Simulation Experiment Description Markup Language (SED-ML) : Level 1 Version 2

Draft

July 9, 2013

Disclaimer: This is a working draft of the Simulation Experiment Description Markup Language (SED-ML) Level 1 Version 2 specification. It is not a normative document.

Editors

Dagmar Waltemath	*University of Rostock, Germany*
Frank T. Bergmann	*University of Washington, Seattle, USA*
Jonathan Cooper	*University of Oxford, UK*
David Nickerson	*Auckland Bioengineering Institute, New Zealand*
Nicolas Le Novère	*European Bioinformatics Institute, UK*

The latest release of the Level 1 Version 2 specification is available at
http://sed-ml.org/

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], resulted in such 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., 2011].

This document presents Level 1 Version 2 of the *Simulation Experiment Description Markup Language* (SED-ML), a computer-readable format for encoding simulation experiments. SED-ML files are encoded in the *eXtensible Markup Language* (XML) [Bray et al., 2006]. The SED-ML format is defined by an XML Schema [Fallside et al., 2001].

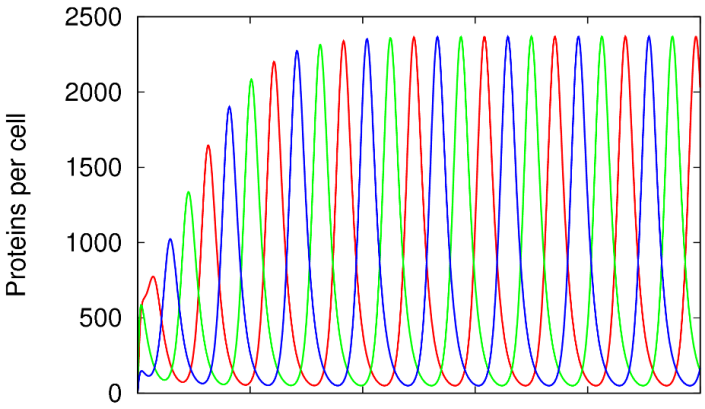

Figure 1.1: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI. The number of repressor proteins lacI, tetR and cI is shown. (taken from Waltemath et al. [2011])*

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. [2011].

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 together 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 (URI) [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] led to the result shown in Figure 1.1.

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.

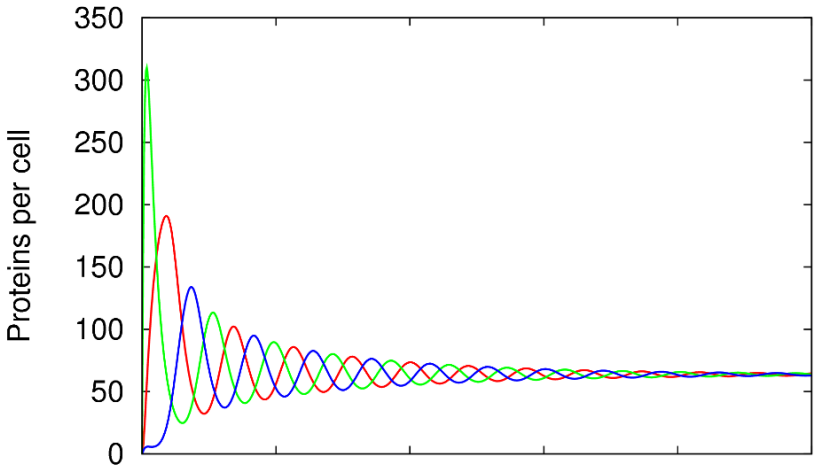

Figure 1.2: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI after modification of the initial values of the* protein copies per promoter *and the* leakiness in protein copies per promoter*. The number of repressor proteins lacI, tetR and cI is shown. (taken from Waltemath et al. [2011])*

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 1.2 shows the result of the simulation.

1.1.3 Applying post-processing

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 given in section 1.1.1 above.)

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

Figure 1.3 on the following page illustrates the result of the simulation after post-processing of the output data.

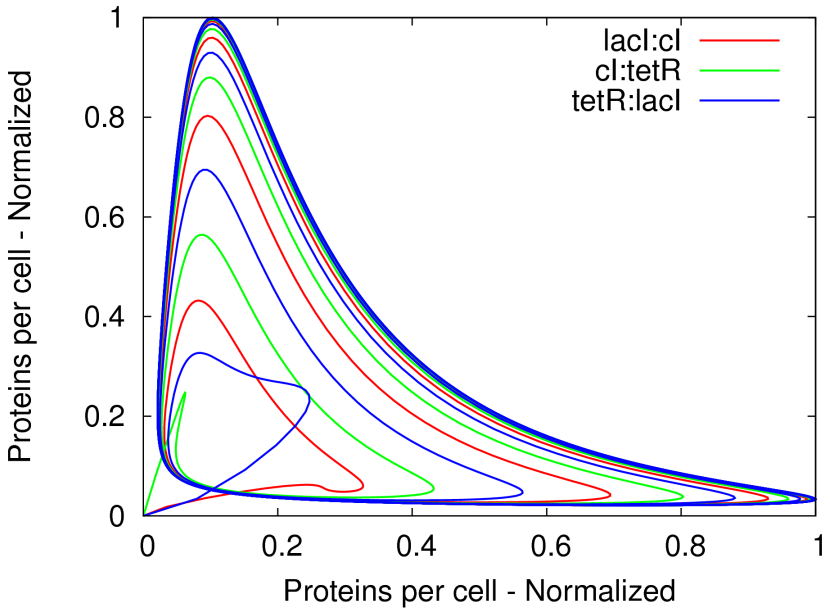

Figure 1.3: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI, showing the normalized temporal evolution of repressor proteins lacI, tetR and cI in phase-plane. (taken from Waltemath et al. [2011])*

2. SED-ML technical specification

This document represents the technical specification of SED-ML. We also provide an XML Schema [W3C, 2004] and a UML Class diagram representation of that XML Schema (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.

It should however be noted that some of the concepts of SED-ML cannot be captured using XML Schema alone. In these cases it is the specification that is considered the normative document.

2.1 Conventions used in this document

2.1.1 UML Classes

A SED-ML UML class (Figure 2.1) 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`.

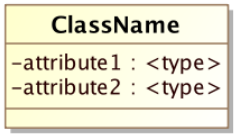

Figure 2.1: *SED-ML UML Class with class names and attributes*

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`.

2.1.2 UML Relationships

2.1.2.1 UML Relation Types

Figure 2.2: *UML Class connectors*

Links between classes specify the connection of objects with each other (Figure 2.2). 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 2.3) 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).

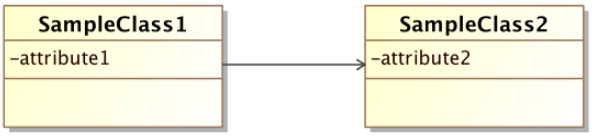

Figure 2.3: *UML Association*

The aggregation (Figure 2.4 on the following page, top) indicates that the objects of the participating classes are connected in a way that one class (`Whole`) consists of several parts (`Part`). In an aggregation, the parts may be independent of the whole. For example, a car (`Whole`) has several parts called wheel (`Part`); however, the wheels can exist independently of the car while the car requires the wheels in order to function.

The composite aggregation (Figure 2.4 on the next page, bottom) indicates that the objects of the participating classes are connected in a way that one class (`Whole`) consists of several parts (`Part`). In contrast to the aggregation, the subelements (`Part`) are dependent on the parent class (`Whole`). An example is that a university (`Whole`) consists of a number of departments (`Part`) which have a so-called

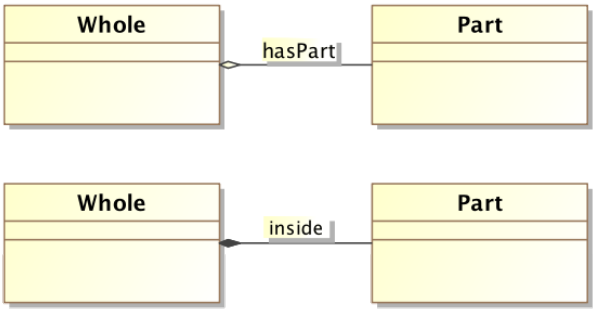

Figure 2.4: *UML Aggregation*

"lifetime responsibility" with the university, e. g. if the university vanishes, the departments will vanish with it [Bell, 2003].

The generalisation (Figure 2.5) allows to extend classes (`BaseClass`) by additional properties. The derived class (`DerivedClass`) inherites all properties of the base class and defines additional ones. In the given example, an instance of `DerivedClass` has two attributes `attribute1` and `attribute2`.

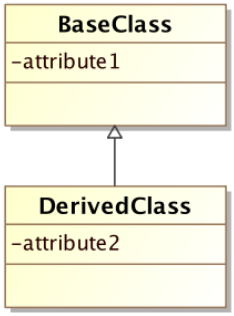

Figure 2.5: *UML Generalisation*

2.1.2.2 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 2.6 expresses that a class is given by a professor, and a professor might give one to many classes.

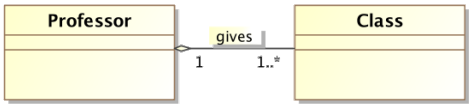

Figure 2.6: *UML Multiplicity in an Aggregation*

2.1.3 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 current SED-ML XML Schema only makes use of *complex type definitions*. An example for a complex type definition is given in listing 2.1:

```
<xs:element name="computeChange">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfVariables" minOccurs="0" />
                        <xs:element ref="listOfParameters" minOccurs="0" />
                        <xs:element ref="math" />
                    </xs:sequence>
                    <xs:attribute name="target" use="required" type="xs:token" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
```

Listing 2.1: *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 current SED-ML XML Schema only uses the *sequence* of elements.

2.1.3.1 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 2.2.

```
<xs:element name="dataGenerator">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfVariables" minOccurs="0" />
                        <xs:element ref="listOfParameters" minOccurs="0" />
                        <xs:element ref="math" />
                    </xs:sequence>
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
```

Listing 2.2: *Multiplicity for complex types in XML Schema*

In this example, 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.

2.1.4 Type extensions

XML Schema offers mechanisms 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 2.3.

```
<xs:element name="sedML">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="SEDBase">
                <xs:sequence>
                    <xs:element ref="listOfSimulations" minOccurs="0" />
                    <xs:element ref="listOfModels" minOccurs="0" />
                    <xs:element ref="listOfTasks" minOccurs="0" />
                    <xs:element ref="listOfDataGenerators" minOccurs="0" />
                    <xs:element ref="listOfOutputs" minOccurs="0" />
                </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>
```

Listing 2.3: *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 `version`.

2.2 Concepts used in SED-ML

2.2.1 MathML subset

The SED-ML specification allows for the encoding of pre-processing applied to the computational 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 a representation of mathematical expressions in other formats, such as SBML and CellML, two of the languages supported by SED-ML.

2.2.1.1 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`

2.2.1.2 MathML Symbols

All the operations listed above only operate on *scalar* values. However, as one of SED-ML's aims 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 2. 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.

2.2.1.3 MathML functions

The following aggregate functions are available for use in SED-ML Level 1 Version 2.

- *min*: Where the minimum of a variable represents the smallest value the simulation experiment yielded (Listing 2.4).

```
<apply>
    <csymbol encoding="text" definitionURL="http://sed-ml.org/#min">
        min
    </csymbol>
    <ci> variableId </ci>
</apply>
```

Listing 2.4: *Example for the use of the MathML* `min` *function.*

- *max*: Where the maximum of a variables represents the largest value the simulation experiment yielded (listing 2.5).

```
<apply>
    <csymbol encoding="text" definitionURL="http://sed-ml.org/#max">
        max
    </csymbol>
    <ci> variableId </ci>
</apply>
```

Listing 2.5: *Example for the use of the MathML* `max` *function.*

- *sum*: All values of the variable returned by the simulation experiment are summed (listing 2.6).

```
<apply>
    <csymbol encoding="text" definitionURL="http://sed-ml.org/#sum">
        sum
    </csymbol>
    <ci> variableId </ci>
</apply>
```

Listing 2.6: *Example for the use of the MathML* `sum` *function.*

- *product*: All values of the variable returned by the simulation experiment are multiplied (listing 2.7).

```
<apply>
    <csymbol encoding="text" definitionURL="http://sed-ml.org//#product">
        product
    </csymbol>
    <ci> variableId </ci>
</apply>
```

Listing 2.7: *Example for the use of the MathML* `product` *function.*

These represent the only exceptions. At this point SED-ML Level 1 Version 2 does not define a complete algebra of vector values. For more information see the description of the DataGenerator class.

2.2.2 URI Scheme

URIs are needed at different points in SED-ML Level 1 Version 2. 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 particular information that can unambiguously be identified.

2.2.2.1 Model references

The preferred way for referencing a model from a SED-ML file is adopted from the MIRIAM URI Scheme. MIRIAM enables identification of a data resource (in this case a model 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]. Its 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[1]. 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 identifiable 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[2]. For the above example of the `urn:miriam:biomodels.db:BIOMD0000000048` model, the resolved URL may look like:

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

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

An alternative means to obtain a model may be to provide a single resource containing necessary models and a SED-ML file. Although a specification of such a resource is beyond the scope of this document, one proposal – SED-ML archive format – is described in Appendix D. Further information on the source attribute referencing the model location is provided in Section 2.4.1.2.

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

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 being `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://sed-ml.org/`. Further information on the language attribute is provided in Section 2.4.1.1.

2.2.2.3 Implicit variables

Some variables used in an experiment are not explicitly defined in the model, but may be implicitly 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 explicitly 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://sed-ml.org/`. From that source, a mapping of SED-ML symbols on possibly existing concepts in the single languages supported by SED-ML is provided.

2.2.2.4 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/`.

2.2.3 XPath usage

XPath is a language for finding information in an XML document [Clarke and DeRose, 1999]. Within Level 1 Version 2, XPath version 1 expressions are used to identify nodes and attributes within an XML representation of a biological model in the following ways:

1. Within a Variable definition, where XPath identifies the model variable required for manipulation in SED-ML.
2. Within a Change definition, where XPath is used to identify the target XML to which a change should be applied.

For proper application, XPath expressions should contain prefixes that allow their resolution to the correct XML namespace within an XML document. For example, the XPath expression referring to a species X in an SBML model:

`/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='X']` ✔ *-CORRECT*

is preferable to

`/sbml/model/listOfSpecies/species[@id='X']` ✘ *-INCORRECT*

which will only be interpretable by standard XML software tools if the SBML file declares no namespaces (and hence is invalid SBML).

Following the convention of other XPath host languages such as XPointer and XSLT, the prefixes used within XPath expressions must be declared using namespace declarations within the SED-ML document, and be in-scope for the relevant expression. Thus for the correct example above, there must also be an ancestor element of the node containing the XPath expression that has an attribute like:

`xmlns:sbml='http://www.sbml.org/sbml/level3/version1/core'`

(a different namespace URI may be used; the key point is that the prefix 'sbml' must match that used in the XPath expression).

2.2.4 KiSAO

An important aspect of a simulation experiment is the simulation algorithm used to solve the system. But the sole reference of a simulation algorithm through its name in form of a string is error prone and ambiguous. 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 abbreviations that are commonly used.

These problems can be solved by using a controlled vocabulary to refer to a particular simulation algorithm. One attempt to provide such a vocabulary is the *Kinetic Simulation Algorithm Ontology* (KiSAO, [Courtot et al., 2011]). 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. KiSAO is available from BioPortal at `http://purl.bioontology.org/ontology/KiSAO`. The project homepage is at `http://www.biomodels.net/kisao/`.

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.

2.2.5 SED-ML resources

Information on SED-ML can be found on `http://sed-ml.org`. 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/`.

2.3 General attributes and classes

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

2.3.1 id

Most objects in SED-ML carry an id attribute. The id attribute, if it exists for an object, is always required and identifies SED-ML constituents unambiguously. The data type for `id` is `SId` which is a datatype derived from the basic XML type `string`, but with restrictions about the characters permitted and the sequences in which those characters may appear. The definition is shown in Figure 2.7.

```
letter   ::=  'a'..'z','A'..'Z'
digit    ::=  '0'..'9'
idChar   ::=  letter | digit | '_'
SId      ::=  ( letter | '_' ) idChar*
```

Figure 2.7: *The definition of the type SId*

For a detailed description see also the SBML specification on the "Type SId" [Hucka et al., 2010, p. 11].

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 2.8.

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

Listing 2.8: *SED-ML identifier definition, e. g. for a model*

The defined model carries the `id` `m00001`. If the model is referenced elsewhere in the SED-ML document, it is referred to by that `id`.

2.3.2 name

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

Listing 2.9 extends the model definition in listing 2.8 by a model name.

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

Listing 2.9: *SED-ML name definition, e. g. for a model*

2.3.3 SEDBase

SEDBase is the base class of SED-ML Level 1 Version 2. All other classes are derived from it. As such it provides means to attach additional information on all other classes (Figure 2.8 on the next page). That information can be specified by human readable Notes or custom Annotations.

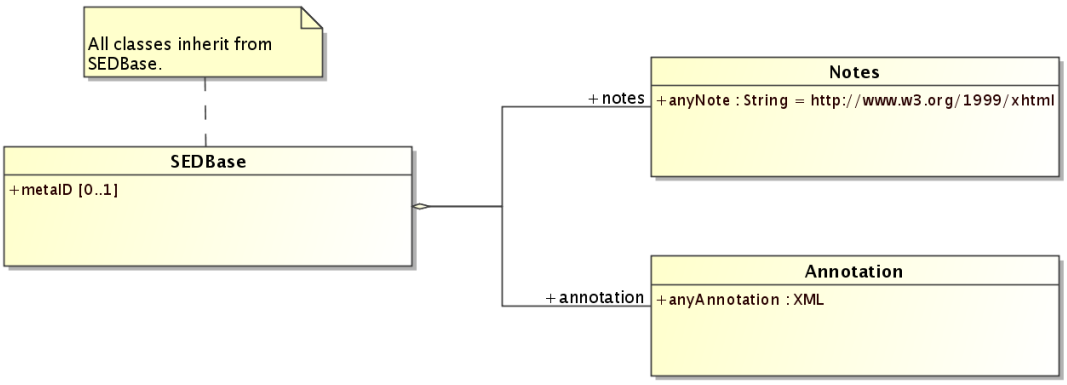

Figure 2.8: *The SEDBase class*

Table 2.1 shows all attributes and sub-elements for the SEDBase element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaID^o	page 19
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.1: *Attributes and nested elements for SEDBase.* xy^o *denotes optional elements and attributes.*

2.3.3.1 `metaid`

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.

