Simulation Experiment Description Markup Language (SED-ML) : Level 1 Version 3

July 19, 2017

Disclaimer: This is a working draft of the Simulation Experiment Description Markup Language (SED-ML) Level 1 Version 3 specification. It is not a normative document.

Editors

Matthias König	*Humboldt University Berlin, Germany*
David Nickerson	*Auckland Bioengineering Institute*
Brett Olivier	*University Amsterdam*
Lucian Smith	*University of Washington, US*
Dagmar Waltemath	*University of Rostock, Germany*

The latest release of the Level 1 Version 3 specification is available at
http://identifiers.org/combine.specifications/sed-ml.level-1.version-3

To discuss any aspect of SED-ML and the SED-ML specification write to the mailing list sed-ml-discuss@googlegroups.com.

To contact the SED-ML editors write to sed-ml-editors@googlegroups.com.

1. Introduction

The Simulation Experiment Description Markup Language (SED-ML) is an XML-based format for the description of simulation experiments.

The number of computational models of biological systems is growing at an ever increasing pace. At the same time, their size and complexity are also increasing. It is now generally accepted that one must be able to exchange the mathematical structure of such models, for instance to build on existing studies by reusing models or for the reproduction of model results. The efforts to standardise the representation of computational models in various areas of biology, such as the Systems Biology Markup Language (SBML) [15], CellML [8] or NeuroML [12], resulted in an increase of the exchange and re-use of models.

However, the description of the structure of models is not sufficient for the reproduction of simulation results. One also needs to describe the procedures the models are subjected to, i.e. the minimal set of information that should be provided to allow the reproduction of simulation experiments among users and software tools as described by the Minimum Information About a Simulation Experiment (MIASE [22]). The increasing use of computational simulation experiments to inform modern biological research creates new challenges to reproduce, annotate, archive, and share such experiments.

SED-ML describes in a computer-readable exchange format the information to enable the reproduction of simulation experiments. SED-ML is a software-independent format encoded in the Extensible Markup Language (XML) [4] not specific to particular simulation tools and independent of the underlying model implementation.

SED-ML is developed as a community project and defined via a detailed technical specification and a corresponding XML Schema.

This document describes Level 1 Version 3 of SED-ML which is the successor of Level 1 Version 2 and Level 1 Version 1 (described in [23]).

1.1 SED-ML overview

A SED-ML document specifies for a given simulation experiment

- what datasets to use
- which models to use in an simulation experiment
- which modifications to apply to models before simulation
- which simulation procedures to run on each model
- what analysis results to plot or report
- and how these results should be presented

SED-ML containts the following main objects to describe this information: `DataDescription` (2.4.1), `Model` (2.4.4), `Simulation` (2.4.6), `Task` (2.4.8), `DataGenerator` (2.4.12), and `Output` (2.4.13).

DataDescription

The `DataDescription` (2.4.1) allows to specify data sets used in a simulation experiment. Such data can be used for instance for parametrization of model simulations or to plot data with simulation results.

Model

The Model (2.4.4) is used to reference the models used in the simulation experiment. SED-ML itself is independent of the model encoding underlying the models. The only requirement is that the model is referenced by an unambiguous identifier, for example using a MIRIAM URI. To specify the language in which the model is encoded, a set of predefined language URNs is provided.

The SED-ML Change (2.4.5) allows the application of changes to the referenced models, including changes on the XML attributes, e.g. changing the value of an observable, computing the change of a value using mathematics, or general changes on any XML element of the model representation that is addressable by XPath expressions, e.g. substituting a piece of XML by an updated one.

Simulation

The Simulation (2.4.6) defines the simulation settings and the steps taken during simulation. These include the particular type of simulation and the algorithm used for the execution of the simulation.

Task

SED-ML makes uses the Task (2.4.8) to combine a defined model (from Model) and a defined simulation setting (from Simulation).

DataGenerator

The DataGenerator class (2.4.12) allows to encode post-processing of simulation results before output, e.g. one might want to normalise a plot before output, or apply post-processing like mean-value calculation. In the definition of a DataGenerator, any addressable variable or parameter of any defined model may be referenced, and new entities might be specified using MathML.

Output

The Output (2.4.13) defines the output of the simulation, which can be either a two dimensional plot Plot2D (2.4.13.1), a three dimensional plot Plot3D (2.4.13.2), or data table Report (2.4.13.3). The Output is based on the DataGenerators.

This section provided a high level overview over the content of a SED-ML file. For the detailed technical specification see Chapter 2.

1.2 Example simulation experiment

In this section an introductory example is given how simulation experiments can be described with SED-ML. The example experiment uses the *repressilator* [10] a famous model capable of displaying rich and variable behaviors.

The *repressilator* is a synthetic oscillating network of transcription regulators in Escherichia coli. 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 can be obtained by numerical integration.

TODO MK: add sample experiment to examples in appendix TODO MK: create simulation results with tellurium and SED-ML webtools to show reproducibility

1.2.1 Time-course simulation

The first simulation experiment we run with the model reproduces the oscillation behavior of the model shown in Figure 1c of the reference publication [10]. This simulation experiment can be described as:

1. Import the model identified by the Unified Resource Identifier (URI) [3] urn:miriam:biomodels.db:BIOMD0000000012.

2. Select a deterministic simulation 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 tools supporting SED-ML results in the output depicted in Figure 1.1. TODO: legend and xaxis label missing, rerun with tellurium and SED-ML webtools.

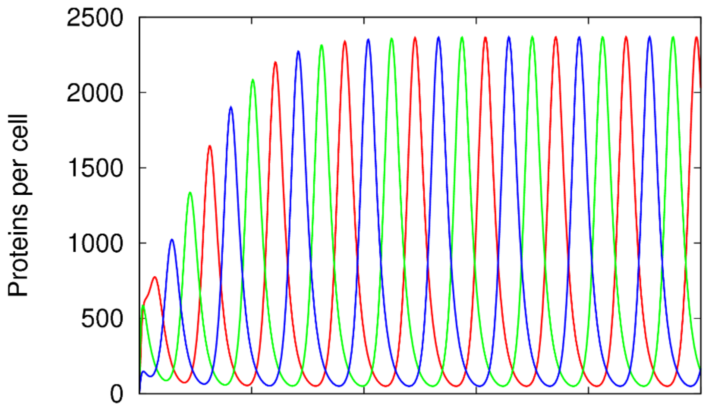

Figure 1.1: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI. The number of repressor proteins lacI, tetR and cI is depicted.*

1.2.2 Applying pre-processing

Before simulation model parameters can be adjusted. When changing the parameter values *protein copies per promoter* `tps_repr` and *leakiness in protein copies per promoter* `tps_active` like depicted below, the system's behavior switches from sustained oscillations to damped oscillations towards an asymptotic steady-state. The model changes leading to that behavior are described as:

1. Import the model as above.

2. Change the value of the parameter `tps_repr` from "0.0005" to "1.3e-05".

3. Change the value of the parameter `tps_active` from "0.5 " to "0.013".

4. Select a deterministic method.

5. Run a uniform time course for the duration of 1000 min with an output interval of 1 min.

6. Plot the amount of lacI, tetR and cI against time in a 2D Plot.

Figure 1.2 on the next page shows the result of the simulation.

1.2.3 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 1.2.1), depicting the influence of one variable on another (in phase-planes), one performes the additional steps:

(Please note that the description steps 1 - 4 remain as given in section 1.2.1 above.)

