator. The data produced by that particular data generator is used to fill the according data set in the report.

Listing 45 shows the use of the `dataSet` element in a SED-ML file as defined by the SED-ML Level 1 Version 1 XML Schema.

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

Listing 45: *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 52).

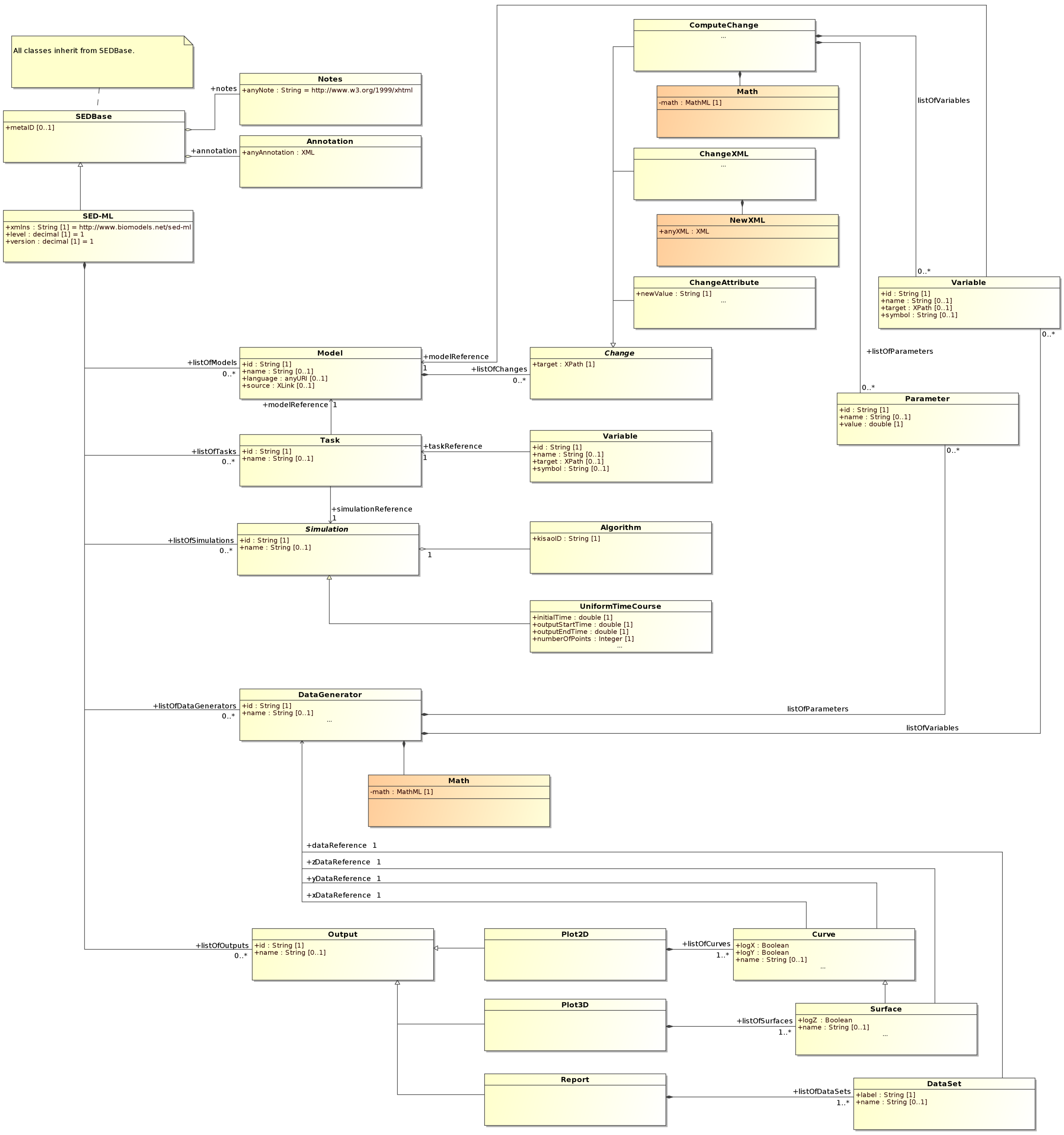

Figure 35: *The SED-ML UML class diagram*

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.