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

2.3.3.2 `Notes`

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 used in top level XHTML elements contained within the `notes` element. For further options of how to set the namespace and detailed examples, please refer to [Hucka et al., 2010, p. 14].

Table 2.2 shows all attributes and sub-elements for the Notes element as defined by the SED-ML Level 1 Version 2 XML Schema. `Notes` does not have any further sub-elements defined in SED-ML, nor

attribute	description
xmlns:string “http://www.w3.org/1999/xhtml”	page 22
sub-elements	
well-formed content permitted in XHTML	

Table 2.2: *Attributes and nested elements for Notes.* xy^o *denotes optional elements and attributes.*

attributes associated with it.

Listing 2.10 shows the use of the `notes` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<sedML [..]>
 <notes >
  <p xmlns="http://www.w3.org/1999/xhtml">The enclosed simulation description shows the oscillating
       behaviour of
    the Repressilator model using deterministic and stochastic simulators.</p>
 </notes>
</sedML>
```

Listing 2.10: *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.

2.3.3.3 Annotation

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 corresponding section in the SBML specification [Hucka et al., 2010, pp. 14-16]. The style of annotations in SED-ML is briefly described in Section 2.2.2.4 on page 16.

Table 2.3 shows all attributes and sub-elements for the Annotation element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
none	
sub-elements	**description**
none in the SED-ML namespace	

Table 2.3: *Attributes and nested elements for Annotation.* xy^o *denotes optional elements and attributes.*

Listing 2.11 shows the use of the `annotation` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 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 2.11: *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 the MIRIAM reference standard. 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.

2.3.4 SED-ML top level element

Each SED-ML Level 1 Version 2 document has a main class called SED-ML which defines the document's structure and content (Figure 2.9). It consists of several parts; the parts are all connected to the SED-ML class through aggregation: the Model class (for model specification, see Section 2.4.1), the Simulation class (for simulation setup specification, see Section 2.4.3), the AbstractTask class (for the linkage of models and simulation setups, see Section 2.4.4), the DataGenerator class (for the definition of post-processing, see Section 2.4.7), and the Output class (for the output specification, see Section 2.4.8). All of them are shown in Figure 2.9 and will be explained in more detail in the relevant sections of this document.

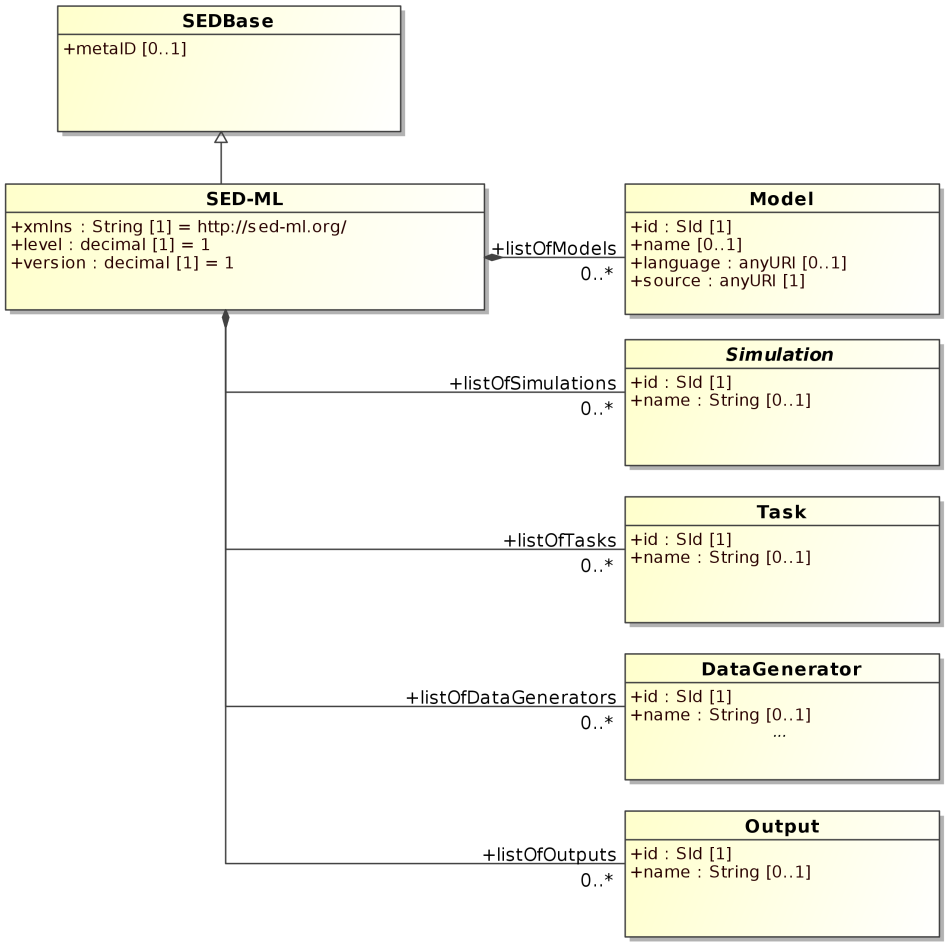

Figure 2.9: *The sub-classes of SED-ML*

Table 2.4 on the next page shows all attributes and sub-elements for the SED-ML element as defined by the SED-ML Level 1 Version 2 XML Schema.

A SED-ML document needs to have the SED-ML namespace defined through the mandatory xmlns attribute. In addition, the SED-ML level and version attributes are mandatory.

The basic XML structure of a SED-ML file is shown in listing 2.12.

```
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns:math="http://www.w3.org/1998/Math/MathML"
       xmlns="http://sed-ml.org/" level="1" version="2">
 <listOfModels />
  [MODEL REFERENCES AND APPLIED CHANGES]
 <listOfSimulations />
  [SIMULATION SETUPS]
 <listOfTasks />
  [MODELS LINKED TO SIMULATIONS]
 <listOfDataGenerators />
  [DEFINITION OF POST-PROCESSING]
```

attribute	description
metaID^o	page 19
xmlns	page 22
level	page 22
version	page 22
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfModels^o	page 29
listOfSimulations^o	page 30
listOfTasks^o	page 30
listOfDataGenerators^o	page 31
listOfOutputs^o	page 31

Table 2.4: *Attributes and nested elements for SED-ML.* xy^o *denotes optional elements and attributes.*

```
12  <listOfOutputs />
13    [DEFINITION OF OUTPUT]
14 </sedML>
```

Listing 2.12: *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).

2.3.4.1 `xmlns`

The xmlns attribute declares the namespace for the SED-ML document. The pre-defined namespace for SED-ML documents is `http://sed-ml.org/`.

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 2.2.1 on page 14.

SED-ML notes use the XHTML namespace `http://www.w3.org/1999/xhtml`. The Notes class is described in Section 2.3.3.2 on page 19.

Additional external namespaces might be used in annotations.

2.3.4.2 `level`

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 2 the value is set to `1`, as shown in the example in Listing 2.12.

2.3.4.3 `version`

The current SED-ML version is "version 2". Minor revisions containing corrections and refinements of SED-ML elements, or new constructs which do not affect backwards compatibility, 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 2 the value is set to `2`, as shown in the example in Listing 2.12.

2.3.5 Reference relations

The reference concept is used to refer to a particular element inside the SED-ML document. It may occur in five different ways in the SED-ML document:

1. as an association between two Models (modelReference),
2. as an association between a Variable and a Model (modelReference),
3. as an association between a Variable and an AbstractTask (taskReference),
4. as an association between a Task and the simulated Model (modelReference) or
5. as an association between a Task and the Simulation run (simulationReference).
6. as an association between an Output and a DataGenerator (dataReference),

The definition of a Task object demands a reference to a particular Model object (modelReference, see Section 2.3.5.1 on page 23); furthermore, the Task object must be associated with a particular Simulation object (simulationReference, see Section 2.3.5.3 on page 24).

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, e.g. 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 2.3.5.1 on page 23).

b. If the reference is used as an association between a Variable object and an AbstractTask object inside the dataGenerator class, then the `variable` element contains a taskReference to unambiguously refer to an observable in a given task (see Section 2.3.5.2 on page 24).

Four different types of data references exist in SED-ML Level 1 Version 2. They are used depending on the *type* of output for the simulation. A 2d plot has an xDataReference and a yDataReference assigned. A 3D plot has in addition a zDataReference assigned. To define a report, each data column has a dataReference assigned.

2.3.5.1 modelReference

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

The `source` attribute of a Model is allowed to reference either a URI or an `SId` to a second Model. Constructs where a model `A` refers to a model `B` and `B` to `A` (directly or indirectly) are invalid.

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 2.13 shows the use of the `modelReference` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 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 2.13: *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` element a list of variables is defined. One of those variable is `v1` which is defined in another model (`cellML`). The XPath expression given in the target attribute identifies the variable in the model which carries the ID `cellML`.

The modelReference is also used to indicate that a Model object is used in a particular Task. Listing 2.14 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 2.14: *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`.

2.3.5.2 taskReference

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

For certain types of post-processing Variable objects need to be created. These link to a task defined within the listOfTasks 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 an AbstractTask object. Listing 2.15 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 2.15: *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 shown in Listing 2.16.

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

Listing 2.16: *Use of the reference relations in a task definition*

Task `t1` references the model `model1`. Therefore we can conclude that the variable `v1` defined in listing 2.15 targets an element of the model with ID `model1`. The targeting process itself will be explained in section 2.3.6.1 on page 26.

2.3.5.3 simulationReference

The simulationReference is used to refer to a particular Simulation in a Task. Listing 2.14 shows 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.

2.3.5.4 dataReference

The dataReference is used to refer to a particular DataGenerator instance from an Output instance. Listing 2.17 shows the reference to a defined data set for a sample SED-ML document.

```
<listOfOutputs>
  <plot2D id="p1" [..] >
    <curve id="c1" xDataReference="dg1" yDataReference="dg2" />
    [..]
  </plot>
</listOfOutputs>
```

Listing 2.17: *Example for the use of data references in a curve definition*

In the example, the output type is a 2D plot, which defines one curve with id `c1`. A curve must refer to two different data generators which describe how to procure the data that is to be plotted on the x-axis and y-axis respectively.

2.3.6 `Variable`

Variables are references to already existing entities, either existing in one of the defined models or implicitly defined symbols (Figure 2.10).

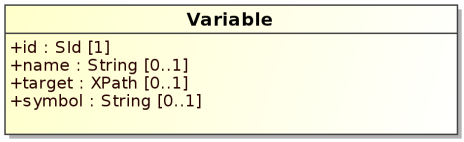

Figure 2.10: *The Variable class*

If the variable is defined through a reference to a model constituent, such as an SBML species, or to an entity within the SED-ML file itself, then the reference is specified using the target attribute. If the variable is defined through a reference to an implicit variable, rather than one explicitly appearing in the model, then the symbol attribute is used, which holds a SED-ML URI. A `variable` is always placed inside a listOfVariables. The `symbol` and `target` attributes must not be used together in a single instance of Variable, although at least one must be present.

Table 2.5 shows all attributes and sub-elements for the Variable element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
symbol	page 27
taskReference	page 24
modelReference	page 23
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.5: *Attributes and nested elements for Variable.* xy^o *denotes optional elements and attributes.*

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. A `variable` element appearing within a functionalRange or setValue element must contain a modelReference if and only if it references a model variable.

Listing 2.18 shows the use of the `variable` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 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 [..]>
```

```
20    <listOfVariables>
21     <variable id="v2" name="time" taskReference="task1" symbol="urn:sedml:symbol:time" />
22       [FURTHER VARIABLE DEFINITIONS]
23    </listOfVariables>
24   </dataGenerator>
25  </listOfDataGenerators>
26  [..]
27 </sedML>
```

Listing 2.18: *SED-ML* `variable` *definitions inside the* `computeChange` *element and inside the* `dataGenerator` *element*

Listing 2.18 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 referenced by the `target` attribute in line 8. The second variable, `v2` (line 21), is used inside a `dataGenerator`. As the variable is time as used in `task1`, the `symbol` attribute is used to refer to the SED-ML URI for time (line 21).

2.3.6.1 `target`

An instance of Variable can refer to a model constituent inside a particular model through an XPath expression stored in the target attribute. XPath can be used to unambiguously identify an element or attribute in an XML file.

The target attribute may also be used to reference an entity within the SED-ML file itself, by containing a fragment identifier consisting of a hash character (`#`) followed by the id of the desired element. As of SED-ML Level 1 Version 2 this is only used to refer to ranges within a repeatedTask (see Listing 2.48 for an example).

Listing 2.19 shows the use of the `target` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
1    <listOfVariables>
2     <variable id="v1" name="TetR protein" taskReference="task1"
3      target="/sbml:sbml/sbml:listOfSpecies/sbml:species[@id='PY']" />
4    </listOfVariables>
```

Listing 2.19: *SED-ML* `target` *definition*

It should be noted that the identifier and names inside the SED-ML document do not *have* to match the identifiers and names that the model and its constituents carry in the model definition. In listing 2.19, 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 also permits using identical identifiers and names as in the referenced models. The following Listing 2.20 is another valid example for the specification of a variable, but uses the sane naming in the variable definition as in the original model (as opposed to Listing 2.19):

```
1    <listOfVariables>
2     <variable id="PY" name="TetR protein"  taskReference="task1"
3      target="/sbml:sbml/sbml:listOfSpecies/sbml:species[@id='PY']" />
4    </listOfVariables>
```

Listing 2.20: *SED-ML variable definition using the original model identifier and name in SED-ML*

```
1 <sbml [..]>
2  <listOfSpecies]
3   <species metaid="PY" id="PY" name="TetR protein" [..]>
4    [..]
5   </species>
6  </listOfSpecies>
7  [..]
8 </sbml>
```

Listing 2.21: *Species definition in the referenced model (extracted from* `urn:miriam:biomodels.db:BIOMD0000000012`*)*

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* (Listing 2.21). Note that while it is possible to write XPath expressions that select multiple nodes within a referenced model, when used within a target attribute a single element or attribute *must* be selected by the expression.

2.3.6.2 `symbol`

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 2.2.2.3 on page 16.

Listing 2.22 shows the use of the `symbol` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The example encodes a computed change of model `m001`. To specify that change, a symbol is defined (i. e. the SED-ML symbol for `time` is assigned to the variable `t1`). How to compute the change itself is explained in Section 2.4.2.6.

```
<listOfVariables>
 <variable id="t1" name="time" taskReference="task1"
 symbol="urn:sedml:symbol:time" />
</listOfVariables>
```

Listing 2.22: *SED-ML `symbol` definition*

2.3.7 `Parameter`

The SED-ML Parameter class creates instances with a constant value (Figure 2.11).

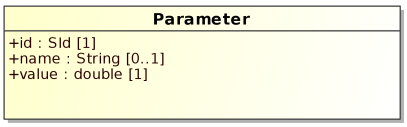

Figure 2.11: *The Parameter class*

SED-ML allows the use of named parameters wherever a mathematical expression is defined to compute some value (e.g. in ComputeChange, FunctionalRange or DataGenerator). In all cases the parameter definitions are local to the particular class defining them. A benefit of naming parameters rather than including numbers directly within the mathematical expression is that notes and annotations may be associated with them.

Table 2.6 shows all attributes and sub-elements for the parameter element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaID^o	page 19
id	page 18
name^o	page 18
value	page 28
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.6: *Attributes and nested elements for parameter.* xy^o *denotes optional elements and attributes.*

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

Listing 2.23 shows the use of the `parameter` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The listing shows the definition of a parameter `p1` with the `value="40"` assigned.

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

Listing 2.23: *The definition of a parameter in SED-ML*

2.3.7.1 `value`

Each parameter has exactly one fixed value. The `value` attribute of XML data type `Double` is required for each `parameter` element.

2.3.8 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 SEDBase. All listOf* elements are optional in a SED-ML document.

2.3.8.1 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 listOfVariables (Figure 2.12). 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 dataGenerator.

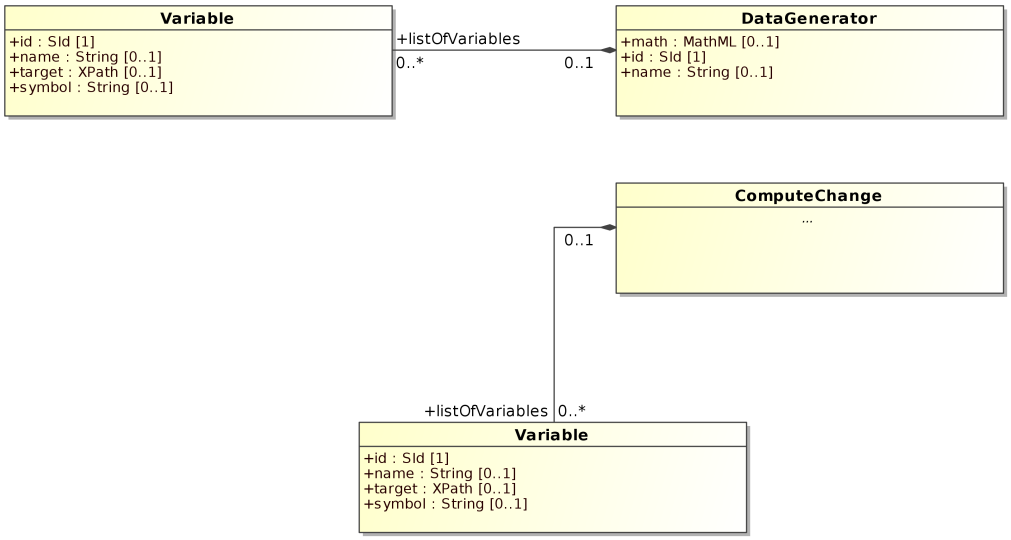

Figure 2.12: *The SED-ML listOfVariables container*

Listing 2.24 shows the use of the `listOfVariables` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The `listOfVariables` is optional and may contain zero to many variables.