5. Collect lacI(t) , tetR(t) and cI(t).

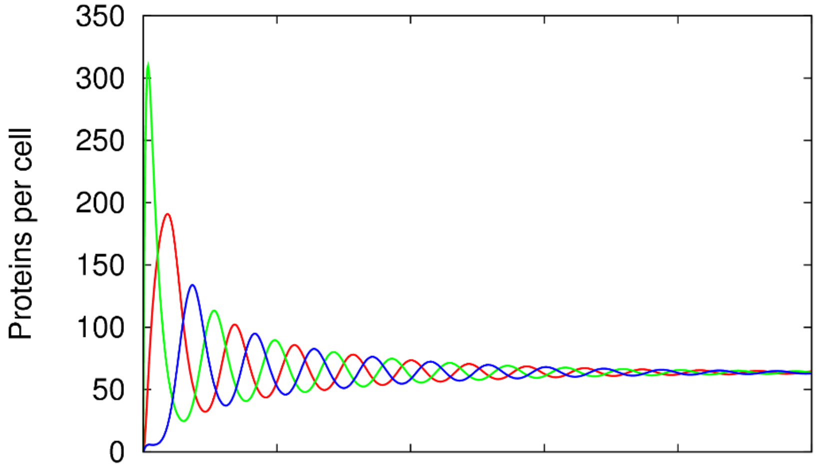

Figure 1.2: *Time-course simulation of the repressilator model, imported from BioModels Database and simulated after modification of the initial parameter values of the* protein copies per promoter *and the* leakiness in protein copies per promoter. *The number of repressor proteins lacI, tetR and cI are depicted.*

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

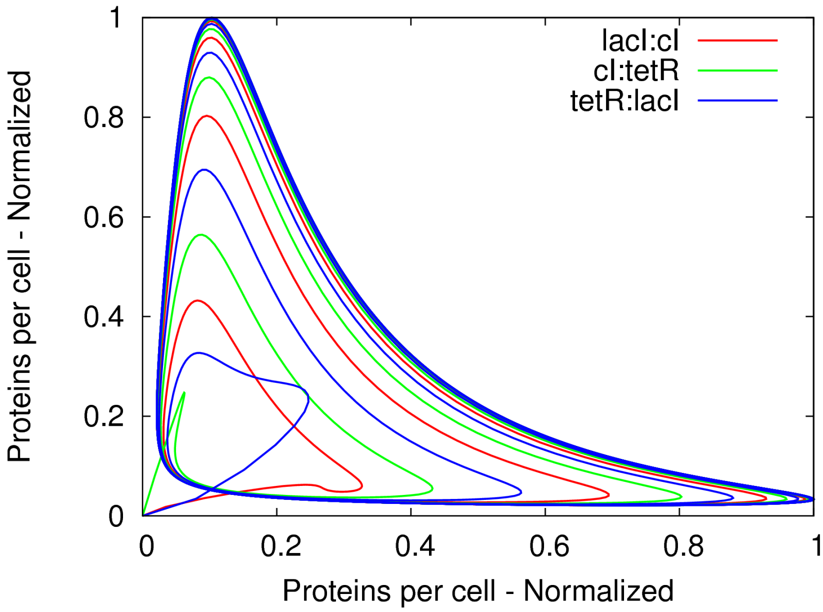

Figure 1.3: *Time-course simulation of the repressilator model imported from BioModels Database and simulated with COPASI. Depicted is the normalized temporal evolution of lacI, tetR and cI in phase-plane.*

2. SED-ML technical specification

This document represents the technical specification of SED-ML Level 1 Version 3. The corresponding UML class diagram is given in Appendix A, the XML Schema in Appendix B, example simulation experiments in Appendix C.

Sample experiment descriptions are given as XML snippets. However, not all concepts of SED-ML can be captured using XML Schema alone. In such cases this specification is 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`. 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`.

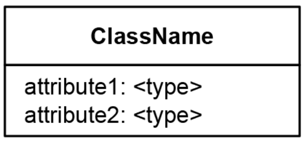

Figure 2.1: *UML class*

2.1.2 UML relationships

2.1.2.1 *UML relation types*

Links between classes specify the connection of objects with each other (Figure 2.2 on the following page). 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 2.2 on the next page) indicates the existence of a connection between the objects of the participating classes. Often associations are directed to show how the label should be read (in which direction). Associations can be uni-directional (one arrowhead), or bidirectional (zero or two arrowheads).

The aggregation (Figure 2.2 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`).

The composite aggregation (Figure 2.2 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`).

The generalisation (Figure 2.3 on the following page) allows to extend classes (`BaseClass`) by additional properties. The derived class (`DerivedClass`) inherits all properties of the base class.

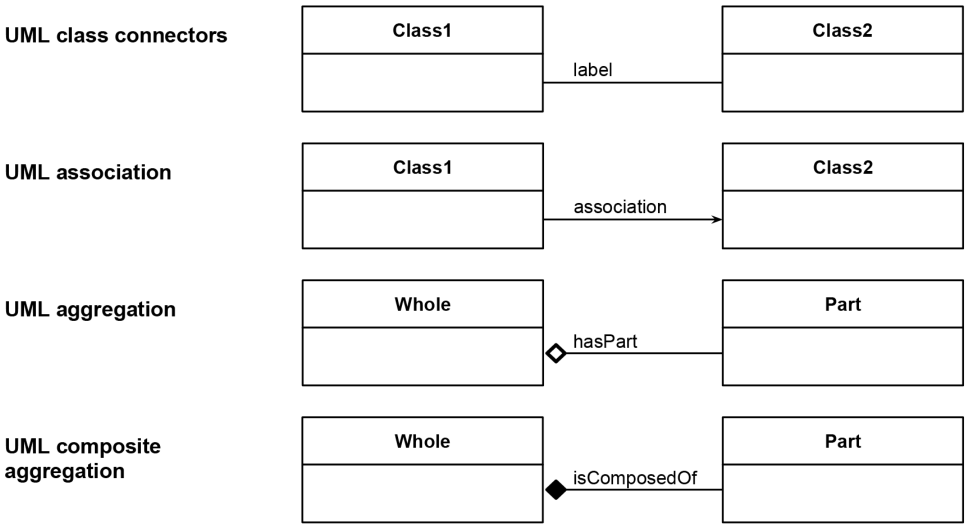

Figure 2.2: *UML class connectors, association, aggregation, composite aggregation*

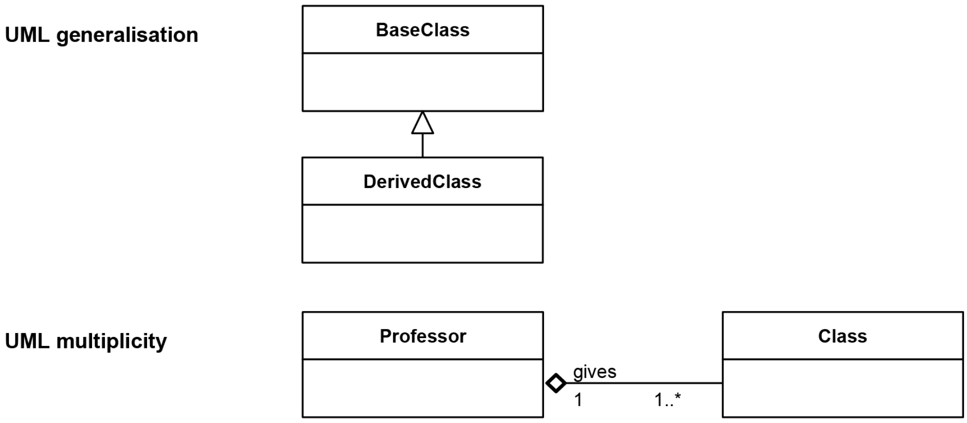

Figure 2.3: *UML generalisation (top) and multiplicity (bottom)*

2.1.2.2 UML multiplicity

UML multiplicity (Figure 2.3 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 asterisk (*).

Multiplicity can be defined for both sides of a relationship between classes. The default relationship is "many to many". The example in Figure 2.3 (bottom) expresses that a class is given by a professor, and a professor might give one to many classes.

2.1.3 XML schema language elements

The main building blocks of an XML Schema specification are simple and complex types, element specifications and 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, predefined 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 the element `computeChange` that is used in SED-ML to change mathematical expressions. The element is defined using an *unnamed* complex type which is built of further elements called `listOfVariables`, `listOfParameters`, and `math`. Additionally, the element `computeChange` has an attribute `target`. The definition of the elements inside the complex type are defined elsewhere in the schema and only referred to in the definition of `ComputeChange`.

The nesting of elements in the schema can be expressed using the concepts `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). 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 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="2" />
            </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 two new attributes `version` and `level`.

2.2 Concepts used in SED-ML

2.2.1 MathML subset

SED-ML files may encode pre-processing steps applied to the computational model, as well as post processing applied to the raw simulation data before output. The corresponding mathematical expressions are encoded using MathML 2.0 [5]. 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 model languages supported by SED-ML.

2.2.1.1 MathML elements

The allowed MathML in SED-ML is restricted to the following subset:

- *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, this basic set needs to be extended by some aggregate functions. Therefore a defined set of MathML symbols that represent vector values are supported by SED-ML Level 1 Version 3. The only allowed symbols to be used in aggregate functions are the identifiers of variables defined in the listOfVariables of DataGenerators. These variables represent the data collected from the simulation experiment in the associated task.

2.2.1.3 MathML functions

The only aggregate functions available in SED-ML Level 1 Version 3 are:

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

```xml
<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).

```xml
<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 3 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 used at various points in SED-ML Level 1 Version 3. They are used

- as a mechanism to reference models (2.2.2.1)
- as a mechanism to reference data files (2.2.2.2)
- to specify the language of the referenced model (2.2.2.3)
- to enable addressing implicit model variables (2.2.2.4)

Finally, annotations (2.2.2.5) 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

One 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 [17]. 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`.