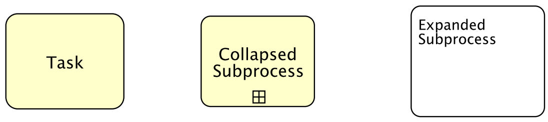

Figure 36: *BPMN activities: task, collapsed process, expanded subprocess*

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.

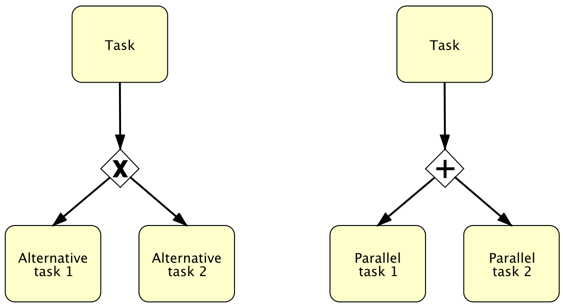

Figure 37: *BPML gateway types: Exclusive (left), parallel (right)*

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, 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, 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).

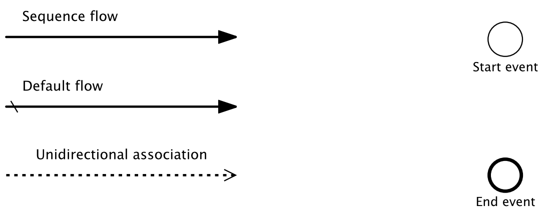

Figure 38: *BPML connectors (left) and events (right).*

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

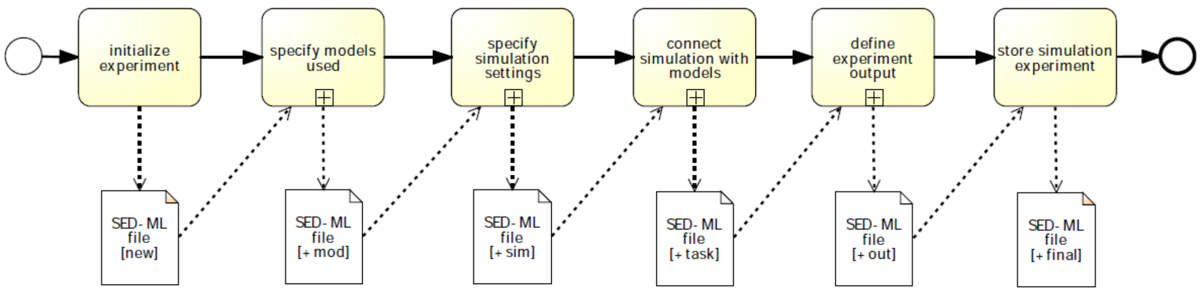

Figure 39: *The process of defining a simulation experiment in SED-ML (overview)*

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. 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 Li et al. [2010], 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

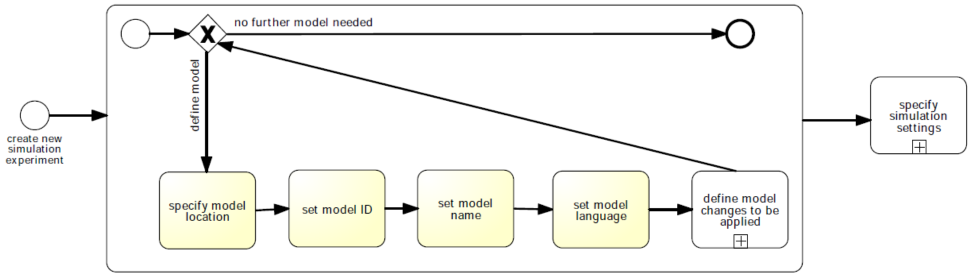

Figure 40: *The process of defining model(s) in SED-ML*

participating in the experiment 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 state analysis or