```
<listOfVariables>
  <variable id="v1" name="maximum velocity" taskReference="task1"
      target="/cellml:model/cellml:component[@cmeta:id='MP']/cellml:variable[@name='vsP']/@initial_value"
          />
  <variable id="v2" taskReference="task2" symbol="urn:sedml:symbol:time" />
</listOfVariables>
```

Listing 2.24: *SED-ML listOfVariables element*

2.3.8.2 listOfParameters: The parameter definition container

All parameters needed throughout the simulation experiment, whether to compute a change on a model prior to or during simulation (ComputeChange and SetValue), to compute values in a FunctionalRange, or to set up a DataGenerator, are defined inside a listOfParameters (Figure 2.13 on the next page).

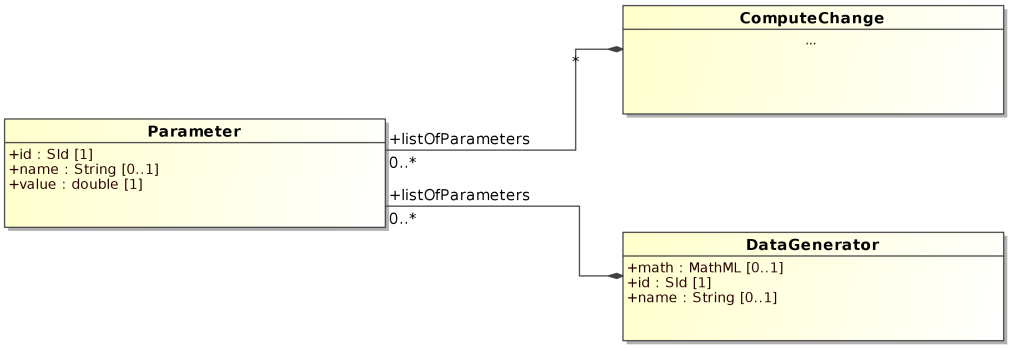

Figure 2.13: *The SED-ML* `listOfParameters` *container*

Listing 2.25 shows the use of the `listOfParameters` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 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 2.25: *SED-ML* `listOfParameters` *element*

2.3.8.3 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 2.14).

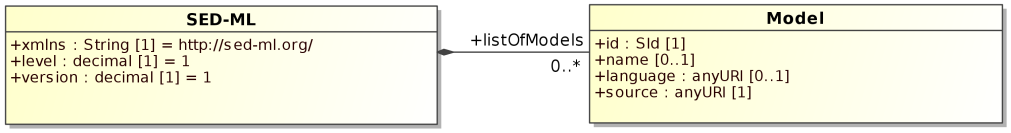

Figure 2.14: *The SED-ML listOfModels container*

Listing 2.26 shows the use of the `listOfModels` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The `listOfModels` is optional and may contain zero to many models. However, if the Level 1 Version 2 document contains one or more `Task` elements, at least one `Model` element must be defined to which the `Task` element refers (c.f. Section 2.3.5.1 on page 23).

```
<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://models.cellml.org/workspace/leloup_gonze_goldbeter_1999/@@rawfile/
        d6613d7e1051b3eff2bb1d3d419a445bb8c754ad/leloup_gonze_goldbeter_1999_a.cellml" />
</listOfModels>
```

Listing 2.26: *SED-ML listOfModels element*

2.3.8.4 listOfChanges: The change definition container

The listOfChanges contains the defined changes to be applied to a particular model (Figure 2.15).

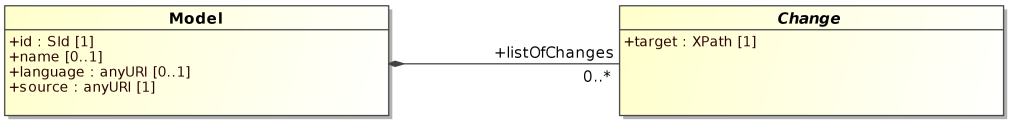

Figure 2.15: *The SED-ML listOfChanges container*

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

Listing 2.27 shows the use of the `listOfChanges` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The `listOfChanges` is nested inside the `model` element.

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

Listing 2.27: *The SED-ML `listOfChanges` element, defining a change on a model*

2.3.8.5 listOfSimulations: The simulation description container

The listOfSimulations element is the container for simulation descriptions (Figure 2.16).

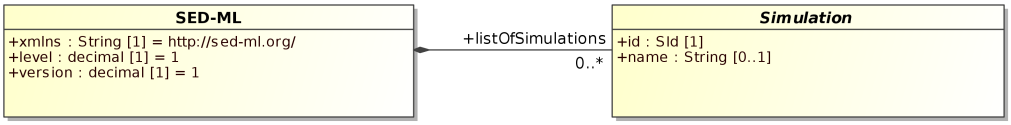

Figure 2.16: *The listOfSimulations container*

Listing 2.28 shows the use of the `listOfSimulation` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.28: *The SED-ML `listOfSimulations` element, containing two simulation setups*

The `listOfSimulations` is optional and may contain zero to many simulations. However, if the Level 1 Version 2 document contains one or more `Task` elements, at least one `Simulation` element must be defined to which the `Task` element refers — see section 2.3.5.3 on page 24.

2.3.8.6 listOfAlgorithmParameters: The container for algorithm parameters

The listOfAlgorithmParameters contains the the settings for the simulation algorithm used in a simulation experiment. It may list several instances of the Algorithm class (Figure **??** on page ??).

Listing 2.29 shows the use of the `listOfAlgorithmParameters` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The `listOfAlgorithmParameters` is optional and may contain zero to many models.

```
<listOfAlgorithmParameters>
 <algorithmParameter kisaoID="KISAO:0000211" value="23"/>
</listOfAlgorithmParameters>
```

Listing 2.29: *SED-ML listOfAlgorithmParameters element*

2.3.8.7 listOfTasks: The task specification container

The listOfTasks element contains the defined tasks for the simulation experiment (Figure 2.17).

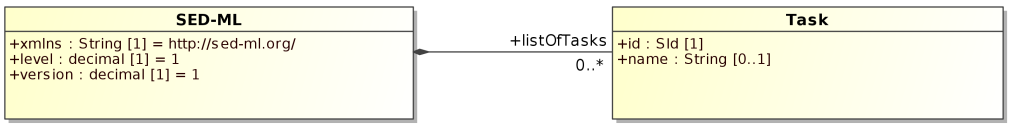

Figure 2.17: *The SED-ML listOfTasks container*

Listing 2.30 shows the use of the `listOfTasks` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.30: *The SED-ML `listOfTasks` element, defining one task*

The `listOfTasks` is optional and may contain zero to many tasks, each of which is an instance of a subclass of AbstractTask. However, if the Level 1 Version 2 document contains a `DataGenerator` element with at least one `Variable` element, at least one task must be defined to which variable(s) in the `DataGenerator` element refer — see Section 2.3.5.2 on page 24.

2.3.8.8 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 2.18).

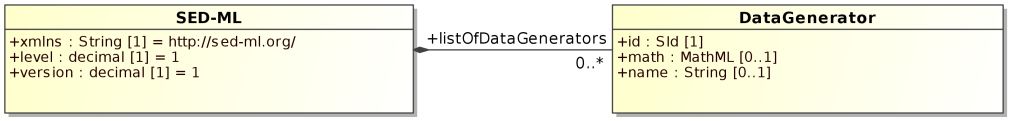

Figure 2.18: *The SED-ML listOfDataGenerators container*

Listing 2.31 shows the use of the `listOfDataGenerators` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.31: *The `listOfDataGenerators` element, defining two data generators* time *and* LaCI repressor

The `listOfDataGenerators` is optional and in general may contain zero to many DataGenerators. However, if the Level 1 Version 2 document contains an `Output` element, at least one `DataGenerator` must be defined to which the `Output` element refers - see section 2.3.5.4 on page 24.

2.3.8.9 listOfOutputs: The output specification container

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

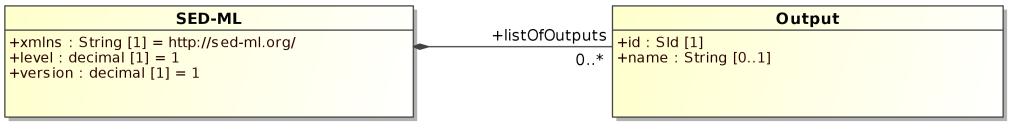

Figure 2.19: *The SED-ML listOfOutputs container*

The output can be defined as either a report, a plot2D or as a plot3D.

Listing 2.32 shows the use of the `listOfOutputs` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The `listOfOutputs` is optional and may contain zero to many outputs.

```
<listOfOutputs>
```

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

Listing 2.32: *The* `listOfOutput` *element*

2.4 SED-ML Components

In this section we describe the major components of SED-ML. We use the UML notation presented in section 2.1.1, and we show the use of SED-ML with XML examples. In addition, we provide a detailed BNMP diagram with explanation of the SED-ML workflow in Appendix E and an XML Schema in Appendix B.

2.4.1 `Model`

The Model class defines the models to be used in the simulation experiment (Figure 2.20).

Figure 2.20: *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).

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

Table 2.7 shows all attributes and sub-elements for the model element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
language^o	page 34
source	page 34
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
change^o	page 35

Table 2.7: *Attributes and nested elements for model.* xy^o *denotes optional elements and attributes.*

Listing 2.33 shows the use of the `model` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
1 <listOfModels>
2   <model id="m0001" language="urn:sedml:language:sbml"
```

```
3    source="urn:miriam:biomodels.db:BIOMD0000000012">
4    <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   <listOfChanges>
12    [MODEL PRE-PROCESSING]
13   </listOfChange>
14  </model>
15  <model id="m0003" language="urn:sedml:language:cellml" source="http://models.cellml.org/workspace/
        leloup_gonze_goldbeter_1999/@@rawfile/d6613d7e1051b3eff2bb1d3d419a445bb8c754ad/
        leloup_gonze_goldbeter_1999_a.cellml">
16   [MODEL PRE-PROCESSING]
17  </model>
18 </listOfModels>
```

Listing 2.33: *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 preprocessing, described in the change element (lines 5-7). Based on the description of the first model `m0001`, the second model is built. 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.

2.4.1.1 `language`

The evaluation of a SED-ML document is required in order for software to decide whether or not it can be used for a particular simulation environment. One crucial criterion 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 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. The list of URNs for languages so far associated with SED-ML is available from the SED-ML web site on `http://sed-ml.org/` (Section 2.2.2.2 on page 16). 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 explicitly 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 for two reasons. Firstly, it helps 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. Secondly, the language attribute is also needed to decide how to handle the implicit variables in the Variable class, as the interpretation of implicit variables depends on the language of the representation format. The concept of implicit variables has been introduced in Section 2.2.2.3 on page 16.

2.4.1.2 `source`

To make a model available for the execution of a SED-ML file, the model `source` must be specified through either a URI or a reference to an `SId` of an existing Model.

The URI should preferably point to a public, consistent location that provides 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[3] can easily be referenced. Even without a MIRIAM URN, SED-ML can be used (Section 2.2.2.1 on page 15).

An example for the definition of a model, and using the URI scheme is given in Listing 2.34.

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

Listing 2.34: *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` using the MIRIAM web service. The resulting URL is `http://www.ebi.ac.uk/biomodels-main/BIOMD0000000012`.

An example for the definition of a model and using a URL is given in Listing 2.35.

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

Listing 2.35: *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` attribute.

2.4.2 Change

SED-ML not only allows to use the sole model for simulation, but also enables the description of changes to be made on the model before simulation (Figure 2.21 on the next page). Changes can be of three distinct types:

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

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

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

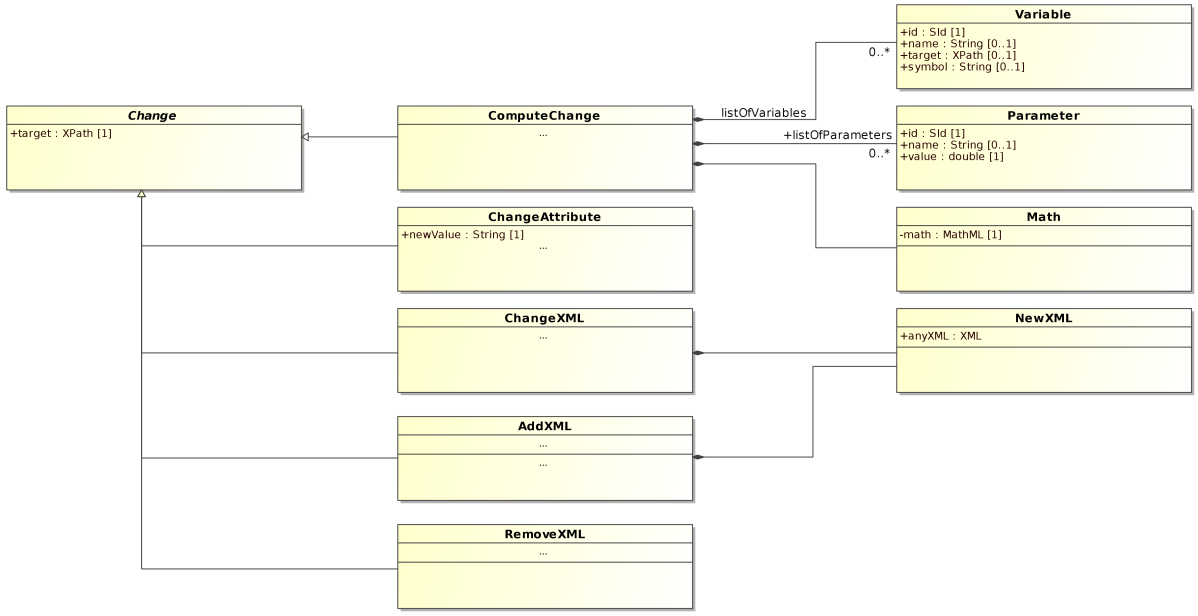

Figure 2.21: *The SED-ML Change class*

Table 2.8 shows all attributes and sub-elements for the change element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.8: *Attributes and nested elements for change.* xy^o *denotes optional elements and attributes.*

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. Except for the cases of ChangeXML and RemoveXML, this XPath expression must always select a *single* element or attribute within the relevant model.

2.4.2.1 `NewXML`

The `newXML` element provides a piece of XML code (Figure 2.21). `NewXML` must hold a valid piece of XML which after insertion into the original model must lead to a valid model file, according to the model language specification (as given by the language attribute).

Table 2.9 shows all attributes and sub-elements for the newXML element as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Table 2.9: *Attributes and nested elements for newXML.* xy^o *denotes optional elements and attributes.*

The `newXML` element is used at two different places inside SED-ML Level 1 Version 2:

1. If it is used as a sub-element of the addXML element, then the XML it contains it is to be *inserted as a child* of the XML element addressed by the XPath.
2. If it is used as a sub-element of the changeXML element, then the XML it contains is to *replace* the XML element addressed by the XPath.

Examples are given in the relevant change class definitions.

2.4.2.2 *AddXML*

The AddXML class specifies a snippet of XML that is to be added as a child of the element selected by the XPath expression in the target attribute (Figure 2.22). The new piece of XML code is provided by the NewXML class.

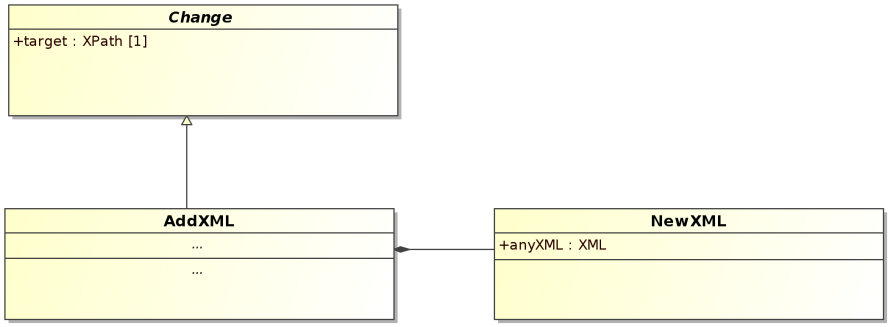

Figure 2.22: *The SED-ML `AddXML` class*

Table 2.10 shows all attributes and sub-elements for the addXml element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
newXML	page 36

Table 2.10: *Attributes and nested elements for addXML. xy^o denotes optional elements and attributes.*

An example for a change that adds an additional parameter to a model is given in listing 2.36.

```
<model language="urn:sedml:language:sbml" [..]>
 <listOfChanges>
  <addXML target="/sbml:sbml/sbml:model/sbml:listOfParameters" >
   <newXML>
     <parameter metaid="metaid_0000010" id="V_mT" value="0.7" />
  </newXML>
  </addXML>
 </listOfChanges>
</model>
```

Listing 2.36: *The `addXML` element with its `newXML` sub-element*

The code of the model is changed so that a parameter with ID `V_mT` is added to its list of parameters. The `newXML` element adds an additional XML element to the original model. The element's name is

`parameter` and it is added to the existing parent element `listOfParameters` that is addressed by the XPath expression in the `target` attribute.

2.4.2.3 *ChangeXML*

The ChangeXML class allows you to replace any XML element(s) in the model that can be addressed by a valid XPath expression (Figure 2.23).

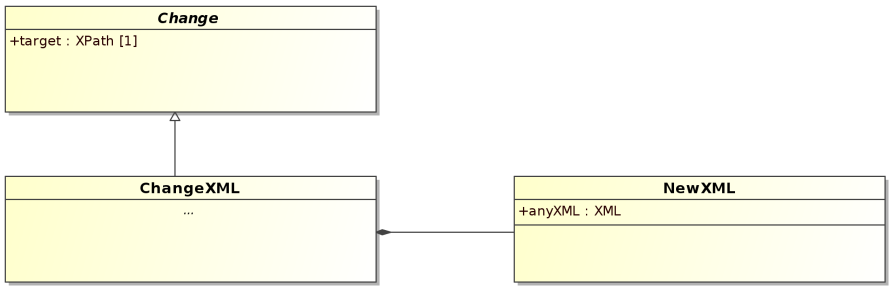

Figure 2.23: *The ChangeXML class*

The XPath expression is specified in the required target attribute (Section 2.3.6.1 on page page 26). The replacement XML content is specified in the NewXML class.

Table 2.11 shows all attributes and sub-elements for the changeXml element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
newXML	page 36

Table 2.11: *Attributes and nested elements for changeXML.* xy^o *denotes optional elements and attributes.*

An example for a change that adds an additional parameter to a model is given in listing 2.37.

```
<model [..]>
 <listOfChanges>
  <changeXML target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='V_mT']" >
   <newXML>
     <parameter metaid="metaid_0000010" id="V_mT_1" value="0.7" />
     <parameter metaid="metaid_0000050" id="V_mT_2" value="4.6"> />
   </newXML>
  </changeXML>
 </listOfChanges>
</model>
```

Listing 2.37: *The changeXML element*

The code of the model is changed in the way that its parameter with ID `V_mT` is substituted by two other parameters `V_mT_1` and `V_mT_2`. The `target` attribute defines that the parameter with ID `V_mT` is to be changed. The `newXML` element then specifies the XML that is to be exchanged for that parameter.