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

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 – COMBINE archive format – is described in Chapter 3. Further information on the source attribute referencing the model location is provided in Section 2.4.4.2.

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

2.2.2.2 Data references

One way for referencing a data file from a SED-ML file is adopted from the MIRIAM URI Scheme. MIRIAM enables identification of a data resource by a predefined URN.

An alternative means to obtain a data file may be to provide a single resource containing necessary data files and the SED-ML file (see COMBINE archive format – is described in Chapter 3). Further information on the source attribute referencing the data file location is provided in Section 2.4.1.1.

2.2.2.3 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 3, Version 1, Revision 1".

The list of URNs is available from `http://sed-ml.org/`. Further information on the language attribute is provided in Section 2.4.4.1.

2.2.2.4 Implicit variables (symbols)

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 existing concepts in the languages supported by SED-ML is provided.

2.2.2.5 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

XPath is a language for finding information in an XML document [6]. Within SED-ML Level 1 Version 3, XPath version 1 expressions are used to identify nodes and attributes within an XML representation of a biological model in the following ways:

- Within a Variable definition, where XPath identifies the model variable required for manipulation in SED-ML.
- 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 NuML

TODO: explain what is NuML and how used to reference external datasets.

2.2.5 KiSAO

The Kinetic Simulation Algorithm Ontology (KiSAO [7]) is used in SED-ML to specify simulation algorithms and algorithm parameters. KiSAO is a community-driven approach of classifying and structuring simulation approaches by model characteristics and numerical characteristics. The ontology is available in OWL format from BioPortal at http://purl.bioontology.org/ontology/KiSAO.

Defining simulation algorithms through KISAO terms not only identifies the simulation algorithm used for the SED-ML simulation, it also enables software to find related algorithms, if the specific implementation is not available. For example, software could decide to use the CVODE integration library for an analysis instead of a specific Runge Kutta 4,5 implementation.

Should a particular simulation algorithm or algorithm parameter not exist in KISAO, please request one via http://www.biomodels.net/kisao/.

2.2.6 SED-ML resources

Information on SED-ML can be found on http://sed-ml.org. The SED-ML XML Schema, the UML schema, SED-ML examples and additional information is available from the SED-ML github project at https://github.com/sed-ml.

2.3 General attributes and classes

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

2.3.1 Primitive data types

Primitive data types comprise the set of data types used in SED-ML classes. Most primitive types in SED-ML are taken from the data types defined in XML Schema 1.0, including `string`, `boolean`, `int`, `positiveInteger`, `double` and `XML`. A few other primitive types are defined by SED-ML itself:

2.3.1.1 Type ID

The XML Schema 1.0 type ID is identical to the XML 1.0 type ID. The literal representation of this type consists of strings of characters restricted as summarized in Figure 2.4. For a detailed description see the SBML specification on the type `ID` [14].

```
NameChar   ::=   letter | digit | '.'  | '-' | ' ' | ':'  | CombiningChar | Extender
ID         ::=   ( letter | ' ' | ':'  )  NameChar*
```

Figure 2.4: *The definition of the type ID. The characters (and) are used for grouping, the character * indicates "zero or more times", and the character | indicates "or". Please consult the XML 1.0 specification for the complete definitions of letter, digit, CombiningChar, and Extender.*

2.3.1.2 Type SId

The type `SId` is the type of the id attribute found on the majority of SED-ML coponents. `SId` 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 2.5. For a detailed description see the SBML specification on the type `SId` [14].

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

Figure 2.5: *The definition of the type SId*

2.3.1.3 Type SIdRef

Type `SIdRef` is used for all attributes that refer to identifiers of type `SId` in a model. This type is derived from `SId`, but with the restriction that the value of an attribute having type `SIdRef` must equal the value of some `SId` attribute in the model where it appears. In other words, a `SIdRef` value must be an existing idetifier in a model.

As with `SId`, the equality of SIdRef values is determined by exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

2.3.1.4 Type XPath

Type `XPath` is used to identify nodes and attributes within an XML representation of a biological model. `XPath` is hereby a XPath version 1 expression which can be used to unambiguously identify an element or attribute in an XML file.

2.3.1.5 Type MathML

Type `MathML` is used to describe mathematical expression. For a description of the allowed subset of `MathML` see Section 2.2.1.

2.3.1.6 Type anyURI

Type **anyURI** is used to reference models, reference data files, specify the language of referenced models, for referencing implicit model variables and in annotations. For a description of the uses of **anyURI** see Section 2.2.2.

2.3.1.7 Type NuMLSIdRef

TODO: explain, probably also NuMLSId

2.3.2 id

Most objects in SED-ML carry an id attribute of data type SId. The id attribute, if it exists for an object, is required and identifies SED-ML constituents unambiguously. All **id**s have a global scope, i. e. the **id** must be unambiguous throughout a whole SED-ML document.

An example for a defined id is given in Listing 2.8. The defined model has the **id m00001**. If the model is referenced elsewhere in the SED-ML document, it is referred to by that **id**.

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

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

2.3.3 name

SED-ML classes may have an optional element **name** of data type **string**. Names do not have identifying character, i. e. several SED-ML constituents may have the same name. The purpose of the **name** attribute is to store a human-readable name of the constituent, e. g. for display to the user.

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

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

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

2.3.4 SEDBase

SEDBase is the base class of SED-ML Level 1 Version 3. All other classes are derived from it. As such it provides means to attach additional information on all other classes (Figure 2.6). That information can be specified by human readable Notes or custom Annotations.

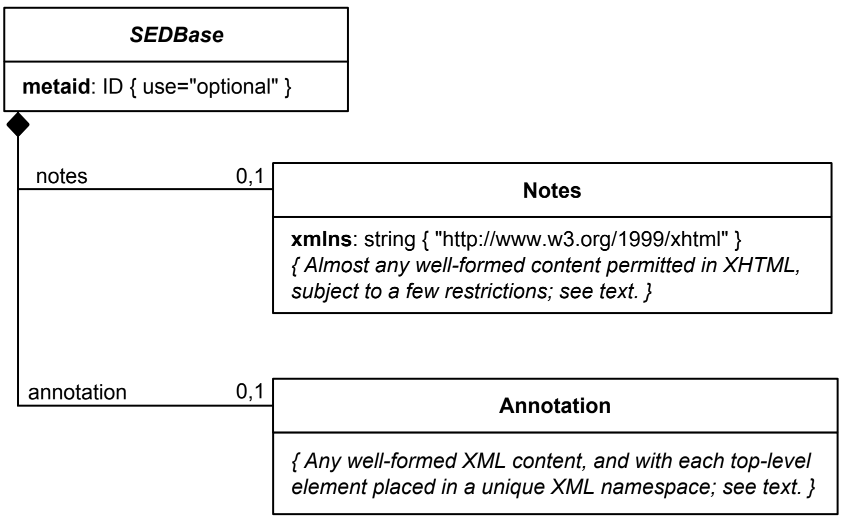

Figure 2.6: *The SEDBase class*

Table 2.1 shows all attributes and sub-elements for the SEDBase element.

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. ^{o}denotes optional elements and attributes.*

2.3.4.1 metaid

The main purpose of the **metaid** attribute of data type ID is to attach semantic annotations in form of the Annotation class to SED-ML elements. The **metaid** attribute is globally unique throughout the SED-ML document, i. e. the **metaid** must be unambiguous throughout a whole SED-ML document. As such it identifies the constituent it is related to.

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

2.3.4.2 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 [20], 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 [14, p. 14].

Table 2.2 shows all attributes and sub-elements for the Notes element.

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. ^{o}denotes optional elements and attributes.*

Notes does not have any further sub-elements defined in SED-ML, nor attributes associated with it.

Listing 2.10 shows the use of the **notes** element.

```
<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`.

2.3.4.3 *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 [14, pp. 14-16]. The style of annotations in SED-ML is briefly described in Section 2.2.2.5 on page 15.

Listing 2.11 shows the use of the `annotation` element.

```
<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 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.

2.3.5 SED-ML top level element

Each SED-ML Level 1 Version 3 document has a main class called SED-ML which defines the document’s structure and content (Figure 2.7 on the following page).

It consists of several parts; the parts are all connected to the SED-ML class through aggregation:

- the DataDescription class (for resolving external data, see Section 2.4.1),
- the Model class (for model specification, see Section 2.4.4),
- the Simulation class (for simulation setup specification, see Section 2.4.6),
- the AbstractTask class (for the linkage of models and simulation setups, see Section 2.4.7),
- the DataGenerator class (for the definition of post-processing, see Section 2.4.12),
- and the Output class (for the output specification, see Section 2.4.13).