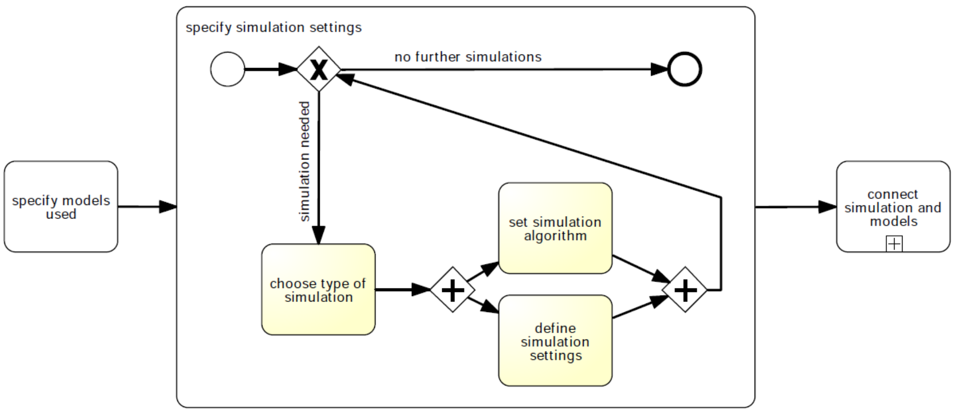

Figure 41: *The process of defining simulation(s) in SED-ML*

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

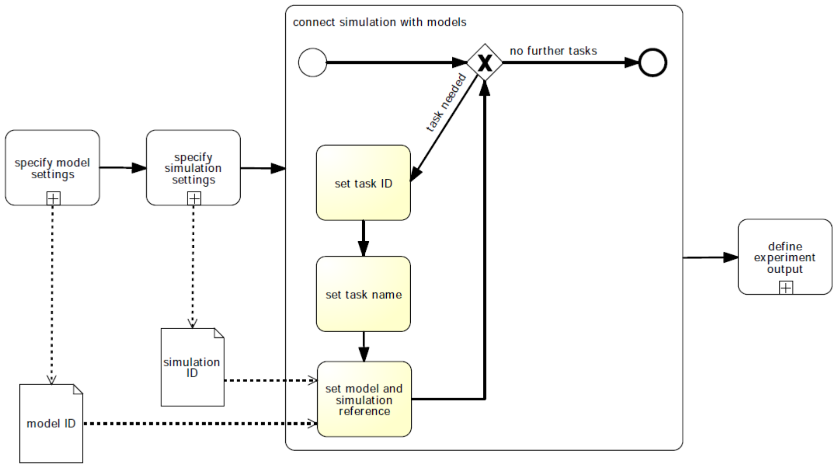

Figure 42: *The process of defining simulation task(s) in SED-ML*

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

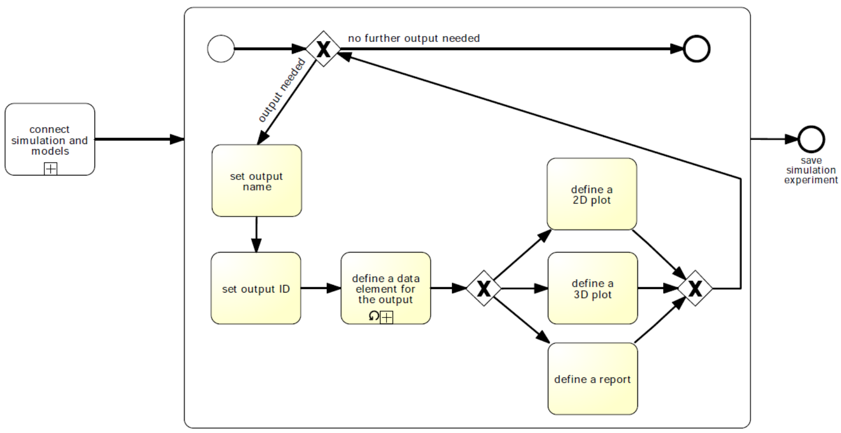

Figure 43: *The process of defining output(s) in SED-ML*

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

B.1.5 Data Generator

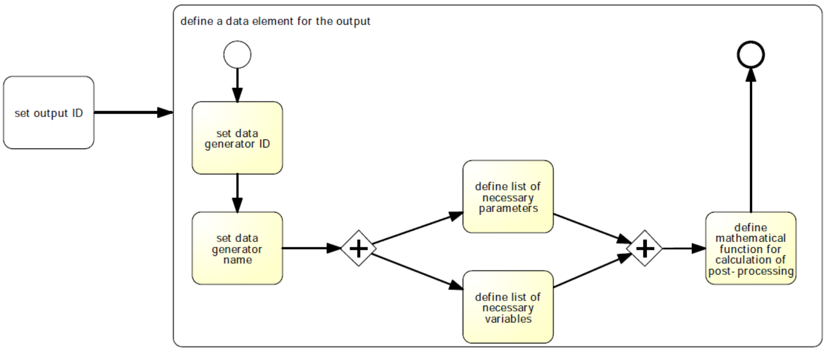

Figure 44: *The process of defining data generator(s) in SED-ML*

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 46 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">
    <xs:import namespace="http://www.w3.org/1998/Math/MathML"
        schemaLocation="sbml-mathml.xsd" />

<!-- global element declarations -->
    <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" />
            <!-- either target or symbol have to be used  in the variable definition-->
            <xs:attribute name="target" type="xs:token" use="optional" />
            <xs:attribute name="symbol" type="xs:string" use="optional" />
            <xs:attribute name="id" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <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:double" use="required" />
        </xs:complexType>
    </xs:element>
    <xs:element name="algorithm">
        <xs:complexType>
            <xs:attribute name="kisaoID" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="uniformTimeCourse">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="algorithm"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" />
            <xs:attribute name="name" type="xs:string" use="optional" />
            <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:complexType>
    </xs:element>
    <xs:element name="task">
        <xs:complexType>
            <xs:attribute name="simulationReference" type="xs:string"
                use="required" />
            <xs:attribute name="name" type="xs:string" use="optional" />
            <xs:attribute name="modelReference" type="xs:string"
                use="required" />
            <xs:attribute name="id" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:element name="notes" type="xs:string" />
    <xs:element name="annotation" type="xs:string" />
    <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>
    <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="level " type="xs:decimal" use="required"
                        fixed="1" />
                    <xs:attribute name="version" type="xs:decimal" use="required"
                        fixed="1" />
                </xs:extension>
            </xs:complexContent>
```

```
        </xs:complexType>
    </xs:element>
    <xs:element name="plot2D">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="listOfCurves" />
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="optional" />
            <xs:attribute name="id" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:element name="plot3D">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="listOfSurfaces" />
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="optional" />
            <xs:attribute name="id" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:element name="report">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="listOfDataSets" />
            </xs:sequence>