2.4.2.4 *RemoveXML*

The RemoveXML class can be used to delete XML elements or attributes in the model that are addressed by the XPath expression (Figure 2.24 on the next page).

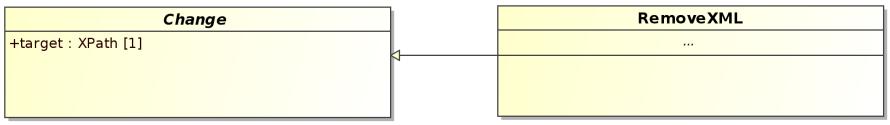

Figure 2.24: *The `RemoveXML` class*

The XPath is specified in the required target attribute.

Table 2.12 shows all attributes and sub-elements for the removeXml element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.12: *Attributes and nested elements for removeXML.* xy^o *denotes optional elements and attributes.*

An example for the removal of an XML element from a model is given in Listing 2.38.

```
<model [..]>
  <listOfChanges>
   <removeXML target="/sbml:sbml/sbml:model/sbml:listOfReactions/sbml:reaction[@id='J1']" />
  </listOfChanges>
</model>
```

Listing 2.38: *The `removeXML` element*

The code of the model is changed by deleting the reaction with ID `V_mT` from the model's list of reactions.

2.4.2.5 `ChangeAttribute`

The ChangeAttribute class allows to define updates on the XML attribute values of the corresponding model (Figure 2.25).

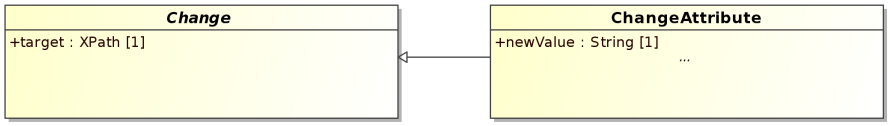

Figure 2.25: *The `ChangeAttribute` class*

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, for the common case of changing an attribute value ChangeAttribute is easier to use, and so it is recommended to use the ChangeAttribute for any changes of an XML attribute's value, and to use the more general ChangeXML for other cases.

ChangeAttribute requires to specify the target of the change, i. e. the location of the addressed XML attribute, and also the new value of that attribute. Note that the XPath expression in the target attribute must select a single attribute within the corresponding model.

Table 2.13 on the following page shows all attributes and sub-elements for the changeAttribute element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
newValue	page 40
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.13: *Attributes and nested elements for ChangeAttribute.* xy^o *denotes optional elements and attributes.*

2.4.2.5.1 `newValue`

The mandatory **`newValue`** attribute assignes a new value to the targeted XML attribute.

The example in Listing 2.39 shows the update of the initial concentration of two parameters inside an SBML model.

```
<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 2.39: *The* `changeAttribute` *element and its* `newValue` *attribute*

2.4.2.6 `ComputeChange`

The ComputeChange class permits to change, prior to the experiment, the numerical value of any element or attribute of a model addressable by an XPath expression, based on a calculation (Figure 2.26).

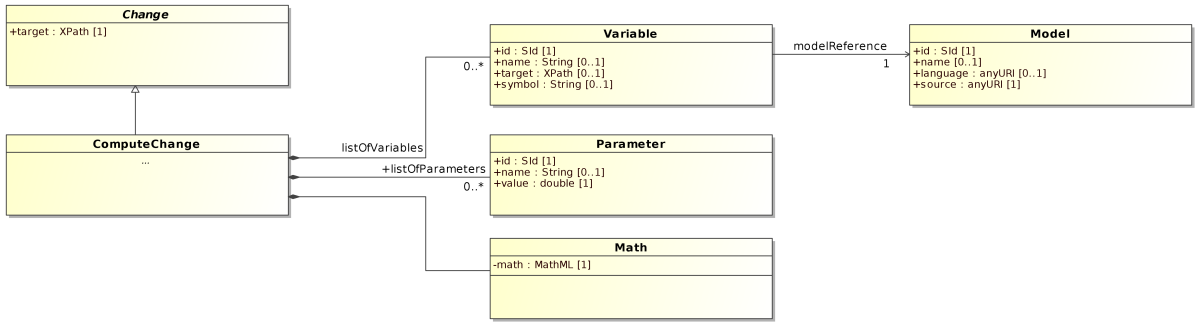

Figure 2.26: *The* `ComputeChange` *class*

The computed new value is described by a mathematical expression using a subset of MathML (see section 2.2.1 on page 14). The computation can use the value of variables from any model defined in the simulation experiment. Those variables need to be defined, and can then be addressed by their ID. A variable used in a ComputeChange must carry a modelReference attribute (page 23) but no taskReference attribute (page 24). To carry out the calculation it may be necessary to introduce additional parameters, that are not defined in any of the models used by the experiment. This is done through the parameter class, and such parameters are thereafter refered to by their ID. Finally, the change itself is specified using an instance of the Math class.

Note that where a ComputeChange refers to another model, that model is not allowed to be modified by ComputeChanges which directly or indirectly refer to this model. In other words, cycles in the definitions of computed changes are prohibited, since then the new values would not be well defined.

Table 2.14 shows all attributes and sub-elements for the computeChange element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 26
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfVariables^o	page 28
listOfParameters^o	page 28
math	page 41

Table 2.14: *Attributes and nested elements for computeChange.* xy^o *denotes optional elements and attributes.*

2.4.2.6.1 Math

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 2 supports the subset of MathML 2.0 shown in section 2.2.1.

Listing 2.40 shows the use of the `computeChange` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<model [..]>
    <computeChange target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='sensor']">
      <listOfVariables>
        <variable modelReference="model1" id="R" name="regulator"
                  target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='regulator']" />
        <variable modelReference="model2" id="S" name="sensor"
                  target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='sensor']" />
      <listOfVariables/>
      <listOfParameters>
        <parameter id="n" name="cooperativity" value="2">
        <parameter id="K" name="sensitivity" value="1e-6">
      <listOfParameters/>
      <math  xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times />
          <ci>S</ci>
          <apply>
            <divide />
            <apply>
              <power />
              <ci>R</ci>
              <ci>n</ci>
            </apply>
            <apply>
              <plus />
              <apply>
                <power />
                <ci>K</ci>
                <ci>n</ci>
              </apply>
              <apply>
                <power />
                <ci>R</ci>
                <ci>n</ci>
              </apply>
            </apply>
          </apply>
        </math>
    </computeChange>
  </listOfChanges>
</model>
```

Listing 2.40: *The computeChange element*

The example in listing 2.40 computes a change of the variable `sensor` of the model `model2`. To do so, it uses the value of the variable `regulator` coming from model `model1`. In addition, the calculation used

two additional parameters, the cooperativity `n`, and the sensitivity `K`. The mathematical expression in the mathML then computes the new initial value of `sensor` using the equation:

$S = S \times \frac{R^n}{K^n + R^n}$.

2.4.3 Simulation

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 2.27).

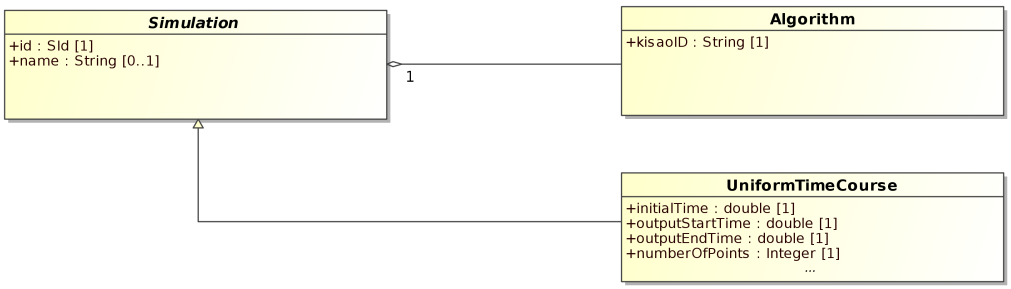

Figure 2.27: *The SED-ML Simulation class*

Simulation is an abstract class and serves as the container for the different types of simulation experiments. SED-ML Level 1 Version 2 offers the predefined simulation classes UniformTimeCourse, OneStep and SteadyState. Further simulation classes are planned for future versions of SED-ML, including simulation classes for bifurcation analysis. Simulation algorithms used for the execution of a simulation setup are defined in the Algorithm class.

Table 2.15 shows all attributes and sub-elements for the simulation element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
algorithm	page 43

Table 2.15: *Attributes and nested elements for simulation. xy^o denotes optional elements and attributes.*

Listing 2.41 shows the use of the `simulation` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<listOfSimulations>
  <uniformTimeCourse [..]>
    [SIMULATION SPECIFICATION]
  </uniformTimeCourse>
  <uniformTimeCourse [..]>
    [SIMULATION SPECIFICATION]
  </uniformTimeCourse>
</listOfSimulations>
```

Listing 2.41: *The SED-ML `listOfSimulations` element, defining two different simulations*

Two timecourses with uniform range are defined.

2.4.3.1 *Algorithm*

SED-ML makes use of the KiSAO ontology (Section 2.2.4 on page 17) 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 (Figure 2.28).

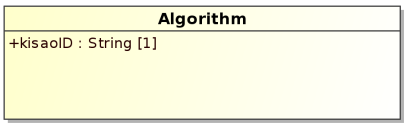

Figure 2.28: *The* `Algorithm` *class*

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, and should be defined in the correct syntax, as defined by the regular expression `KISAO:[0-9]{7}`.

The Algorithm class contains an optional list of parameters (algorithmParameter) that are used to parameterize the particular algorithm used in the simulation.

Table 2.16 shows all attributes and sub-elements for the Algorithm element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
kisaoID	page 17
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
algorithmParameter^o	page 43

Table 2.16: *Attributes and nested elements for algorithm.* xy^o *denotes optional elements and attributes.*

The example given in code snippet in Listing 2.41, completed by algorithm definitions results in the code given in Listing 2.42.

```
<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 2.42: *The SED-ML* `algorithm` *element for two different time course simulations, defining two different algorithms. KISAO:0000030 refers to the* Euler forward method *; KISAO:0000021 refers to the* StochSim nearest neighbor algorithm.

For both simulations, one algorithm is defined. In the first simulation `s1` a deterministic approach has been chosen (Euler forward method), in the second simulation `s2` a stochastic approach is used (Stochsim nearest neighbor).

2.4.3.2 *AlgorithmParameter*

The AlgorithmParameter class allows to parameterize a particular simulation algorithm. The set of possible parameters for a particular instance is determined by the algorithm that is referenced by the kisaoID of the enclosing algorithm element. Parameters of simulation algorithms are unambiguously referenced by the mandatory kisaoID attribute. Their value is set in the mandatory value attribute.

```
<algorithm kisaoID="KISAO:0000032">
```

```
  <listOfAlgorithmParameters>
   <algorithmParameter kisaoID="KISAO:0000211" value="23"/>
  </listOfAlgorithmParameters>
 </algorithm>
```

Listing 2.43: *The SED-ML* `algorithmParameter` *element setting the parameter value for the simulation algorithm. KISAO:0000032 refers to the* explicit fourth-order Runge-Kutta method*; KISAO:00000211 refers to the* absolute tolerance*.*

2.4.3.3 `UniformTimeCourse`

Figure 2.29: *The* `UniformTimeCourse` *class*

Table 2.17 shows all attributes and sub-elements for the uniformTimeCourse element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
initialTime	page 44
outputStartTime	page 44
outputEndTime	page 45
numberOfPoints	page 45
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
algorithm	page 43

Table 2.17: *Attributes and nested elements for uniformTimeCourse.* xy^o *denotes optional elements and attributes.*

Listing 2.44 shows the use of the `uniformTimeCourse` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.44: *The SED-ML* `uniformTimeCourse` *element, defining a uniform time course simulation over 2500 time units with 1000 simulation points.*

2.4.3.3.1 `initialTime`

The attribute `initialTime` of type `double` represents the time from which to start the simulation. Usually this will be `0`. Listing 2.44 shows an example.

2.4.3.3.2 `outputStartTime`

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 2.44.

2.4.3.3.3 outputEndTime

The attribute `outputEndTime` of type `double` marks the end time of the simulation. See Listing 2.44 for an example.

2.4.3.3.4 numberOfPoints

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 hand 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 2.44.

2.4.3.4 OneStep

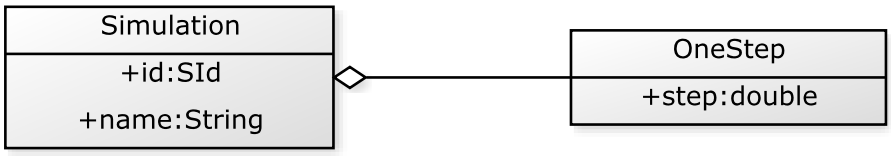

Figure 2.30: *The `OneStep` class*

The SED-ML `oneStep` calculates one further output step for the model from its current state. Note that this does NOT necessarily equate to one integration step. The simulator is allowed to take as many steps as needed. However, after running oneStep, the desired output time is reached.

Table 2.18 shows all attributes and sub-elements for the oneStep element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
step^o	page 46
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
algorithm	page 43

Table 2.18: *Attributes and nested elements for oneStep.* xy^o *denotes optional elements and attributes.*

Listing 2.45 shows the use of the `oneStep` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<listOfSimulations>
  <oneStep id="s1" step="0.1">
    <algorithm kisaoID="KISAO:0000019" />
  </oneStep>
</listOfSimulations>
```

Listing 2.45: *The SED-ML `oneStep` element, specifying to apply the simulation algorithm for another output step of size 0.1.*

2.4.3.4.1 `step`

The `oneStep` class has one required attribute `step` of type `double`. It defines the next output point that should be reached by the algorithm, by specifying the increment from the current output point. Listing 2.45 shows an example.

2.4.3.5 `SteadyState`

The `SteadyState` class represents a steady state computation (as for example implemented by NLEQ or Kinsolve).

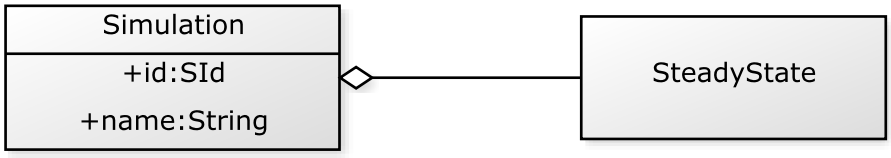

Figure 2.31: *The `SteadyState` class*

Table 2.19 shows all attributes and sub-elements for the steadyState element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
algorithm	page 43

Table 2.19: *Attributes and nested elements for steadyState.* xy^o *denotes optional elements and attributes.*

Listing 2.46 shows the use of the `steadyState` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<listOfSimulations>
  <steadyState id="steady">
    <algorithm kisaoID="KISAO:0000282" />
  </steadyState >
</listOfSimulations>
```

Listing 2.46: *The SED-ML `steadyState` element, defining a steady state simulation with id `steady`.*

2.4.4 `Abstract Task`

An abstract task in SED-ML represents the base class for all SED-ML tasks. It is not meant to be instantiated directly.

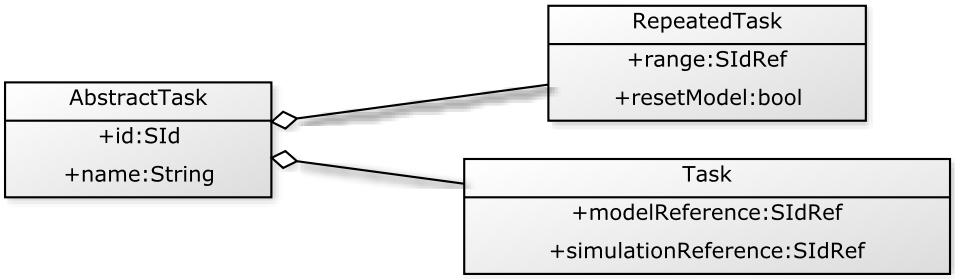

Figure 2.32: *The SED-ML Abstract Task class*

Table 2.20 shows all attributes and sub-elements for the abstractTask element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	**description**
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.20: *Attributes and nested elements for abstractTask. xy^o denotes optional elements and attributes.*

2.4.5 Task

A task in SED-ML links a model to a certain simulation description via their respective identifiers (Figure 2.33), using the modelReference and the simulationReference. The task class receives the id and name attributes from the AbstractTask.

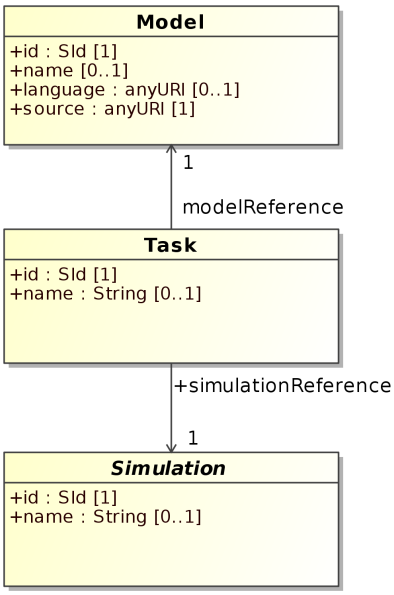

Figure 2.33: *The SED-ML Task class*

In SED-ML Level 1 Version 2 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. Please note that the tasks may be executed in any order, as determined by the implementation.

Table 2.21 shows all attributes and sub-elements for the task element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
modelReference	page 23
simulationReference	page 24
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.21: *Attributes and nested elements for task.* xy^o *denotes optional elements and attributes.*

Listing 2.47 shows the use of the `task` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<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 2.47: *The* `task` *element*

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

2.4.6 Repeated Task

The repeatedTask class provides a generic looping construct, allowing complex tasks to be represented by composing separate steps. It performs a specified task (or sequence of tasks) multiple times (where the exact number is specified through a range construct), while allowing specific quantities in the model to be altered at each iteration (as defined in the listOfChanges).

The RepeatedTask inherits from AbstractTask. Additionally it has two required attributes range and resetModel as well as child elements listOfRanges, listOfChanges and listOfSubTasks. The latter two child elements are optional.