All of them are shown in Figure 2.7 on the next page and will be explained in more detail in the relevant sections of this document.

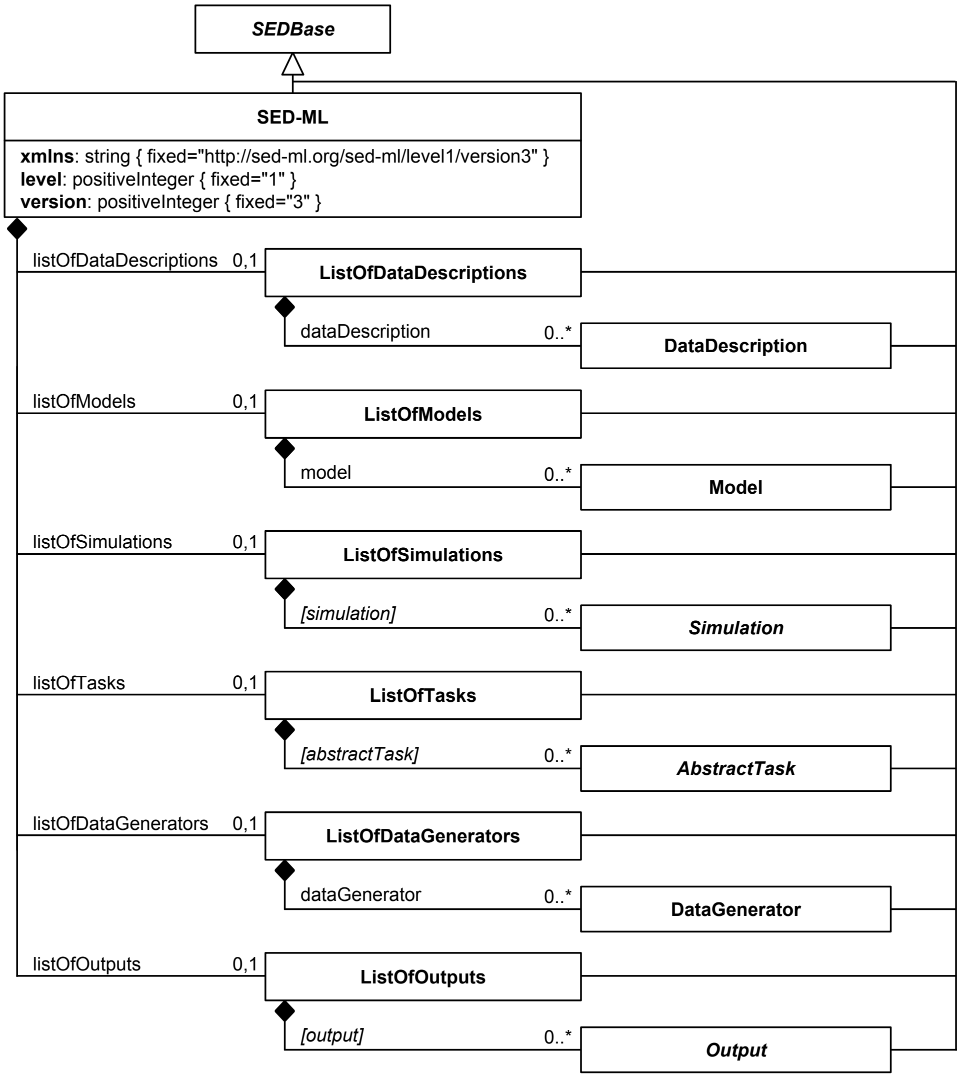

Figure 2.7: *The SED-ML class*

Table 2.3 shows all attributes and sub-elements for the SED-ML element.

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
listOfDataDescriptions^o	page 23
listOfModels^o	page 23
listOfSimulations^o	page 23
listOfTasks^o	page 24
listOfDataGenerators^o	page 24
listOfOutputs^o	page 24

Table 2.3: *Attributes and nested elements for SED-ML. ^{o}denotes optional elements and attributes.*

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/sed-ml/level1/version3" level="1" version="3">
    <listOfDataDescriptions>
        [DATA REFERENCES AND TRANSFORMATIONS]
    </listOfDataDescriptions>
    <listOfModels>
        [MODEL REFERENCES AND APPLIED CHANGES]
    </listOfModels>
    <listOfSimulations>
        [SIMULATION SETUPS]
    </listOfSimulations>
    <listOfTasks>
        [MODELS LINKED TO SIMULATIONS]
    </listOfTasks>
    <listOfDataGenerators>
        [DEFINITION OF POST-PROCESSING]
    </listOfDataGenerators>
    <listOfOutputs>
        [DEFINITION OF OUTPUT]
    </listOfOutputs>
</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 six lists serving as containers for the encoded data (listOfDataDescriptions for all external data sources, listOfModels for all models, listOfSimulations for all simulations, listOfTasks for all tasks, listOfDataGenerators for all post-processing definitions, and listOfOutputs for all output definitions).

2.3.5.1 `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/sed-ml/level1/version3`.

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 13.

SED-ML notes use the XHTML namespace `http://www.w3.org/1999/xhtml`. The Notes class is described in Section 2.3.4.2 on page 19.

Additional external namespaces might be used in annotations.

2.3.5.2 `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 3 the value is set to `1`, as shown in the example in Listing 2.12.

2.3.5.3 `version`

The current SED-ML version is "version 3". 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 3 the value is set to `3`, as shown in the example in Listing 2.12.

2.3.6 `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.

2.3.6.1 `listOfDataDescriptions`

In order to reference data in a simulation experiment, the data files along with a description on how to access such files and what information to extract from it have to be defined. SED-ML uses the listOfDataDescriptions container for all necessary data (Figure 2.7 on page 21). The `listOfDataDescriptions` is optional and may contain zero to many data files.

Listing 2.13 shows the use of the `listOfDataDescriptions` element.

```
<listOfDataDescriptions>
    <dataDescription id="Data1" name="Oscli Time Course Data" source="http://svn.code.sf.net/p/libsedml/code/trunk/Samples/data/oscli.numl">
        <dimensionDescription>
            <compositeDescription indexType="double" id="time" name="time" xmlns="http://www.numl.org/numl/level1/version1">
                <compositeDescription indexType="string" id="SpeciesIds" name="SpeciesIds">
                <atomicDescription valueType="double" name="Concentrations" />
                </compositeDescription>
            </compositeDescription>
        </dimensionDescription>
        <listOfDataSources>
            <dataSource id="dataS1">
                <listOfSlices>
                    <slice reference="SpeciesIds" value="S1" />
                </listOfSlices>
            </dataSource>
            <dataSource id="dataTime" indexSet="time" />
        </listOfDataSources>
    </dataDescription>
</listOfDataDescriptions>
```

Listing 2.13: *SED-ML listOfDataDescriptions element*

2.3.6.2 `listOfModels`

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 2.7 on page 21).

The `listOfModels` is optional and may contain zero to many models. However, if the Level 1 Version 3 document contains one or more `Task` elements, at least one `Model` element must be defined to which the `Task` element refers (c.f. Section 2.3.7.1 on page 26).

Listing 2.14 shows the use of the `listOfModels` element.

```
<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 2.14: *SED-ML listOfModels element*

2.3.6.3 `listOfChanges`

The listOfChanges contains the defined changes to be applied to a particular model (Figure 2.11 on page 34). It always occurs as an optional subelement of the `model` element. The `listOfChanges` is nested inside the `model` element. The listOfChanges is optional and may contain zero to many models.

Listing 2.15 shows the use of the `listOfChanges` element.

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

Listing 2.15: *The SED-ML `listOfChanges` element, defining a change on a model*

2.3.6.4 `listOfSimulations`

The listOfSimulations element is the container for simulation descriptions (Figure 2.7 on page 21). The `listOfSimulations` is optional and may contain zero to many simulations. However, if the Level 1

Version 3 document contains one or more `Task` elements, at least one `Simulation` element must be defined to which the `Task` element refers — see section 2.3.7.3.

Listing 2.16 shows the use of the `listOfSimulation` element.

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

Listing 2.16: *The SED-ML* `listOfSimulations` *element, containing two simulation setups*

2.3.6.5 `listOfAlgorithmParameters`

The listOfAlgorithmParameters contains the settings for the simulation algorithm used in a simulation (Figure 2.14 on page 42). It may list several instances of the AlgorithmParameter class. The `listOfAlgorithmParameters` is optional and may contain zero to many parameters.

Listing 2.17 shows the use of the `listOfAlgorithmParameters` element.

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

Listing 2.17: *SED-ML listOfAlgorithmParameters element*

2.3.6.6 `listOfTasks`

The listOfTasks element contains the defined tasks for the simulation experiment (Figure 2.7 on page 21).

Listing 2.18 shows the use of the `listOfTasks` element.

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

Listing 2.18: *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 3 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 2.3.7.2 on page 26.

2.3.6.7 `listOfDataGenerators`

In SED-ML, all variable and parameter values that are used in the Output class need to be defined as a dataGenerator beforehand. The container for those data generators is the listOfDataGenerators (Figure 2.7 on page 21).

The `listOfDataGenerators` is optional and in general may contain zero to many DataGenerators.

Listing 2.19 shows the use of the `listOfDataGenerators` element.

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

Listing 2.19: *The* `listOfDataGenerators` *element, defining two data generators* time *and* LaCI repressor

2.3.6.8 `listOfOutputs`

The listOfOutputs container holds the output definitions of a simulation experiment (Figure 2.7 on page 21). The listOfOutputs is optional and may contain zero to many outputs.

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

Listing 2.20 shows the use of the `listOfOutputs` element.

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

Listing 2.20: *The listOfOutput element*

2.3.6.9 listOfVariables

SED-ML uses the variable concept to refer to existing entities inside a model. The container for all variables is listOfVariables. It includes all variables that need to be defined to either describe a change in the model by means of mathematical equations via ComputeChange (Figure 2.13 on page 40 or to set up a DataGenerator (Figure 2.19 on page 54). The `listOfVariables` is optional and may contain zero to many variables.

Listing 2.21 shows the use of the `listOfVariables` element.