            <xs:attribute name="name" type="xs:string" use="optional" />
            <xs:attribute name="id" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <xs:element name="model">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="listOfChanges" minOccurs="0" />
            </xs:sequence>
            <xs:attribute name="language" type="xs:anyURI" use="optional" default="urn:sedml:language:xml
                " />
            <xs:attribute name="source" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required" />
            <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
    </xs:element>
    <!--  <xs:element name="math" type="math:Math" /> -->
    <xs:element name="listOfVariables">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="variable" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfParameters">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="parameter" maxOccurs="unbounded" />
            </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:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfOutputs">
        <xs:complexType>
            <xs:sequence minOccurs="0">
                <xs:element ref="plot2D" minOccurs="0"  maxOccurs="unbounded"/>
                <xs:element ref="plot3D" minOccurs="0"  maxOccurs="unbounded"/>
                <xs:element ref="report" minOccurs="0"  maxOccurs="unbounded"/>
            </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="computeChange" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <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>
    <xs:element name="curve">
        <xs:complexType>
        <xs:attribute name="id" use="required" type="xs:string" />
            <xs:attribute name="yDataReference" type="xs:string"
                use="required" />
            <xs:attribute name="xDataReference" type="xs:string"
                use="required" />
            <xs:attribute name="name" use="optional" type="xs:string" />
            <xs:attribute name="logY" use="required" type="xs:boolean" default="false"/>
            <xs:attribute name="logX" use="required" type="xs:boolean" default="false"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="surface">
        <xs:complexType>
            <xs:attribute name="id" use="required" type="xs:string" />
            <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="name" use="optional" type="xs:string" />
            <xs:attribute name="logY" use="required" type="xs:boolean" default="false"/>
            <xs:attribute name="logX" use="required" type="xs:boolean" default="false"/>
            <xs:attribute name="logZ" use="required" type="xs:boolean" default="false"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="dataSet">
        <xs:complexType>
            <xs:attribute name="id" use="required" type="xs:string" />
            <xs:attribute name="dataReference" type="xs:string" use="required"></xs:attribute>
            <xs:attribute name="label" use="required" type="xs:string" />
            <xs:attribute name="name" use="optional" type="xs:string" />
        </xs:complexType>
    </xs:element>
    <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>
    <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>
    <xs:element name="computeChange">
        <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 46: *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 `dataGenerators` 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 -->
```