Note that the order of activities within each iteration of a repeatedTask is as follows. Firstly the model is reset, if specified by the `resetModel` attribute. Secondly any changes to the model specified by `setValue` elements are made. Finally, the `subTasks` are executed once each in order.

The RepeatedTask class may optionally carry modelReference and simulationReference attributes. If these are present it must have no subTasks defined. The attributes provide a short-hand that is equivalent to providing a single `subTask` reference to a task with the given modelReference and simulationReference.

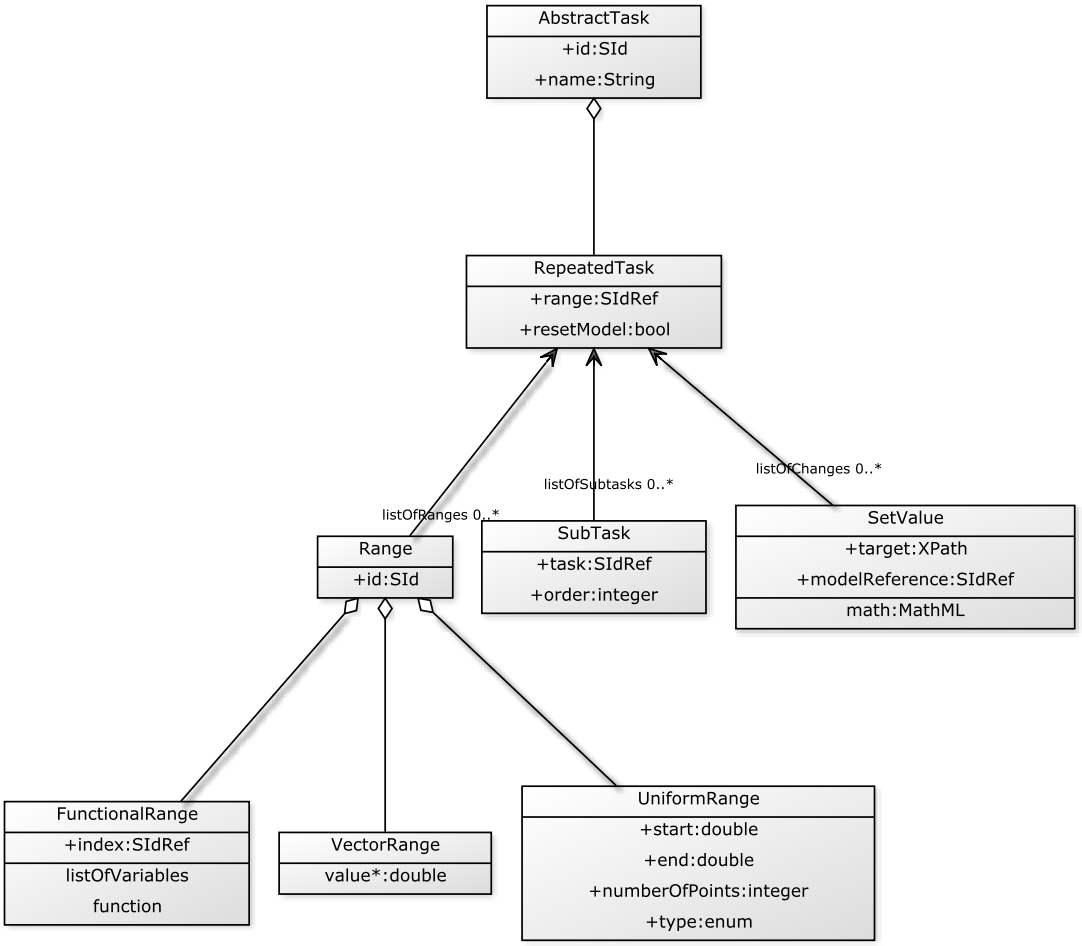

Figure 2.34: *The SED-ML RepeatedTask class*

Table 2.22 shows all attributes and sub-elements for the repeatedTask element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	**description**
metaid^o	page 19
id	page 18
name^o	page 18
range	page 50
resetModel	page 50
modelReference^o	page 23
simulationReference^o	page 24
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
range	page 50
change^o	page 52
subTask^o	page 52

Table 2.22: *Attributes and nested elements for repeatedTask.* xy^o *denotes optional elements and attributes.*

Listing 2.48 shows the use of the `repeatedTask` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<task id="task1" modelReference="model1" simulationReference="simulation1" />

<repeatedTask id="task3" resetModel="false" range="current"
    xmlns:s='http://www.sbml.org/sbml/level3/version1/core'>
  <listOfRanges>
    <vectorRange id="current">
        <value> 1 </value>
        <value> 4 </value>
        <value> 10 </value>
    </vectorRange>
  </listOfRanges>
  <listOfChanges>
     <setValue target="/s:sbml/s:model/s:listOfParameters/s:parameter[@id='w']" modelReference="model1">
       <listOfVariables>
         <variable id="val" name="current range value" target="#current" />
       </listOfVariables>
       <math xmlns="http://www.w3.org/1998/Math/MathML">
         <ci> val </ci>
       </math>
     </setValue>
  </listOfChanges>
  <listOfSubTasks>
    <subTask task="task1" />
  </listOfSubTasks>
</repeatedTask>
```

Listing 2.48: *The* `repeatedTask` *element*

In the example, `task1` is repeated three times, each time with a different value for a model parameter `w`.

2.4.6.1 The `range` attribute

The `repeatedTask` class has a required attribute `range` of type `SId`. It specifies which range defined in the `listOfRanges` this repeated task iterates over. Listing 2.48 shows an example of a `repeatedTask` iterating over a single range comprising the values: `1`, `4` and `10`. If there are multiple ranges in the `listOfRanges`, then only the master range identified by this attribute determines how many iterations there will be in the `repeatedTask`. All other ranges must allow for at least as many iterations as the master range, and will be moved through in lock-step; their values can be used in setValue constructs.

2.4.6.2 The `resetModel` attribute

The `repeatedTask` class has a required attribute `resetModel` of type `boolean`. It specifies whether the model should be reset to the initial state before processing an iteration of the defined subTasks. Here initial state refers to the state of the model as given in the `listOfModels`. In the example in Listing 2.48 the repeated task is not to be reset, so a change is made, `task1` is carried out, another change is made, then `task1` continues from there, another change is applied, and `task1` is carried out a last time.

2.4.6.3 The `listOfRanges`

Ranges represent the iterative element of the nested simulation experiment that provides the collection of values to iterate over. In order to be able to refer to the current value of a range element, an `id` attribute is added. When the value of the `id` attribute is used in a `listOfVariables` within the repeated task class its value is to be replaced with the current value of the range.

There are three different range types permitted in the `listOfRanges`: UniformRange, VectorRange and FunctionalRange. They each inherit from an abstract Range class.

2.4.6.3.1 *Range*

The Range class is abstract and exists solely as the base class for the different types of range. Therefore, a SED-ML document will only contain the derived classes listed below.

2.4.6.3.2 *UniformRange*

The `UniformRange` is quite similar to what is used in the UniformTimeCourse simulation class. This range is defined through four mandatory attributes: `start`, the start value; `end`, the end value and `numberOfPoints` that contains the number of points the range contains. A fourth attribute `type` that can take the values `linear` or `log` determines whether to draw the values logarithmically (with a base of 10) or linearly.

For example:

```
<uniformRange id="current" start="0.0" end="10.0" numberOfPoints="100" type="linear" />
```

Listing 2.49: *The* `UniformRange` *element*

As for UniformTimeCourse, this range will actually produce 101 values uniformly spaced on the interval $[0, 10]$, in ascending order.

The following logarithmic example generates the three values `1`, `10` and `100`.

```
<uniformRange id="current" start="1.0" end="100.0" numberOfPoints="2" type="log" />
```

Listing 2.50: *The* `UniformRange` *element with a logarithmic range.*

2.4.6.3.3 `VectorRange`

A `VectorRange` describes an ordered collection of real values, listing them explicitly within child `value` elements. For example, the range below iterates over the values 1, 4 and 10 in that order.

```
<vectorrange id="current">
    <value> 1 </value>
    <value> 4 </value>
    <value> 10 </value>
</vectorRange>
```

Listing 2.51: *The* `VectorRange` *element*

2.4.6.3.4 `FunctionalRange`

A `FunctionalRange` constructs a range through calculations that determine the next value based on the value(s) of other range(s) or model variables. In this it is quite similar to the ComputeChange element, and shares some of the same child elements. It consists of an optional attribute `range`, two optional elements listOfVariables and listOfParameters, and a required element `function`. and `function`.

The optional attribute `range` may be used as a shorthand to specify the `id` of another Range. The current value of the referenced range may then be used within the function defining this FunctionalRange, just as if that range had been referenced using a variable element, except that the `id` of the range is used directly. In other words, whenever the expression contains a `ci` element that contains the value specified in the `range` attribute, the value of the referenced range is to be inserted.

In the `listOfVariables`, variable elements define identifiers refering to model variables or range values, which may then be used within the `function` expression. These references always retrieve the current value of the model variable or range at the current iteration of the enclosing `repeatedTask`. For a model not being simulated by any `subTask`, the initial state of the model is used.

The `function` encompasses the mathematical expression that is used to compute the value for the functional range at each iteration of the enclosing `repeatedTask`.

For example:

```
<functionalRange id="current" range="index"
    xmlns:s='http://www.sbml.org/sbml/level3/version1/core'>
  <listOfVariables>
    <variable id="w" name="current parameter value" modelReference="model2"
        target="/s:sbml/s:model/s:listOfParameters/s:parameter[@id='w']" />
  </listOfVariables>
  <function>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <times/>
        <ci> w </ci>
        <ci> index </ci>
      </apply>
    </math>
  </function>
</functionalRange>
```

Listing 2.52: *An example of a* `functionalRange` *where a parameter* `w` *of model* `model2` *is multiplied by* `index` *each time it is called.*

Here is another example, this time using the values in a piecewise expression:

```
        <uniformRange id="index" start="0" end="10" numberOfPoints="100" />
        <functionalRange id="current" range="index">
          <function>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
              <piecewise>
```

```
                <piece>
                  <cn> 8 </cn>
                  <apply>
                    <lt />
                    <ci> index </ci>
                    <cn> 1 </cn>
                  </apply>
                </piece>
                <piece>
                  <cn> 0.1 </cn>
                  <apply>
                    <and />
                    <apply>
                      <geq />
                      <ci> index </ci>
                      <cn> 4 </cn>
                    </apply>
                    <apply>
                      <lt />
                      <ci> index </ci>
                      <cn> 6 </cn>
                    </apply>
                  </apply>
                </piece>
                <otherwise>
                  <cn> 8 </cn>
                </otherwise>
              </piecewise>
            </math>
          </function>
        </functionalRange>
```

Listing 2.53: *A* `functionalRange` *element that returns* `8` *if* `index` *is smaller than* `1`*,* `0.1` *if index is between* `4` *and* `6` *and* `8` *otherwise.*

2.4.6.4 The `listOfChanges`

The `listOfChanges` element, when present, contains one or more `setValue` elements. These elements allow the modification of values in the model prior to the next execution of the subTasks.

A `setValue` element inherits from the computeChange class, which allows it to compute arbitrary expressions involving a number of variables and parameters. The element `setValue` adds a mandatory `modelReference` attribute, and two optional attributes `range` and `symbol`.

The value to be changed is identified via the combination of the attributes `modelReference` and either `symbol` or `target`, in order to select an implicit or explicit variable within the referenced model.

As in functionalRange, the atribute `range` may be used as a shorthand for refering to the range whose values will be used to compute a value for the specified model element.

The child `math` contains the expression computing the value by refering to optional parameters, variables or ranges. Again as for functionalRange, variable references always retrieve the current value of the model variable or range at the current iteration of the enclosing `repeatedTask`. For a model not being simulated by any `subTask`, the initial state of the model is used.

```
<listOfChanges>
  <setValue target="/s:sbml/s:model/s:listOfParameters/s:parameter[@id='w']"
            range="current" modelReference="model1">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <ci> current </ci>
    </math>
  </setValue>
</listOfChanges>
```

Listing 2.54: *A* `setValue` *element setting* `w` *to the values of the range with id* `current`*.*

2.4.6.5 The `listOfSubTasks`

The `listOfSubTasks` contains one or more `subTask` elements that specify what simulations are to be performed by the `RepeatedTask`. All `subTask`s have to be carried out sequentially, each continuing from the current model state (i.e. as at the end of the previous `subTask`, assuming it simulates the same model), and with their results concatenated (thus appearing identical to a single complex simulation). The `subTask` itself has one required attribute `task` that references the `id` of another task defined in the `listOfTasks`. The order in which to run multiple `subTask`s must be specified using `order` attributes on the `subTask` elements; if these are omitted the ordering is implementation defined. In order to establish that one `subTask` should be carried out before another its `order` attribute has to have a lower number (c.f. Listing 2.55).

```
<listOfSubTasks>
  <subTask task="task1" order="2"/>
  <subTask task="task2" order="1"/>
</listOfSubTasks>
```

Listing 2.55: *The* `subTask` *element. In this example the task* `task2` *has to be carried out before* `task1`*.*

2.4.7 DataGenerator

The DataGenerator class prepares the raw simulation results for later output (Figure 2.35). It encodes the post-processing to be applied to the simulation data. The post-processing steps could be anything, from simple normalisations of data to mathematical calculations.

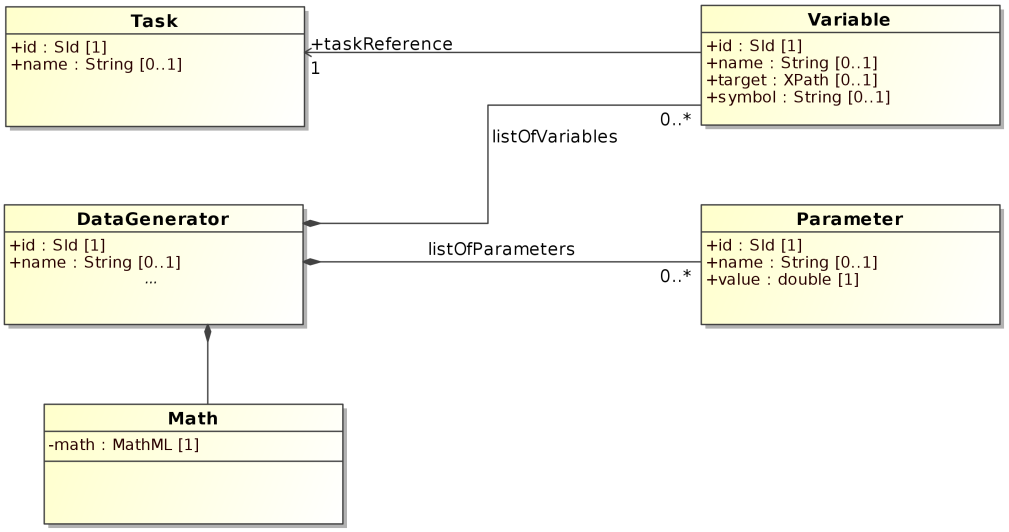

Figure 2.35: *The SED-ML DataGenerator class*

Each instance of the DataGenerator class is identifiable within the experiment by its unambiguous id. It can be further characterised by an optional name. The related Math class contains a mathML expression for the calculation of the data generator. Mathematical functions available for the specification of data generators are given in Section 2.2.1 on page 14. Variable and Parameter instances can be used to encode the mathematical expression.

Table 2.23 shows all attributes and sub-elements for the dataGenerator element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18

sub-elements	description
math	page 41
notes^o	page 19
annotation^o	page 20
variable^o	page 25
parameter^o	page 27

Table 2.23: *Attributes and nested elements for dataGenerator.* xy^o *denotes optional elements and attributes.*

Listing 2.56 on the next page shows the use of the `dataGenerator` element in a SED-ML file as defined

by the SED-ML Level 1 Version 2 XML Schema.

```
<listOfDataGenerators>
 <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 2.56: *Definition of two* `dataGenerator` *elements,* 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.

2.4.8 Output

The Ouput class describes how the results of a simulation should be presented to the user (Figure 2.36).

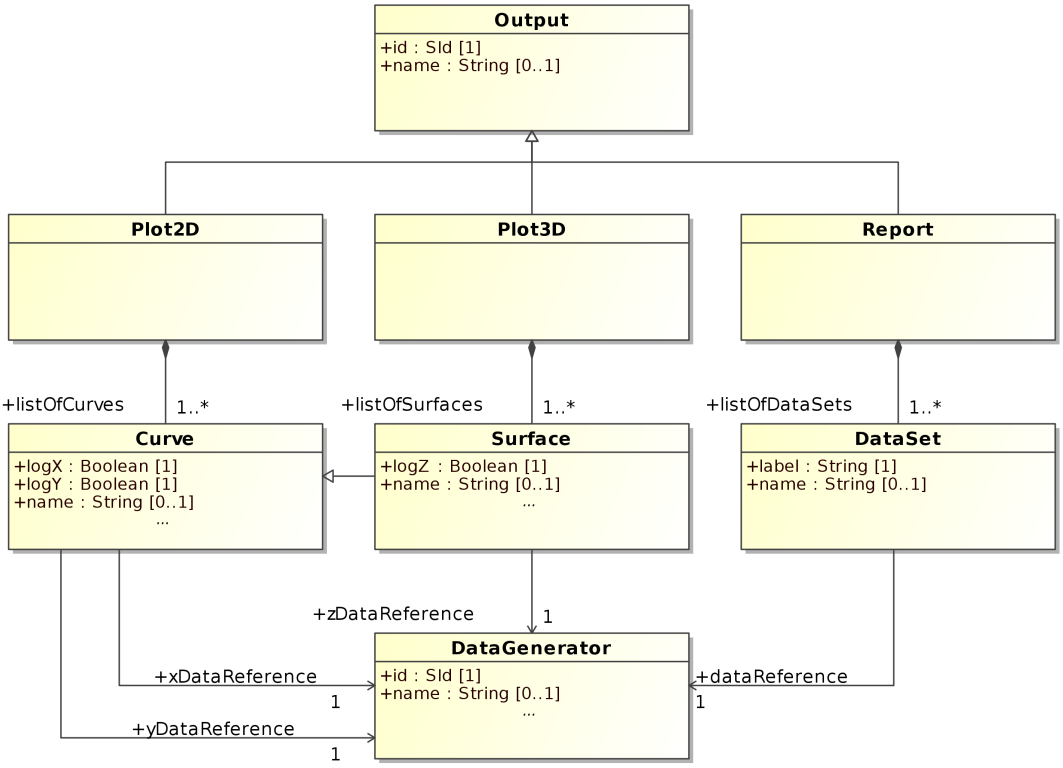

Figure 2.36: *The SED-ML Output class*

It does not contain the data itself, but the type of output and the data generators used to produce a particular output.