```
<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 2.21: *SED-ML listOfVariables element*

2.3.6.10 listOfParameters

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. See Figure 2.13 on page 40 or Figure 2.19 on page 54.

Listing 2.22 shows the use of the `listOfParameters` element. 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 2.22: *SED-ML `listOfParameters` element*

2.3.7 Reference relations

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

- as an association between two Models (modelReference),
- as an association between a Variable and a Model (modelReference),
- as an association between a Variable and an AbstractTask (taskReference),
- as an association between a Task and the simulated Model (modelReference),
- as an association between a Task and the Simulation run (simulationReference), or
- as an association between an Output and a DataGenerator (dataReference).

The definition of a Task object requires a reference to a particular Model object (modelReference, see Section 2.3.7.1 on page 26); furthermore, the Task object must be associated with a particular Simulation object (simulationReference, see Section 2.3.7.3 on page 27).

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 2.3.7.1 on page 26).

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 2.3.7.2 on page 26).

Four different types of data references exist in SED-ML Level 1 Version 3. 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.

2.3.7.1 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 2.23 shows the use of the `modelReference` element.

```
<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 2.23: *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 2.24 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 2.24: *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`.

2.3.7.2 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 2.25 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 2.25: *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 2.26.

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

Listing 2.26: *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 2.25 targets an element of the model with ID `model1`. The targeting process itself will be explained in section 2.3.8.1 on page 29.

2.3.7.3 simulationReference

The simulationReference is used to refer to a particular Simulation in a Task. Listing 2.24 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.

2.3.7.4 dataReference

The dataReference is used to refer to a particular DataGenerator instance from an Output instance. Listing 2.27 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 2.27: *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.

2.3.8 Variable

Variables are references to already existing entities, either existing in one of the defined models or implicitly defined symbols (Figure 2.8).

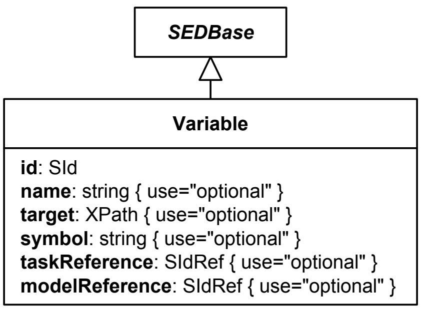

Figure 2.8: *The Variable class*

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 2.4 shows all attributes and sub-elements for the Variable element.

attribute	**description**
metaid^o	page 19
id	page 18
name^o	page 18
target	page 29
symbol	page 29
taskReference	page 26
modelReference	page 26
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.4: *Attributes and nested elements for Variable. ^{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 2.28 shows the use of the `variable` element.

```
1  <sedML>
2      <listOfModels>
3          <model [..]>
4              <listOfChanges>
5                  <computeChange target="TARGET ELEMENT OR ATTRIBUTE">
6                  <listOfVariables>
7                     <variable id="v1" name="maximum velocity" target="XPath TO MODEL ELEMENT/ATTRIBUTE" />
8                     [FURTHER VARIABLE DEFINITIONS]
9                  </listOfVariables>
10                 [..]
11                 </computeChange>
12             </listOfChanges>
13             [..]
14         </model>
15         [..]
16     </listOfModels>
17     <listOfDataGenerators>
18         <dataGenerator [..]>
19             <listOfVariables>
20                 <variable id="v2" name="time" taskReference="task1" symbol="urn:sedml:symbol:time" />
21                 [FURTHER VARIABLE DEFINITIONS]
22             </listOfVariables>
23         </dataGenerator>
24     </listOfDataGenerators>
25     [..]
26 </sedML>
```

Listing 2.28: *SED-ML `variable` definitions inside the `computeChange` element and inside the `dataGenerator` element*

Listing 2.28 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).

2.3.8.1 `target`

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

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 3 this is only used to refer to ranges within a repeatedTask (see Listing 2.51 for an example).

Listing 2.29 shows the use of the `target` attribute in a SED-ML file.

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

Listing 2.29: *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 2.29, 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 2.30 is another valid example for the specification of a variable, but uses the same naming in the variable definition as in the original model (as opposed to Listing 2.29):

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

Listing 2.30: *SED-ML variable definition using the original model identifier and name in SED-ML*

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

Listing 2.31: *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 2.31). 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.

2.3.8.2 `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 2.2.2.4 on page 15.

Listing 2.32 shows the use of the `symbol` attribute in a SED-ML file. 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 2.4.5.6.

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

Listing 2.32: *SED-ML* `symbol` *definition*

2.3.9 `Parameter`

The SED-ML Parameter class creates instances with a constant value (Figure 2.9 on the next page).

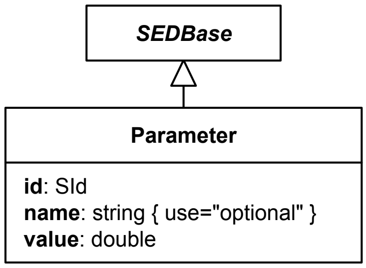

Figure 2.9: *The Parameter class*

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 2.5 shows all attributes and sub-elements for the parameter element.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
value	page 30
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.5: *Attributes and nested elements for parameter. ^{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 2.33 shows the use of the `parameter` element. The listing shows the definition of a parameter `p1` with the `value="40"` assigned.

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

Listing 2.33: *The definition of a parameter in SED-ML*

2.3.9.1 value

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

2.4 SED-ML Components

In this section the major components of SED-ML are described. We use the UML notation presented in section 2.1. The use of SED-ML is shown based on XML listings. The corresponding UML class diagram is given in Appendix A, the XML Schema in Appendix B, example simulation experiments in Appendix C.

2.4.1 DataDescription

The DataDescription class (Figure 2.10 on the following page) references a file containing data points, along with a description on how to access that file, and what information to extract from it.

The DataDescription class introduces three attributes: the required attributes `id` and `source` and the optional attribute `name`. Additionally two elements are defined: `dimensionDescription` and `listOfDataSources`.

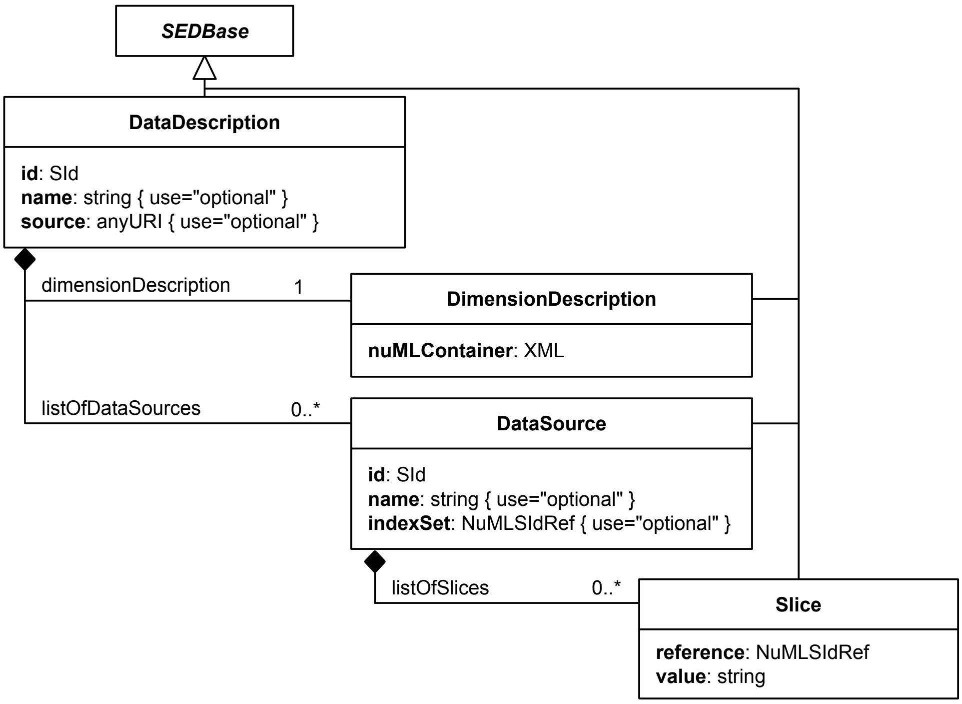

Figure 2.10: *The SED-ML DataDescription class*

Table 2.6 shows all attributes and sub-elements for the dataDescription element.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
source	page 31
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
dimensionDescription^o	page 32
listOfDataSources^o	page 32

Table 2.6: *Attributes and nested elements for dataDescription. ^{o}denotes optional elements and attributes.*

2.4.1.1 source

Analog to how the `source` attribute on the Model is handled, this attribute provides a location of a data file. In order to resolve the `source` attribute, the same mechanisms are allowed as for Model element: be it a local file system, a relative link or an online resource. In the Level 1 Version 3 only source files encoded in either NuML or CSV are allowed, with NuML being the recommended data format.

In case of CSV as source encoding the file

- must contain a header row which defines the ids

- must only contain numerical values
- must use the comma "," as field separator
- must use the dot "." as separator in numbers
- may contain comment rows which start with "#"
- the dimensionDescription of the CSV dataDescription must be two dimensional and correspond to the content of the CSV file

Listing 2.34 shows the use of the `dataDescription` element.