```
  <dataGenerator id="tim1" name="tim mRNA">
   <listOfVariables>
    <variable id="v1" taskReference="task1" target="/cellml:model/cellml:component[@cmeta:id='MT']" />
   </listOfVariables>
   <math:math>
    <math:apply>
     <math:plus />
     <math:ci>v1</math:ci>
    </math:apply>
   </math:math>
  </dataGenerator>
  <!-- definition of data generator for the nuclear PER-TIM complex -->
  <dataGenerator id="per-tim" name="nuclear PER-TIM complex">
   <listOfVariables>
    <variable id="v1" taskReference="task1" target="/cellml:model/cellml:component[@cmeta:id='CN']" />
   </listOfVariables>
   <math:math>
    <math:apply>
     <math:plus />
     <math:ci>v1</math:ci>
    </math:apply>
   </math:math>
  </dataGenerator>
  <!-- definition of data generator for pertubated tim mRNA -->
  <dataGenerator id="tim2" name="tim mRNA (changed parameters)">
   <listOfVariables>
    <variable id="v2" taskReference="task2" target="/cellml:model/cellml:component[@cmeta:id='MT']" />
   </listOfVariables>
   <math:math>
    <math:apply>
     <math:plus />
     <math:ci>v2</math:ci>
    </math:apply>
   </math:math>
  </dataGenerator>
  <!-- definition of data generator for perturbated nuclear PER-TIM complex -->
  <dataGenerator id="per-tim2" name="nuclear PER-TIM complex">
   <listOfVariables>
    <variable id="v1" taskReference="task2" target="/cellml:model/cellml:component[@cmeta:id='CN']" />
   </listOfVariables>
   <math:math>
    <math:apply>
     <math:plus />
     <math:ci>v1</math:ci>
    </math:apply>
   </math:math>
  </dataGenerator>
 </listOfDataGenerators>
 <!-- output definition -->
  <listOfOutputs>
   <!-- definition of a 2D plot to show the tim mRNA concentration with different initial conditions -->
   <plot2D id="plot1" name="tim mRNA with Oscillation and Chaos">
    <!-- definition of two output curves, both plotting per-tim (original and perturbated) against the
         tim concentration (original and perturbated) -->
    <listOfCurves>
     <curve logX="false" logY="false" xDataReference="per-tim" yDataReference="tim1" />
     <curve logX="false" logY="false" xDataReference="per-tim2" yDataReference="tim2" />
    </listOfCurves>
   </plot2D>
  </listOfOutputs>
</sedML>
```

Listing 47: *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, oral presentation at the SBGN workshop, japan, 2008.

Chen Li, Marco Donizelli, Nicolas Rodriguez, Harish Dharuri, Lukas Endler, Vijayalakshmi Chelliah, Lu Li, Enuo He, Arnaud Henry, Melanie Stefan, Jacky Snoep, Michael Hucka, Nicolas Le Novere, and Camille Laibe. Biomodels database: An enhanced, curated and annotated resource for published quantitative kinetic models. *BMC Systems Biology*, 4(1):92+, June 2010. ISSN 1752-0509. doi: 10.1186/1752-0509-4-92. URL http://dx.doi.org/10.1186/1752-0509-4-92.

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.