The types of output pre-defined in SED-ML Level 1 Version 2 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.

Table 2.24 shows all attributes and sub-elements for the output element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
plot2D^o	page 55
plot3D^o	page 56
report^o	page 56

Table 2.24: *Attributes and nested elements for output.* xy^o *denotes optional elements and attributes.*

2.4.8.1 Plot2D

A 2 dimensional plot (Figure 2.37) contains a number of curve definitions.

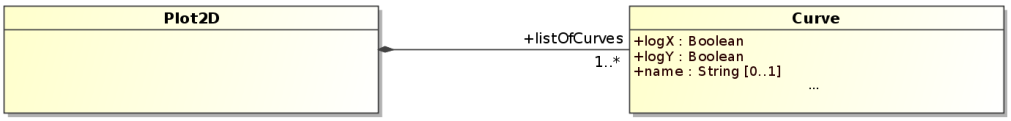

Figure 2.37: *The SED-ML Plot2D class*

Table 2.25 shows all attributes and sub-elements for the plot2D element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
curve	page 57

Table 2.25: *Attributes and nested elements for plot2D.* xy^o *denotes optional elements and attributes.*

Listing 2.57 shows the use of the `listOfCurves` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

```
  </listOfCurves>
</plot2D>
```

Listing 2.57: *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 2.4.9.1 on page 57.

2.4.8.2 `Plot3D`

A 3 dimensional plot (Figure 2.38) contains a number of surface definitions.

Figure 2.38: *The SED-ML Plot3D class*

Table 2.26 shows all attributes and sub-elements for the plot3D element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
surface	page 59

Table 2.26: *Attributes and nested elements for plot3D.* xy^o *denotes optional elements and attributes.*

Listing 2.58 shows the use of the `plot3D` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<plot3D>
 <listOfSurfaces>
  <surface>
   [SURFACE DEFINITION]
  </surface>
  [FURTHER SURFACE DEFINITIONS]
 </listOfSurfaces>
</plot3D>
```

Listing 2.58: *The `plot3D` element with the nested `listOfSurfaces` element*

The example defines a surface for the 3 dimensional plot. The surface definition follows in Section 2.4.9.2 on page 59.

2.4.8.3 The Report class

The Report class defines a data table consisting of several single instances of the DataSet class (Figure 2.39 on the following page). Its output returns the simulation result in actual *numbers*. The particular columns of the report table are defined by creating an instance of the DataSet class for each column.

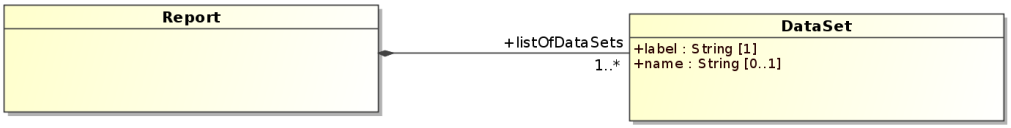

Figure 2.39: *The SED-ML Report class*

Table 2.27 shows all attributes and sub-elements for the report element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
dataSet	page 60

Table 2.27: *Attributes and nested elements for report.* xy^o *denotes optional elements and attributes.*

Listing 2.59 shows the use of the `listOfDataSets` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.59: *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.

The encoding of simulation results is outside the scope of SED-ML, but other efforts exist, for example the *Systems Biology Result Markup Language* (SBRML, [Dada et al., 2010]).

2.4.9 Output components

2.4.9.1 Curve

One or more instances of the Curve class define a 2D plot. 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.

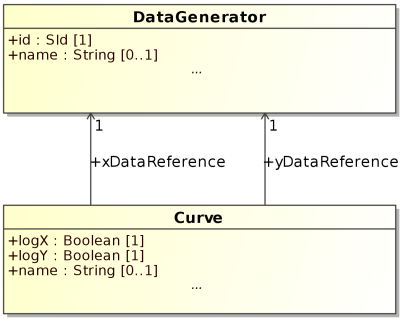

Figure 2.40: *The SED-ML Curve class*

Table 2.28 shows all attributes and sub-elements for the curve element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
logX	page 58
xDataReference	page 58
logY	page 58
yDataReference	page 58
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.28: *Attributes and nested elements for curve.* xy^o *denotes optional elements and attributes.*

Listing 2.60 shows the use of the `curve` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.60: *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.

2.4.9.1.1 *logX*

logX is a required attribute of the Curve class and defines whether or not the data output on the x-axis is logarithmic. The data type of logX is `boolean`. To make the output on the x-axis of a plot logarithmic, logX must be set to "true", as shown in the sample Listing 2.60.

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

2.4.9.1.2 *logY*

logY is a required attribute of the Curve class and defines whether or not the data output on the y-axis is logarithmic. The data type of logY is `boolean`. To make the output on the y-axis of a plot logarithmic, logY must be set to "true", as shown in the sample Listing 2.60.

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

2.4.9.1.3 *xDataReference*

The xDataReference is a mandatory attribute of the Curve object. Its 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 2.60. xDataReference is also used in the definition of the x-axis of a Surface in a 3D Plot.

2.4.9.1.4 *yDataReference*

The yDataReference is a mandatory attribute of the Curve object. Its 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 2.60. yDataReference is also used in the definition of the y-axis of a Surface in a 3D Plot.

2.4.9.2 `Surface`

A surface is a three-dimensional figure representing a simulation result (Figure 2.41).

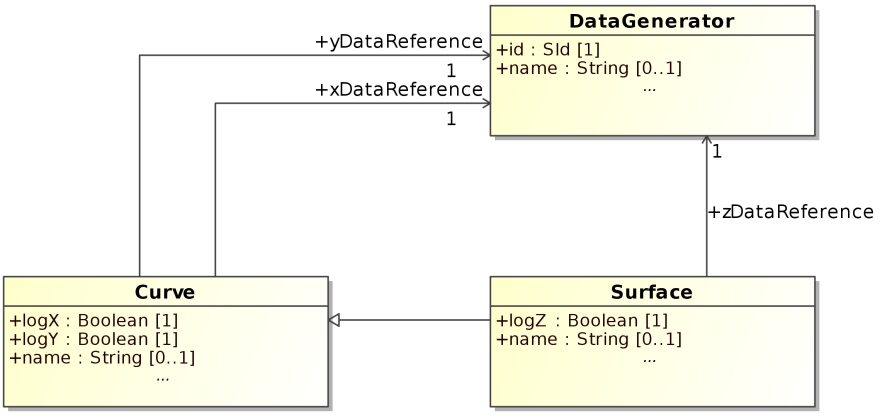

Figure 2.41: *The SED-ML Surface class*

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 2.29 shows all attributes and sub-elements for the surface element as defined by the SED-ML Level 1 Version 2 XML Schema. Listing 2.61 shows the use of the `surface` element in a SED-ML file

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
logX	page 58
xDataReference	page 58
logY	page 58
yDataReference	page 58
logZ	page 60
zDataReference	page 60
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.29: *Attributes and nested elements for surface.* xy^o *denotes optional elements and attributes.*

as defined by the SED-ML Level 1 Version 2 XML Schema.

```
<listOfSurfaces>
  <surface id="s1" name="surface" xDataReference="dg1" yDataReference="dg2" zDataReference="dg3"
   logX="true"  logY="false" logZ="false" />
  [FURTHER SURFACE DEFINITIONS]
</listOfSurfaces>
```

Listing 2.61: *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.

2.4.9.2.1 *logZ*

logZ is a required 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`. 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 2.61.

2.4.9.2.2 *zDataReference*

The zDataReference is a mandatory attribute of the Surface object. Its 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 2.61 on page 59.

2.4.9.3 *DataSet*

The DataSet class holds definitions of data to be used in the Report class (Figure 2.42).

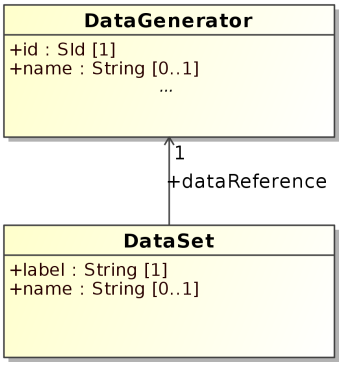

Figure 2.42: *The SED-ML DataSet class*

Data sets are labeled references to instances of the DataGenerator class.

Table 2.30 shows all attributes and sub-elements for the dataSet element as defined by the SED-ML Level 1 Version 2 XML Schema.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
dataReference	page 61
label	page 61
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.30: *Attributes and nested elements for dataSet.* xy^o *denotes optional elements and attributes.*

2.4.9.3.1 label

Each data set in a Report does have to carry an unambiguous label. A label is a human readable descriptor of a data set for use in a report. For example, for a tabular data set of time series results, the label could be the column heading.

2.4.9.3.2 dataReference

The dataReference attribute contains the ID of a dataGenerator element and as such represents a link to it. The data produced by that particular data generator fills the according data set in the report. Listing 2.62 shows the use of the `dataSet` element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

Listing 2.62: *The SED-ML* `dataSet` *element, defining a data set containing the result of the referenced task*

3. 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.

Thanks to David Nickerson for feedback and help with Example C2.

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

A. SED-ML UML Overview

Figure A.1 shows the complete UML diagram of the SED-ML. It gives the full picture of all implemented classes (see the XML Schema definition on page 64).

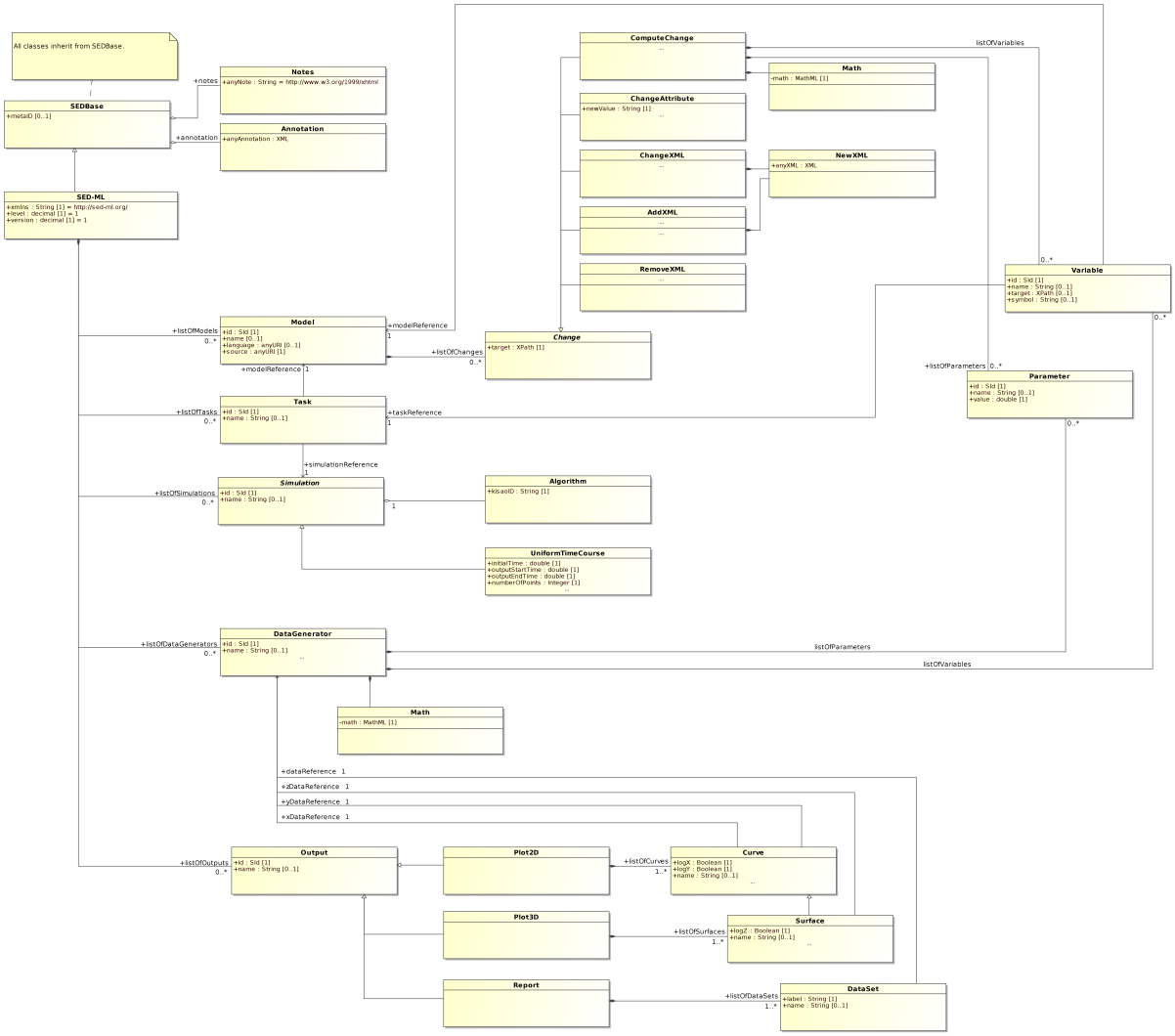

Figure A.1: *The SED-ML UML class diagram*

B. XML Schema

Listing B.1 shows the full SED-ML XML Schema. The code is commented inline.

```
<xs:schema targetNamespace="http://sed-ml.org/" xmlns="http://sed-ml.org/"
    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="sedml-mathml.xsd" />


    <xs:simpleType name="SId">
        <xs:annotation>
            <xs:documentation>
                The type SId is used throughout SED-ML as the
                type
                of the 'id' attributes on model elements.
            </xs:documentation>
        </xs:annotation>
        <xs:restriction base="xs:string">
            <xs:pattern value="(_|[a-z]|[A-Z])(_|[a-z]|[A-Z]|[0-9])*" />
        </xs:restriction>
    </xs:simpleType>

    <!--  attribute group for elements with ID/name att -->
    <xs:attributeGroup name="idGroup">
        <xs:attribute name="id" use="required" type="SId"></xs:attribute>
        <xs:attribute name="name" use="optional" type="xs:string"></xs:attribute>
    </xs:attributeGroup>



    <!-- SED Base class -->
    <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>
        <!--
            This must be a variable-type identifier ,i.e., (Letter | '_')
            (NCNameChar)* that is unique in the document.
        -->
        <xs:attribute name="metaid" type="xs:ID" use="optional"></xs:attribute>
    </xs:complexType>
    <xs:element name="sedML">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfSimulations" minOccurs="0" />
                        <xs:element ref="listOfModels" minOccurs="0" />
                        <xs:element ref="listOfTasks" minOccurs="0" />
                        <xs:element ref="listOfDataGenerators" minOccurs="0" />
                        <xs:element ref="listOfOutputs" minOccurs="0" />
                    </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>
    <!-- notes and annotations -->
    <xs:element name="notes">
```

```
        <xs:complexType>
            <xs:sequence>
                <xs:any namespace="http://www.w3.org/1999/xhtml"
                    processContents="skip" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="annotation">
        <xs:complexType>
            <xs:sequence>
                <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- KiSAO ID type -->
    <xs:simpleType name="KisaoType">
        <xs:restriction base="xs:string">
            <xs:pattern value="KISAO:[0-9][0-9][0-9][0-9][0-9][0-9][0-9]" />
        </xs:restriction>
    </xs:simpleType>

    <!-- global element declarations -->
    <xs:element name="variable">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <!--  at least one of taskReference or modelReference must be set -->
                    <xs:attribute name="taskReference" type="SId" use="optional" />
                    <xs:attribute name="modelReference" type="SId" 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:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="parameter">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attributeGroup ref="idGroup" />
                    <xs:attribute name="value" type="xs:double" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="algorithm">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="kisaoID" type="KisaoType" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="uniformTimeCourse">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="algorithm" />
                    </xs:sequence>
                    <xs:attributeGroup ref="idGroup" />
                    <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:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="task">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="simulationReference" type="SId"
                        use="required" />

                    <xs:attribute name="modelReference" type="SId" use="required" />
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
```

```
        </xs:complexType>
    </xs:element>
    <xs:element name="plot2D">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfCurves" minOccurs="0" />
                    </xs:sequence>
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="plot3D">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfSurfaces" minOccurs="0" />
                    </xs:sequence>
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="report">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfDataSets" minOccurs="0" />
                    </xs:sequence>
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="model">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <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:anyURI" use="required" />
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <!-- listOf elements -->
    <xs:element name="listOfVariables">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="variable" minOccurs="0" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfParameters">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="parameter" minOccurs="0" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfTasks">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="task" minOccurs="0" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfSimulations">
```

```
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="uniformTimeCourse" minOccurs="0"
                            maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfOutputs">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence minOccurs="0">
                        <xs:choice minOccurs="0" maxOccurs="unbounded">
                            <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:choice>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfModels">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="model" minOccurs="0" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfDataGenerators">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="dataGenerator" minOccurs="0"
                            maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfCurves">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="curve" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfSurfaces">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="surface" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="listOfDataSets">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="dataSet" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- change -->
    <xs:element name="listOfChanges">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence minOccurs="0">
                        <xs:choice minOccurs="0" maxOccurs="unbounded">
```

```
                            <xs:element ref="changeAttribute" minOccurs="0"
                                maxOccurs="unbounded" />
                            <xs:element ref="changeXML" minOccurs="0" maxOccurs="unbounded" />
                            <xs:element ref="addXML" minOccurs="0" maxOccurs="unbounded" />
                            <xs:element ref="removeXML" minOccurs="0" maxOccurs="unbounded" />
                            <xs:element ref="computeChange" minOccurs="0"
                                maxOccurs="unbounded" />
                        </xs:choice>
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="newXML">
        <xs:complexType>
            <xs:sequence>
                <xs:any processContents="skip" minOccurs="1" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="changeAttribute">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="target" use="required" type="xs:token" />
                    <xs:attribute name="newValue" type="xs:string" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="changeXML">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="newXML" />
                    </xs:sequence>
                    <xs:attribute name="target" use="required" type="xs:token" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="addXML">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="newXML" />
                    </xs:sequence>
                    <xs:attribute name="target" use="required" type="xs:token" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="removeXML">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="target" use="required" type="xs:token" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="computeChange">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfVariables" minOccurs="0" />
                        <xs:element ref="listOfParameters" minOccurs="0" />
                        <xs:element ref="math:math" />
                    </xs:sequence>
                    <xs:attribute name="target" use="required" type="xs:token" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <!-- data generator -->
    <xs:element name="dataGenerator">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfVariables" minOccurs="0" />
                        <xs:element ref="listOfParameters" minOccurs="0" />
                        <xs:element ref="math:math" />
                    </xs:sequence>
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
```

```
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="curve">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attributeGroup ref="idGroup" />
                    <xs:attribute name="yDataReference" type="SId" use="required" />
                    <xs:attribute name="xDataReference" type="SId" use="required" />

                    <xs:attribute name="logY" use="required" type="xs:boolean" />
                    <xs:attribute name="logX" use="required" type="xs:boolean" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="surface">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attributeGroup ref="idGroup" />
                    <xs:attribute name="yDataReference" type="SId" use="required" />
                    <xs:attribute name="xDataReference" type="SId" use="required" />
                    <xs:attribute name="zDataReference" type="SId" use="required" />
                    <xs:attribute name="logY" use="required" type="xs:boolean" />
                    <xs:attribute name="logX" use="required" type="xs:boolean" />
                    <xs:attribute name="logZ" use="required" type="xs:boolean" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="dataSet">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="dataReference" type="SId" use="required"></xs:attribute>
                    <xs:attribute name="label" use="required" type="xs:string" />
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Listing B.1: *The SED-ML XML Schema definition*

C. Examples

This appendix present a few examples SED-ML uses. These examples are only illustrative and do not intend to demonstrate the full capabilities of SED-ML. Please read the specification for a for a more comprehensive view (Chapter 2).