```
<dataDescription id="Data1" name="Oscli Time Course Data"
    source="http://svn.code.sf.net/p/libsedml/code/trunk/Samples/data/oscli.numl" >
    [...]
</dataDescription>
```

Listing 2.34: *SED-ML* `dataDescription` *element*

2.4.1.2 dimensionDescription

The dimensionDescription element is the data description from an NuML file. Consider for example:

```
<dimensionDescription>
    <compositeDescription indexType="double" id="time" name="time"
        xmlns="http://www.numl.org/numl/level1/version1">
        <compositeDescription indexType="string" id="SpeciesIds" name="SpeciesIds">
            <atomicDescription valueType="double" name="Concentration" />
        </compositeDescription>
    </compositeDescription>
</dimensionDescription>
```

Listing 2.35: *SED-ML* `dimensionDescription` *element*

Here a nested NuML `compositeDescription` with `time` spanning one dimension and `SpeciesIds` another dimension. This two dimensional space is then filled with `double` values representing concentrations.

2.4.1.3 listOfDataSources

The listOfDataSources contains one or more DataSource elements that are then used in the remainder of the SED-ML document.

2.4.2 DataSource

The DataSource class (Figure 2.10 on the previous page) extracts chunks out of the data file provided by the outer DataDescription element.

The DataSource class introduces three attributes: the required attribute `id` and the optional attributes `name` and `indexSet`. Additionally the `listOfSlices` element is defined (Figure 2.10 on the preceding page).

Table 2.7 shows all attributes and sub-elements for the dataSource element.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
indexSet	page 33
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfSlices^o	page 33

Table 2.7: *Attributes and nested elements for dataSource.* ^{o}*denotes optional elements and attributes.*

Once the DataSource elements are defined, they can be reused anywhere in the SED-ML Description. Specifically their **id** attribute can be referenced within the **listOfVariables** of **DataGenerators**, **computeChange** or **setValue** objects. Here an example that re-uses the data source **dataS1**:

```
<dataGenerator id="dgDataS1" name="S1 (data)">
    <listOfVariables>
        <variable id="varS1" modelReference="model1" target="#dataS1" />
    </listOfVariables>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> varS1 </ci>
    </math>
</dataGenerator>
```

This represents a change from Level 1 Version 1 and Level 1 Version 2, in which a **taskReference** was always present for a **variable** in a data generator.

To indicate that the target is an entity defined within the current SED-ML description the hashtag (#) with the reference to an **id** was used. Additionally, this example uses the **modelReference**, in order to facilitate a mapping of the data encoded in the NuML document with a given model.

Since data elements in NuML can be either values or indices, the DataSource element provides two ways of addressing those elements. The **indexSet** attribute allows to address all indices provided by NuML elements with **indexType**.

2.4.2.1 indexSet

Since data elements in NuML can be either values or indices, the DataSource element provides two ways of addressing those elements. The **indexSet** attribute allows to address all indices provided by NuML elements with **indexType**.

For example for the **indexSet time** below, a dataSource would extract the set of all timepoints stored in the index.

```
<dataSource id="dataTime" indexSet="time" />
```

Similarly:

```
<dataSource id="allIds" indexSet="SpeciesIds" />
```

would extract all the species id strings stored in that index set. Valid values for **indexSet** are all NuML Id elements declared in the **dimensionDescription**.

If the **indexSet** attribute is specified the corresponding **dataSource** may not define any **slice** elements.

2.4.2.2 listOfSlices

The listOfSlices contains one or more Slice elements. The listOfSlices container holds the Slice definitions of a DataSource (Figure 2.10 on page 31). The listOfSlices is optional and may contain zero to many Slices.

2.4.3 Slice

If a DataSource does not define the **indexSet** attribute, it will contain Slice elements. Each slice removes one dimension from the data hypercube.

The Slice class introduces two required attributes: **reference** and **value** (Figure 2.10 on page 31).

Table 2.8 on the following page shows all attributes and sub-elements for the slice element.

2.4.3.1 reference

The **reference** attribute references one of the indices described in the **dimensionDescription**. In the example above, valid values would be: **time** and **SpeciesIds**.

2.4.3.2 value

The **value** attribute takes the value of a specific index in the referenced set of indices. For example:

```
<dataSource id="dataS1">
    <listOfSlices>
```

attribute	description
metaid^o	page 19
reference	page 33
value	page 33
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.8: *Attributes and nested elements for slice. ^{o}denotes optional elements and attributes.*

```
        <slice reference="SpeciesIds" value="S1" />
    </listOfSlices>
</dataSource>
```

would isolate the index set of all species ids specified, to only the single entry for `S1`, however over the full range of the `time` index set. As stated before, there could be multiple slice elements present, so it would be feasible to slice the data again, to obtain a single time point, for example the initial one:

```
<dataSource id="dataS1">
    <listOfSlices>
        <slice reference="time" value="0" />
        <slice reference="SpeciesIds" value="S1" />
    </listOfSlices>
</dataSource>
```

2.4.4 `Model`

The Model class defines the models used in a simulation experiment (Figure 2.11).

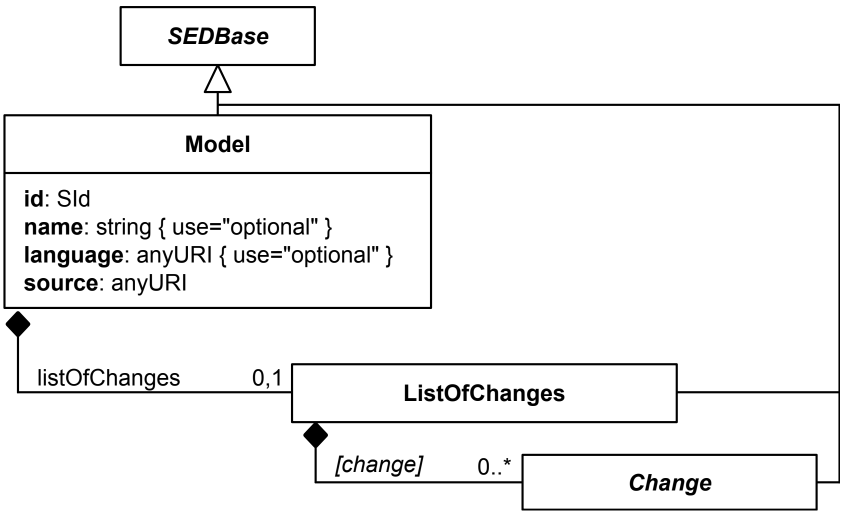

Figure 2.11: *The SED-ML Model class*

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-processing steps are specified in the SED-ML Change class.

Table 2.9 shows all attributes and sub-elements for the model element.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
language^o	page 35
source	page 36
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
change^o	page 36

Table 2.9: *Attributes and nested elements for model. ^{o}denotes optional elements and attributes.*

Listing 2.36 shows the use of the `model` element.