The current examples make use of models encoded in SBML and CellML. It is important to remember that SED-ML is not restricted to those formats, but can be used with models encoded in many formats, so long as they are serialized in XML. A list of formats known to have been used, at least tentatively, with SED-ML is available on http://sed-ml.org/.

C.1 Le Loup Model (SBML)

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).

This model is referenced by its SED-ML ID `model1` and refers to the model with the MIRIAM URN `urn:miriam:biomodels.db:BIOMD0000000021`. Software applications interpreting this example know how to dereference this URN and access the model in BioModels Database [Le Novère et al., 2006].

A second model is defined in l. 11 of the example, using `model1` as a source and applying even further changes to it, in this case updating two model parameters.

One simulation setup is defined in the `listOfSimulations`. It is a `uniformTimeCourse` over 380 time units, providing 1000 output points. The algorithm used is the CVODE solver, as denoted by the KiSAO ID `KiSAO:0000019`.

A number of `dataGenerator`s are defined in ll. 23-62. Those are the prerequisite for defining the outputs of the simulation. The first dataGenerator named `time` collects the simulation time. `tim1` in l. 31 maps on the `Mt` entity in the model that is used in `task1` which here is the model with ID `model1`. The dataGenerator named `per-tim1` in l. 39 maps on the `Cn` entity in `model1`. Finally the fourth and fifth dataGenerators map on the `Mt` and `per-tim` entity respectively in the updated model with ID `model2`.

The `output` defined in the experiment consists of three 2D plots. The first plot has two different curves (ll. 65-70) and provides the time course of the simulation using the tim mRNA concentrations from both simulation experiments. The second plot shows the `per-tim` concentration against the `tim` concentration for the oscillating model. And the third plot shows the same plot for the chaotic model. The resulting three plots are shown in Figure C.1.

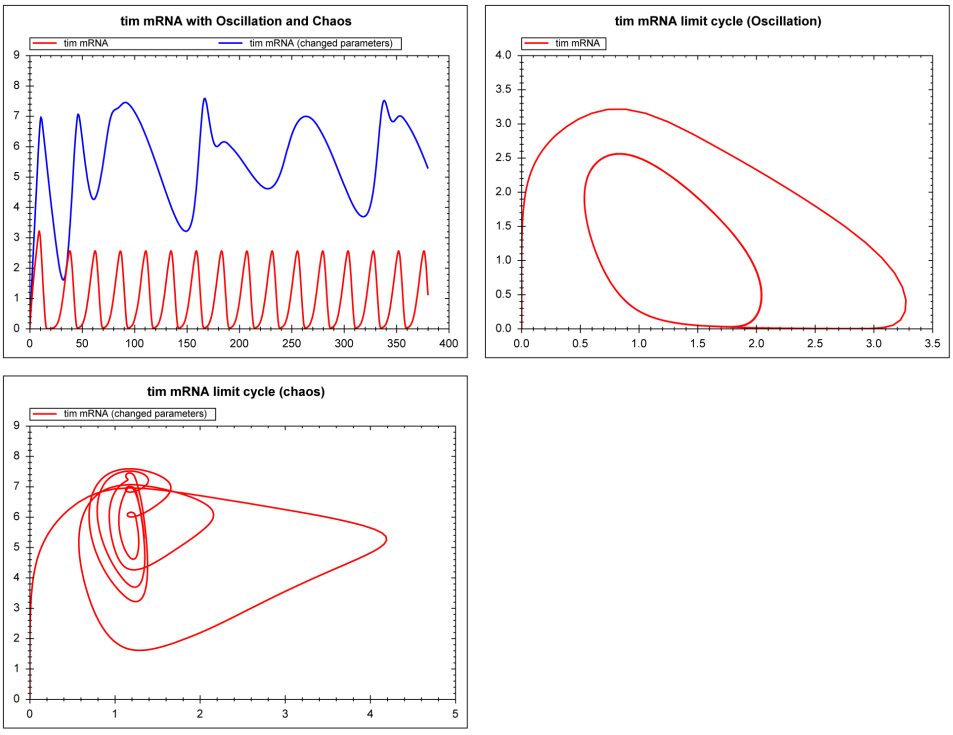

Figure C.1: *The simulation result gained from the simulation description given in listing C.1*

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!-- Written by libSedML v1.1.4092.21172 see http://libsedml.sf.net -->
3 <sedML xmlns="http://sed-ml.org/" level="1" version="1">
4   <listOfSimulations>
5     <uniformTimeCourse id="simulation1" initialTime="0" outputStartTime="0" outputEndTime="380"
          numberOfPoints="1000">
6       <algorithm kisaoID="KISAO:0000019" />
```

```
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" name="Circadian Oscillations" language="urn:sedml:language:sbml" source="
        urn:miriam:biomodels.db:BIOMD0000000021" />
    <model id="model2" name="Circadian Chaos" language="urn:sedml:language:sbml" source="model1">
      <listOfChanges>
        <changeAttribute target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id=&quot;
            V_mT&quot;]/@value" newValue="0.28" />
        <changeAttribute target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id=&quot;
            V_dT&quot;]/@value" newValue="4.8" />
      </listOfChanges>
    </model>
  </listOfModels>
  <listOfTasks>
    <task id="task1" modelReference="model1" simulationReference="simulation1" />
    <task id="task2" modelReference="model2" simulationReference="simulation1" />
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time" name="time">
      <listOfVariables>
        <variable id="t" taskReference="task1" symbol="urn:sedml:symbol:time" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> t </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="tim1" name="tim mRNA">
      <listOfVariables>
        <variable id="v1" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
            sbml:species[@id='Mt']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> v1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="per_tim1" name="nuclear PER-TIM complex">
      <listOfVariables>
        <variable id="v1a" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
            sbml:species[@id='Cn']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> v1a </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="tim2" name="tim mRNA (changed parameters)">
      <listOfVariables>
        <variable id="v2" taskReference="task2" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
            sbml:species[@id='Mt']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> v2 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="per_tim2" name="nuclear PER-TIM complex">
      <listOfVariables>
        <variable id="v2a" taskReference="task2" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
            sbml:species[@id='Cn']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> v2a </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="tim mRNA with Oscillation and Chaos">
      <listOfCurves>
        <curve id="c1" logX="false" logY="false" xDataReference="time" yDataReference="tim1" />
        <curve id="c2" logX="false" logY="false" xDataReference="time" yDataReference="tim2" />
      </listOfCurves>
    </plot2D>
   <plot2D id="plot2" name="tim mRNA limit cycle (Oscillation)">
      <listOfCurves>
        <curve id="c3" logX="false" logY="false" xDataReference="per_tim1" yDataReference="tim1" />
      </listOfCurves>
    </plot2D>
   <plot2D id="plot3" name="tim mRNA limit cycle (chaos)">
      <listOfCurves>
        <curve id="c4" logX="false" logY="false" xDataReference="per_tim2" yDataReference="tim2" />
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing C.1: *LeLoup Model Simulation Description in SED-ML*

C.2 Le Loup Model (CellML)

The following example provides a SED-ML description for the simulation of the model based on the publication by Leloup, Gonze and Goldbeter "Limit Cycle Models for Circadian Rhythms Based on Transcriptional Regulation in Drosophila and Neurospora" (PubMed ID: 10643740). Whereas the previous example used SBML to encode the simulation experiment, here the model 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/7606a47e222bc3b3d9117baa08d2e7246d67eedd/leloup_gonze_goldbeter_1999_a.cellml`, ll. 14).

A second model is defined in l. 15 of the example, using `model1` as a source and applying even further changes to it, in this case updating two model parameters.

One simulation setup is defined in the `listOfSimulations`. It is a `uniformTimeCourse` over 380 time units, using 1000 simulation points. The algorithm used is the CVODE solver, as denoted by the KiSAO ID `KiSAO:0000019`.

A number of `dataGenerator`s are defined in ll. 27-73. Those are the prerequisite for defining the output of the simulation. The dataGenerator named `tim1` in l. 37 maps on the `Mt` entity in the model that is used in `task1`, which here is the model with ID `model1`. The dataGenerator named `per-tim` in l. 46 maps on the `CN` entity in `model1`. Finally the fourth and fifth dataGenerators map on the `Mt` and `per-tim` entity respectively in the updated model with ID `model2`.

The `output` defined in the experiment consists of three 2D plots (ll. 74-91). They reproduce the same output as the previous example.

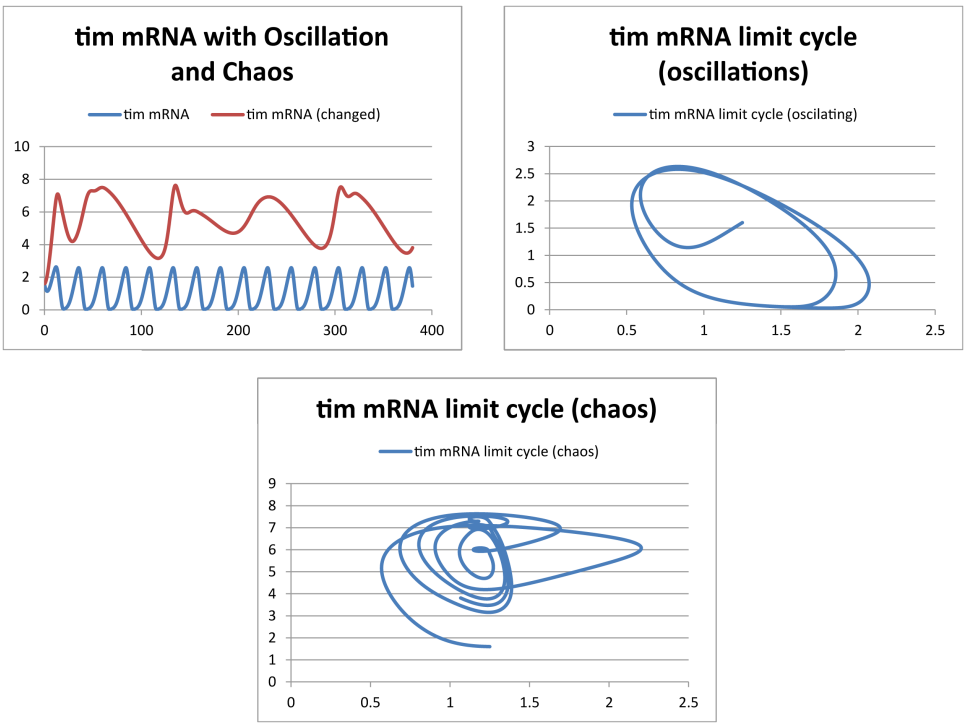

Figure C.2: *The simulation result gained from the simulation description given in listing C.2*

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <sedML  xmlns="http://sed-ml.org/"
3         xmlns:math="http://www.w3.org/1998/Math/MathML" level="1" version="1">
4   <notes><p xmlns="http://www.w3.org/1999/xhtml">Comparing Limit Cycles and strange attractors for
5          oscillation in Drosophila</p></notes>
6   <listOfSimulations>
7     <uniformTimeCourse id="simulation1"
8      initialTime="0" outputStartTime="0" outputEndTime="380"
9      numberOfPoints="1000" >
10      <algorithm kisaoID="KISAO:0000019"/>
```

```
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
   <model id="model1" name="Circadian Oscillations" language="urn:sedml:language:cellml" source="http://
        models.cellml.org/workspace/leloup_gonze_goldbeter_1999/@@rawfile/7606
        a47e222bc3b3d9117baa08d2e7246d67eedd/leloup_gonze_goldbeter_1999_a.cellml"/>
   <model id="model2" name="Circadian Chaos" language="urn:sedml:language:cellml" source="model1">
    <listOfChanges>
     <changeAttribute target="/cellml:model/cellml:component[@name='MT']/cellml:variable[@name='vmT']/
          @initial_value" newValue="0.28"/>
     <changeAttribute target="/cellml:model/cellml:component[@name='T2']/cellml:variable[@name='vdT']/
          @initial_value" newValue="4.8"/>
    </listOfChanges>
   </model>
  </listOfModels>

   <listOfTasks>
     <task id="task1" name="Limit Cycle" modelReference="model1" simulationReference="simulation1"/>
     <task id="task2" name="Strange attractors" modelReference="model2" simulationReference="simulation1"/
          >
   </listOfTasks>
   <listOfDataGenerators>
     <dataGenerator id="time" name="time">
       <listOfVariables>
         <variable id="t" taskReference="task1" target="/cellml:model/cellml:component[@name='environment
              ']/cellml:variable[@name='time']" />
       </listOfVariables>
       <math:math>
         <math:ci>t</math:ci>
       </math:math>
     </dataGenerator>

     <dataGenerator id="tim1" name="tim mRNA">
       <listOfVariables>
         <variable id="v0" taskReference="task1" target="/cellml:model/cellml:component[@name='MT']/
              cellml:variable[@name='MT']" />
       </listOfVariables>
       <math:math>
         <math:ci>v0</math:ci>
       </math:math>
     </dataGenerator>

     <dataGenerator id="per_tim" name="nuclear PER-TIM complex">
       <listOfVariables>
         <variable id="v1" taskReference="task1" target="/cellml:model/cellml:component[@name='CN']/
              cellml:variable[@name='CN']" />
       </listOfVariables>
       <math:math>
         <math:ci>v1</math:ci>
       </math:math>
     </dataGenerator>

     <dataGenerator id="tim2" name="tim mRNA (changed parameters)">
       <listOfVariables>
         <variable id="v2" taskReference="task2" target="/cellml:model/cellml:component[@name='MT']/
              cellml:variable[@name='MT']" />
       </listOfVariables>
         <math:math>
           <math:ci>v2</math:ci>
         </math:math>
     </dataGenerator>

     <dataGenerator id="per_tim2" name="nuclear PER-TIM complex">
       <listOfVariables>
         <variable id="v3" taskReference="task2" target="/cellml:model/cellml:component[@name='CN']/
              cellml:variable[@name='CN']" />
       </listOfVariables>
       <math:math>
         <math:ci>v3</math:ci>
       </math:math>
     </dataGenerator>
   </listOfDataGenerators>

   <listOfOutputs>
     <plot2D id="plot1" name="tim mRNA with Oscillation and Chaos">
       <listOfCurves>
         <curve id="c1" logX="false" logY="false" xDataReference="time" yDataReference="tim1" />
         <curve id="c2" logX="false" logY="false" xDataReference="time" yDataReference="tim2" />
       </listOfCurves>
     </plot2D>
    <plot2D id="plot2" name="tim mRNA limit cycle (Oscillation)">
       <listOfCurves>
         <curve id="c3" logX="false" logY="false" xDataReference="per_tim" yDataReference="tim1" />
       </listOfCurves>
     </plot2D>
    <plot2D id="plot3" name="tim mRNA limit cycle (chaos)">
       <listOfCurves>
         <curve id="c4" logX="false" logY="false" xDataReference="per_tim2" yDataReference="tim2" />
       </listOfCurves>
```

```
90      </plot2D>
91    </listOfOutputs>
92  </sedML>
```

Listing C.2: *LeLoup Model Simulation Description in SED-ML*

C.3 The IkappaB-NF-kappaB signaling module (SBML)

The following example provides a SED-ML description for the simulation of the IkappaB-NF-kappaB signaling module based on the publication by Hoffmann, Levchenko, Scott and Baltimore "The IkappaB-NF-kappaB signaling module: temporal control and selective gene activation. " (PubMed ID: 12424381)

This model is referenced by its SED-ML ID `model1` and refers to the model with the MIRIAM URN `urn:miriam:biomodels.db:BIOMD0000000140`. Software applications interpreting this example know how to dereference this URN and access the model in BioModels Database [Le Novère et al., 2006].

The simulation description specifies one simulation `simulation1`, which is a uniform timecourse simulation that simulates the model for 41 hours. `task1` then applies this simulation to the model.

As output this simulation description collects four parameters: `Total_NFkBn`, `Total_IkBbeta`, `Total_IkBeps` and `Total_IkBalpha`. These variables are to be plotted against the simulation time and displayed in four separate plots, as shown in Figure C.3.

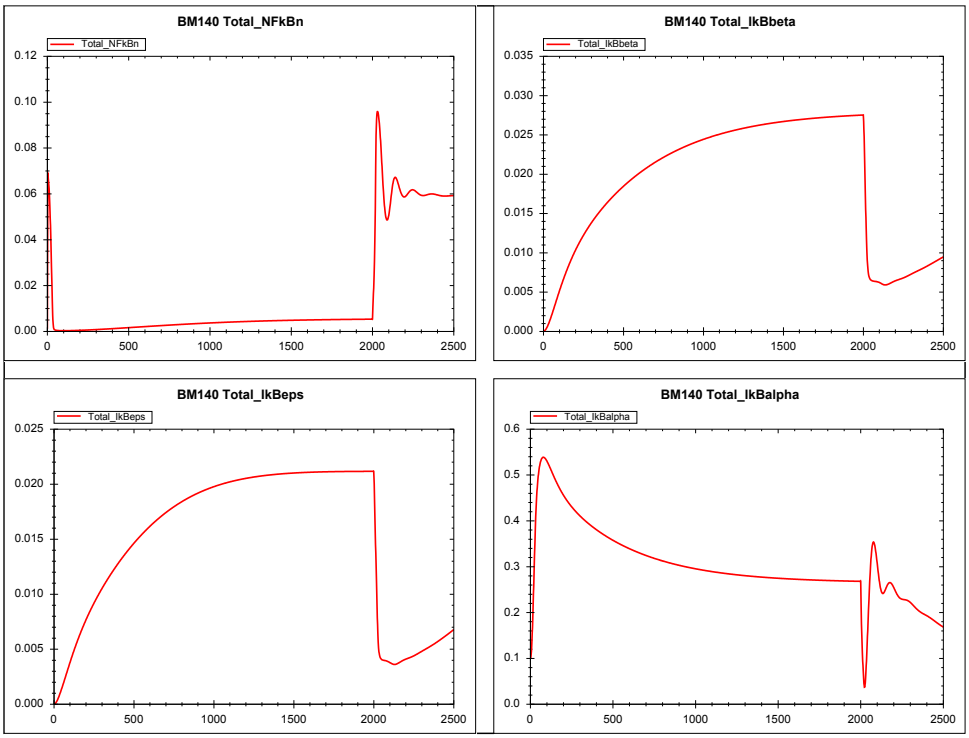

Figure C.3: *The simulation result gained from the simulation description given in listing C.3*

The SED-ML description of the simulation experiment is given in listing C.3.