```
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 2.36: *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.

2.4.4.1 `language`

The evaluation of a SED-ML document is required in order for software to decide whether or not it can be used in 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 2.2.2.3 on page 15). 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 2.2.2.4 on page 15.

2.4.4.2 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 14).

An example for the definition of a model, and using the URI scheme is given in Listing 2.37.

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

Listing 2.37: *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 2.38.

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

Listing 2.38: *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.

2.4.5 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 2.12 on the following page). Changes can be of the following types:

- Changes on attributes of the model's XML representation (ChangeAttribute)

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

- Changes on any XML snippet of the model's XML representation (AddXML, ChangeXML, RemoveXML)
- 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.

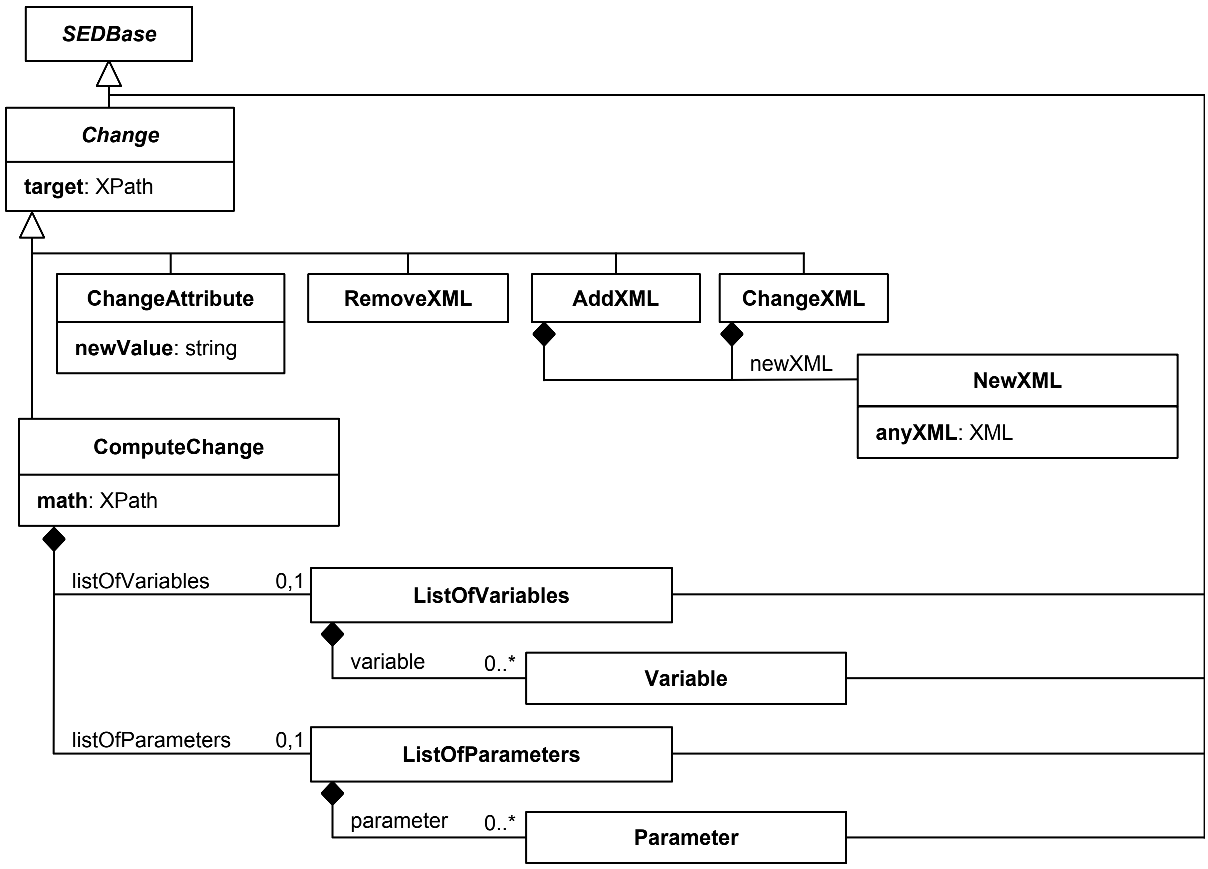

Figure 2.12: *The SED-ML Change class*

Table 2.10 shows all attributes and sub-elements for the change element.

attribute	**description**
metaid^o	page 19
id	page 18
name^o	page 18
target	page 29
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.10: *Attributes and nested elements for change. ^{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.

2.4.5.1 NewXML

The `newXML` element provides a piece of XML code (Figure 2.12 on the preceding page). `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 2.11 shows all attributes and sub-elements for the newXML element.

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

Table 2.11: *Attributes and nested elements for newXML. ^{o}denotes optional elements and attributes.*

The `newXML` element is used at two different places inside SED-ML Level 1 Version 3:

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.

2.4.5.2 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 2.12 on the preceding page). The new piece of XML code is provided by the NewXML class.

An example for a change that adds an additional parameter to a model is given in Listing 2.39. In the example 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.

```
<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 2.39: *The `addXML` element with its `newXML` sub-element*

2.4.5.3 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 2.12 on the previous page).

The XPath expression is specified in the required target attribute (Section 2.3.8.1 on page 29). The replacement XML content is specified in the NewXML class.

An example for a change that adds an additional parameter to a model is given in Listing 2.40. In the example 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.

```
<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 2.40: *The* `changeXML` *element*

2.4.5.4 RemoveXML

The RemoveXML class can be used to delete XML elements or attributes in the model that are addressed by the XPath expression (Figure 2.12 on page 37). The XPath is specified in the required target attribute.

An example for the removal of an XML element from a model is given in Listing 2.41. In the example the model is changed by deleting the reaction with ID `V_mT` from the model's list of reactions.

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

Listing 2.41: *The* `removeXML` *element*

2.4.5.5 ChangeAttribute

The ChangeAttribute class allows to define updates on the XML attribute values of the corresponding model (Figure 2.12 on page 37).

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 2.12 shows all attributes and sub-elements for the changeAttribute element.

attribute	description
metaid^{o}	page 19
id	page 18
name^{o}	page 18
target	page 29
newValue	page 39
sub-elements	**description**
notes^{o}	page 19
annotation^{o}	page 20

Table 2.12: *Attributes and nested elements for ChangeAttribute. ^{o}denotes optional elements and attributes.*

2.4.5.5.1 newValue

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

The example in Listing 2.42 shows the update of the initial concentration of two parameters inside an SBML model.

```xml
<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 2.42: *The* `changeAttribute` *element and its* `newValue` *attribute*

2.4.5.6 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 2.13).

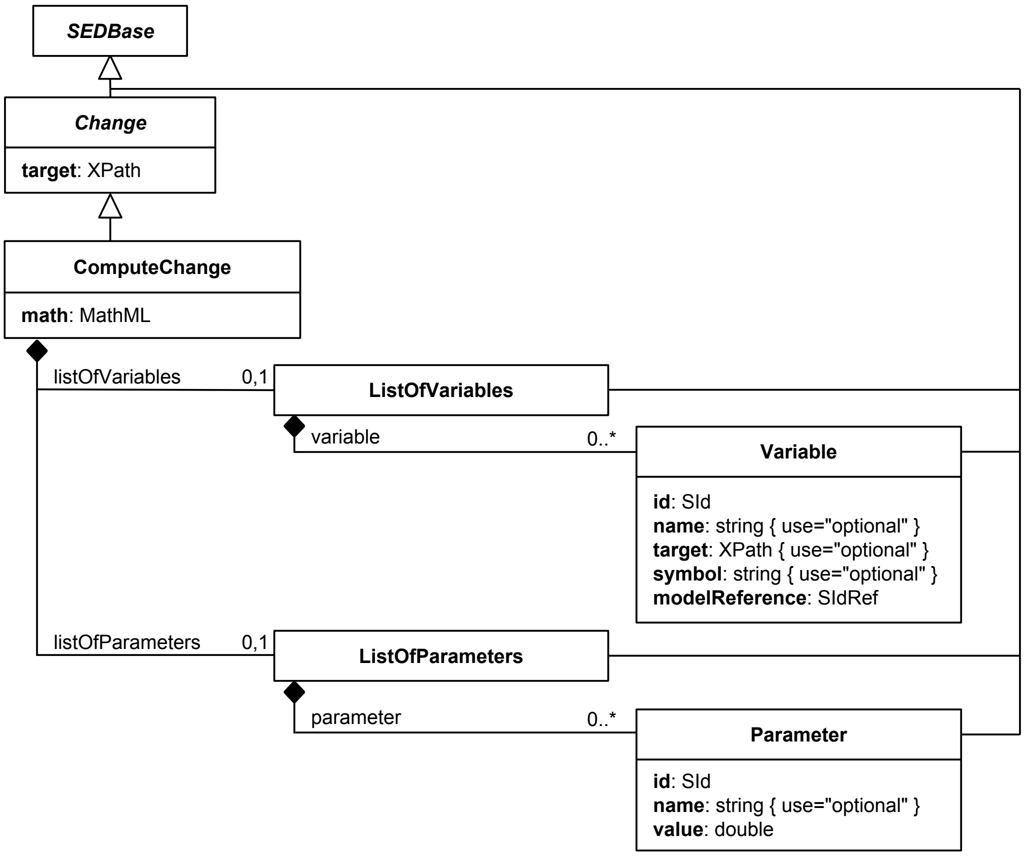

Figure 2.13: *The* `ComputeChange` *class*

The computed new value is described by a mathematical expression using a subset of MathML (see section 2.2.1 on page 13). 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 26) but no taskReference attribute (page 26).

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 2.13 shows all attributes and sub-elements for the computeChange element.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
target	page 29
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfVariables^o	page 25
listOfParameters^o	page 25
math	page 41

Table 2.13: *Attributes and nested elements for computeChange. ^{o}denotes optional elements and attributes.*

2.4.5.6.1 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 3 supports the subset of MathML 2.0 shown in section 2.2.1.

Listing 2.43 shows the use of the `computeChange` element.