```
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns="http://sed-ml.org/" level="1" version="1">
  <listOfSimulations>
    <uniformTimeCourse id="simulation1"
    initialTime="0" outputStartTime="0" outputEndTime="2500"
    numberOfPoints="1000" >
      <algorithm kisaoID="KISAO:0000019"/>
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.db:BIOMD0000000140
        "/>
  </listOfModels>
  <listOfTasks>
    <task id="task1" modelReference="model1"
    simulationReference="simulation1"/>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time" name="time">
      <listOfVariables>
        <variable id="time1" taskReference="task1" symbol="urn:sedml:symbol:time"/>
      </listOfVariables>
```

```
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci>time1</ci>
      </math>
    </dataGenerator>
    <dataGenerator id="Total_NFkBn" name="Total_NFkBn">
      <listOfVariables>
        <variable id="Total_NFkBn1" taskReference="task1"
        target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='Total_NFkBn']"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci>Total_NFkBn1</ci>
      </math>
    </dataGenerator>
    <dataGenerator id="Total_IkBbeta" name="Total_IkBbeta">
      <listOfVariables>
        <variable id="Total_IkBbeta1" taskReference="task1"
        target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='Total_IkBbeta']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci>Total_IkBbeta1</ci>
      </math>
    </dataGenerator>
    <dataGenerator id="Total_IkBeps" name="Total_IkBeps">
      <listOfVariables>
        <variable id="Total_IkBeps1" taskReference="task1"
        target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='Total_IkBeps']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci>Total_IkBeps1</ci>
      </math>
    </dataGenerator>
    <dataGenerator id="Total_IkBalpha" name="Total_IkBalpha">
      <listOfVariables>
        <variable id="Total_IkBalpha1" taskReference="task1"
        target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='Total_IkBalpha']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci>Total_IkBalpha1</ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="BM140 Total_NFkBn">
      <listOfCurves>
        <curve  id="c1" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_NFkBn" />
      </listOfCurves>
    </plot2D>
    <plot2D id="plot2" name="BM140 Total_IkBbeta">
      <listOfCurves>
        <curve  id="c2" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_IkBbeta" />
      </listOfCurves>
    </plot2D>
    <plot2D id="plot3" name="BM140 Total_IkBeps">
      <listOfCurves>
        <curve  id="c3" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_IkBeps" />
      </listOfCurves>
    </plot2D>
    <plot2D id="plot4" name="BM140 Total_IkBalpha">
      <listOfCurves>
        <curve id="c4" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_IkBalpha" />
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing C.3: *IkappaB-NF-kappaB signaling Model Simulation Description in SED-ML*

C.4 Examples for Simulation Experiments involving repeatedTasks

The repeatedTask introduced in Level 1 Version 2 makes it possible to encode a large number of different simulation experiments. In this section several simulation experiment are presented that use the repeated tasks construct.

C.4.1 One dimensional steady state parameter scan

Here the repeated task calls out to a oneStep task (performing a steady state computation). Each time a parameter is carried in order to collect different responses.

In the description below the range to be used in the setValue construct use of the `range` attribute.

C.4.2 One dimensional steady state parameter scan

D. The COMBINE archive

A COMBINE archive is a single file containing the various documents (and in the future, references to documents), necessary for the description of a model and all associated data and procedures. This includes for instance, but not limited to, simulation experiment descriptions in SED-ML, all models needed to run the simulations in SBML and their graphical representations in SBGN-ML. It is a convenient alternative if a model source URI cannot be resolved, or if an end-user is offline.

The SED-ML archive described in appendix D of the SED-ML Level 1 Version 1 specification formed the basis for the COMBINE archive with contributions from the SED-ML and COMBINE communities.

The COMBINE archive is described at: http://co.mbine.org/documents/archive

E. 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.

E.1 Conventions

The Business 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 (Figure E.1). Each collapsed subprocess has a corresponding expanded subprocess definition.

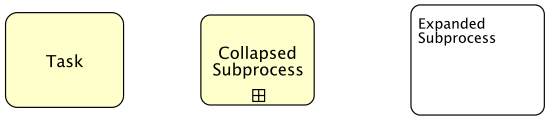

Figure E.1: *BPMN activities: task, collapsed process, expanded subprocess*

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 split. Graphically, a gateway is represented as a diamond.

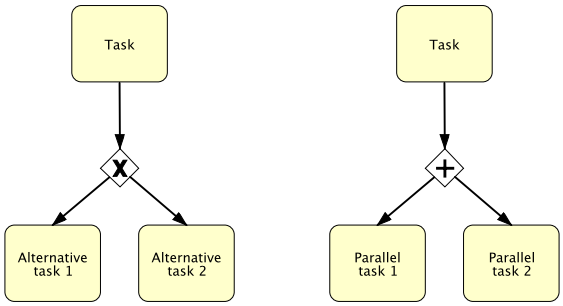

Figure E.2: *BPML gateway types: Exclusive (left), parallel (right)*

While there exist a number of different gateway types [White et al., 2004, p. 93], the SED-ML specification only uses the parallel and the exclusive gates (Figure E.2).

Exclusive gateways – also denoted as decisions – allow the sequence flow to take two or more alternative paths (Figure E.2, 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 E.2, 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 E.3).

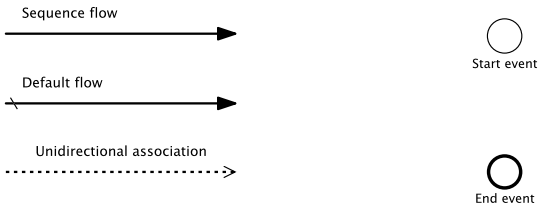

Figure E.3: *BPML connectors (left) and events (right).*

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 events 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, p. 30] show the full list of valid connections). SED-ML uses only a subset of available connectors, namely sequence flow, default flow, and unidirectional associations (Figure E.3). *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 E.4. The process of defining a SED-ML simulation

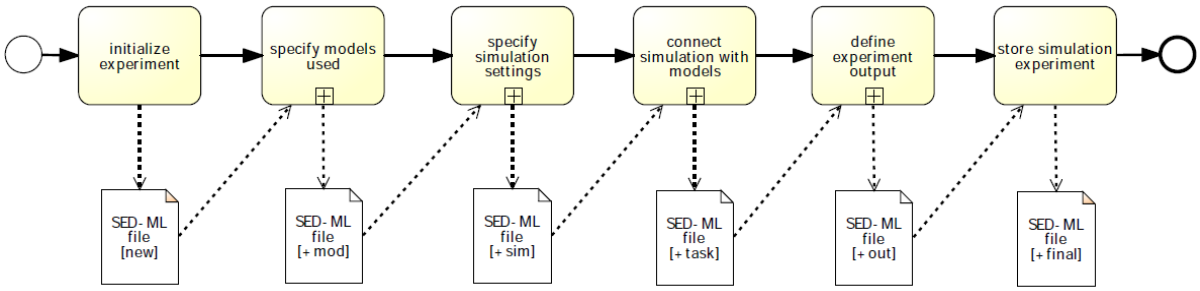

Figure E.4: *The process of defining a simulation experiment in SED-ML (overview)*

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 (Section E.2). In a third step, the simulation experiment setups are defined and stored into the same file (Section E.3). To assign a setup to a number of models used in the experiment, these connections have to be defined and recorded (Section E.4), called task in SED-ML. After simulation, the output should be defined, based on the specified tasks and performed simulation experiment. The information is added to the existing SED-ML file (Section E.5). In the end, the whole experiment is stored in the final SED-ML file. All collapsed processes are described in the following sections. Examples in XML are provided in the more technical description.

E.2 Models

To define a simulation experiment, first of all 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 database (e. g. Biomodels Database [Li et al., 2010] or CellML Repository [Beard et al., 2009]). All necessary changes to correctly simulate the model are defined, e. g., assigning new parameter values or updating the mathematics of the model (Figure E.5). The procedure is repeated until all models

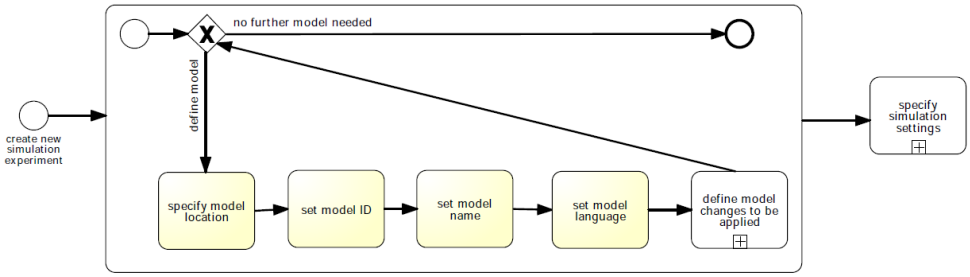

Figure E.5: *The process of defining model(s) in SED-ML*

participating in the experiment have been described. Each such model gets an internal SED-ML ID and an optional name.

E.3 Simulation setup

Secondly, the simulation setups (zero or many) used throughout the simulation experiment are described (Figure E.6). Those may stem from various different types of simulation, e. g., steady state analysis or

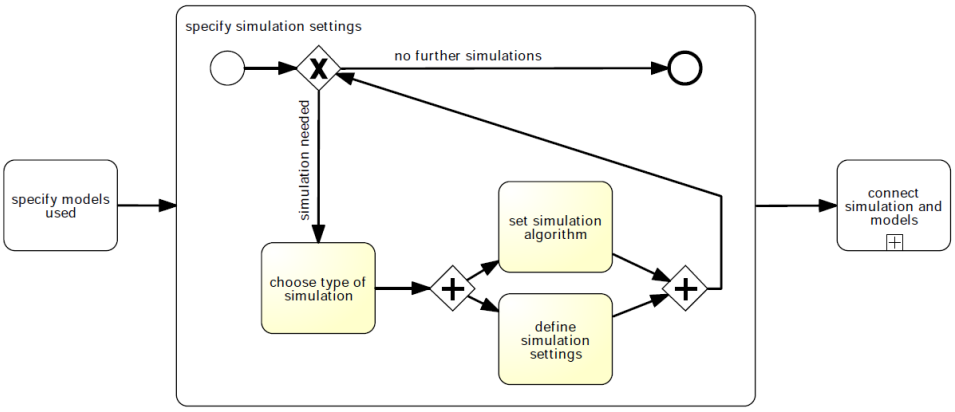

Figure E.6: *The process of defining simulation(s) in SED-ML*

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 complex. 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* (Section 2.2.4). The setup definition is repeated until all different simulations have been described.

E.4 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 E.7). To do so, each

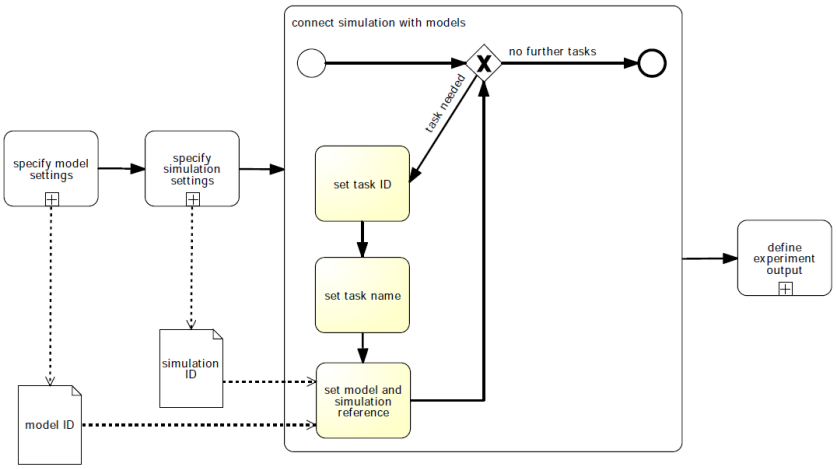

Figure E.7: *The process of defining simulation task(s) in SED-ML*

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.

E.5 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 E.8), and also which data is part of the output. Therefore, SED-ML first defines a set of data generators (Figure E.9), which are

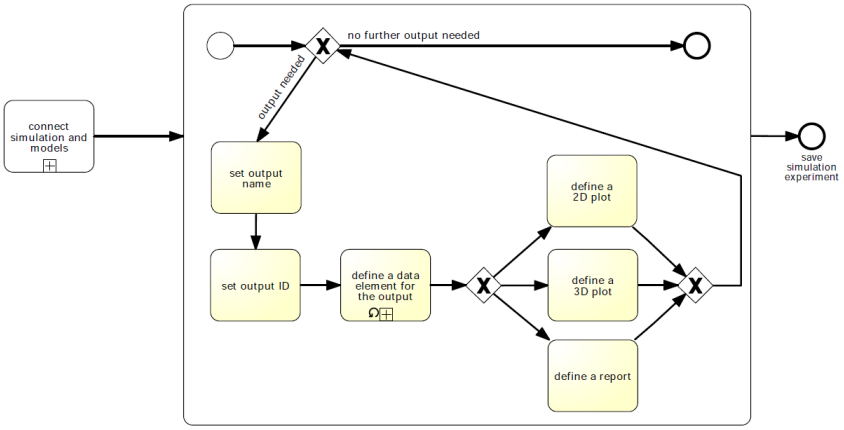

Figure E.8: *The process of defining output(s) in SED-ML*

then used to specify a particular result, i. e. output (Section E.6).

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.

E.6 Data Generator

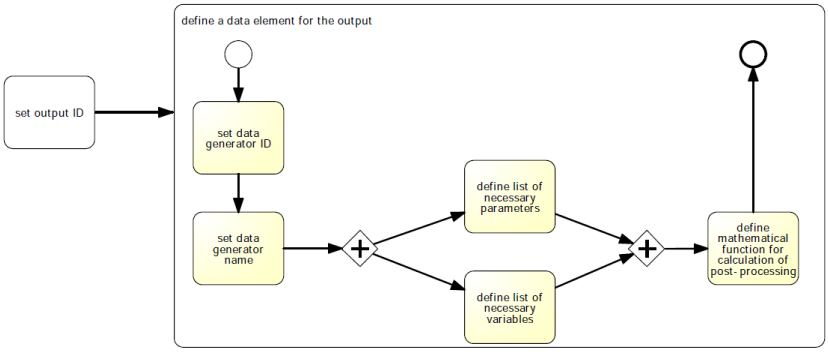

Figure E.9: *The process of defining data generator(s) in SED-ML*

A data generator may use as input data elements, e. g., variables or parameters, that have been taken directly from a model as simulated in a task. Within each data generator, either a single variable may be passed unchanged to the output definitions, or post-processing may be performed by applying mathematical functions to one or more variables or parameters. In a SED-ML file, any number of data generators can be created for later re-use in the output definition.

Bibliography

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.

D. 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, C.M. 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.

J. Clarke and S. DeRose. XML Path Language (XPath) Version 1.0, 1999. URL http://www.w3.org/TR/xpath/.

Melanie Courtot, N. Juty, Christian Knüpfer, D. Waltemath, A. Dräger, A. andFinney, M. Golebiewski, S. Hoops, S. Keating, D.B. Kell, S. Kerrien, J. Lawson, A. Lister, J. Lu, R. Machne, P. Mendes, M. Pocock, N. Rodriguez, A. Villeger, S. Wimalaratne, C. Laibe, M. Hucka, and N. Le Novère. Controlled vocabularies and semantics in systems biology. *Mol Sys Biol*, 7, October 2011. doi: 10.1038/msb.2011.77.

J.O. Dada, I. Spasić, N.W. Paton, and P. Mendes. SBRML: a markup language for associating systems biology data with models. *Bioinformatics (Oxford, England)*, 26(7):932–938, April 2010. ISSN 1367-4811. doi: 10.1093/bioinformatics/btq069.

M.B. 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.

S. Hoops, S. Sahle, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI - a COmplex PAthway SImulator. *Bioinformatics (Oxford, England)*, 22(24):3067–3074, December 2006. ISSN 1460-2059. doi: 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.

M. Hucka, F.T. Bergmann, S. Hoops, S. Keating, S. Sahle, and D.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.

N. Le Novère, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J. L. Snoep, and M. Hucka. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res*, 34(Database issue), January 2006. ISSN 1362-4962.

C. Li, M. Donizelli, N. Rodriguez, H. Dharuri, L. Endler, V. Chelliah, L. Li, E. He, A. Henry, M. Stefan, J. Snoep, M. Hucka, N. Le Novère, and C. 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.

C.M. Lloyd, M.D.B. Halstead, and P.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). *PLoS Comput Biol*, 7:e1001122, 2011. doi: 10.1371/journal.pcbi.1001122.

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