```
<model [..]>
    <listOfChanges>
    <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 2.43: *The computeChange element*

The example in Listing 2.43 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:

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

2.4.6 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 2.14).

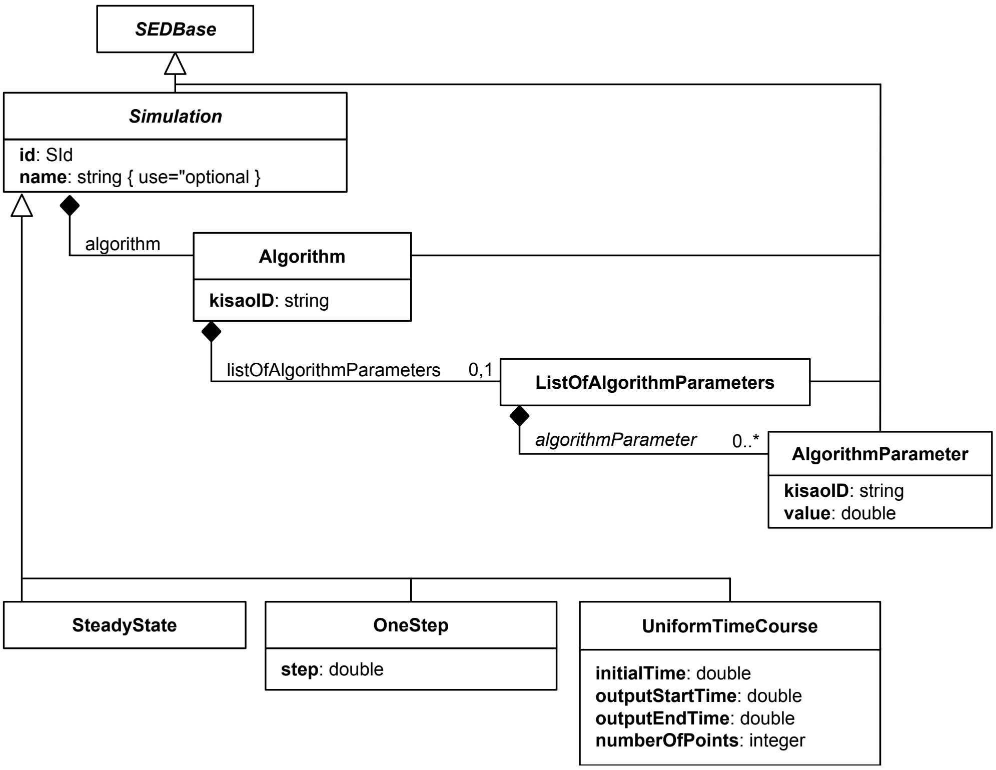

Figure 2.14: *The SED-ML Simulation class*

Simulation is an abstract class and serves as the container for the different types of simulation experiments. SED-ML Level 1 Version 3 provides the predefined simulation classes UniformTimeCourse, OneStep and SteadyState. Simulation algorithms used for the execution of a simulation are defined via the Algorithm class.

Each instance of the Simulation class has an unambiguous and mandatory `id`. An additional, optional `name` may be given to the simulation. Every simulation has a required element `algorithm`.

Table 2.14 on the next page shows all attributes and sub-elements for the simulation element.

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 2.14: *Attributes and nested elements for simulation. ^{o}denotes optional elements and attributes.*

Listing 2.44 shows the use of the `simulation` element.

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

Listing 2.44: *The SED-ML* `listOfSimulations` *element, defining two different UniformTimecourse simulations*

2.4.6.1 Algorithm

SED-ML makes use of the KiSAO ontology (Section 2.2.5 on page 16) 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 2.14 on the previous page).

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 `listOfAlgorithmParameters`, a collection of parameters (algorithmParameter) that are used to parameterize the particular algorithm used in the simulation.

Table 2.15 shows all attributes and sub-elements for the Algorithm element.

attribute	description
metaid^o	page 19
kisaoID	page 16
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfAlgorithmParameters^o	page 44

Table 2.15: *Attributes and nested elements for algorithm. ^{o}denotes optional elements and attributes.*

The example given in Listing 2.44, completed by algorithm definitions results in the code given in Listing 2.45.

```
<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 2.45: *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).

2.4.6.2 `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 (Figure 2.14 on page 42). 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 2.46: *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*.*

2.4.6.3 `UniformTimeCourse`

Each instance of the UniformTimeCourse class has, in addition to the elements from Simulation, the mandatory elements `initialTime`, `outputStartTime`, `outputEndTime`, and `numberOfPoints` (Figure 2.14 on page 42).

Table 2.16 shows all attributes and sub-elements for the uniformTimeCourse element.

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

Table 2.16: *Attributes and nested elements for uniformTimeCourse. ^{o}denotes optional elements and attributes.*

Listing 2.47 shows the use of the `uniformTimeCourse` element.

```
<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 2.47: *The SED-ML* **`uniformTimeCourse`** *element, defining a uniform time course simulation over 2500 time units with 1000 simulation points.*

2.4.6.3.1 *initialTime*

The attribute `initialTime` of type `double` represents the time from which to start the simulation. Usually this will be `0`. Listing 2.47 shows an example.

2.4.6.3.2 *outputStartTime*

Sometimes a researcher is not interested in simulation results at the start of the simulation (i.e. the initial time). The `uniformTimeCourse` class uses the attribute `outputStartTime` of type `double` to describe this simulation experiment. To be valid the `outputStartTime` cannot be before `initialTime`. For an example, see Listing 2.47.

2.4.6.3.3 *outputEndTime*

The attribute `outputEndTime` of type `double` marks the end time of the simulation. See Listing 2.47 for an example.

2.4.6.3.4 *numberOfPoints*

When executed, the `uniformTimeCourse` simulation produces an 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` 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 2.47.

2.4.6.4 *OneStep*

The SED-ML oneStep calculates one further output step for the model from its current state. Each instance of the OneStep class has, in addition to the elements from Simulation, the mandatory element `step` (Figure 2.14 on page 42).

Table 2.17 shows all attributes and sub-elements for the oneStep element.

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

Table 2.17: *Attributes and nested elements for oneStep. ^{o}denotes optional elements and attributes.*

Listing 2.48 shows the use of the `oneStep` element.

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

Listing 2.48: *The SED-ML* `oneStep` *element, specifying to apply the simulation algorithm for another output step of size 0.1.*

2.4.6.4.1 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 2.48 shows an example.

Note that the `step` 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.

2.4.6.5 SteadyState

The `SteadyState` class represents a steady state computation (as for example implemented by NLEQ or Kinsolve). The OneStep class has has no additional elements than the elements from Simulation (Figure 2.14 on page 42).

Table 2.18 shows all attributes and sub-elements for the steadyState element.

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 2.18: *Attributes and nested elements for steadyState. ^{o}denotes optional elements and attributes.*

Listing 2.49 shows the use of the `steadyState` element.

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

Listing 2.49: *The SED-ML `steadyState` element, defining a steady state simulation with id `steady`.*

2.4.7 Abstract Task

An abstractTask in SED-ML represents the base class for all SED-ML tasks, i.e. Task and RepeatedTask.

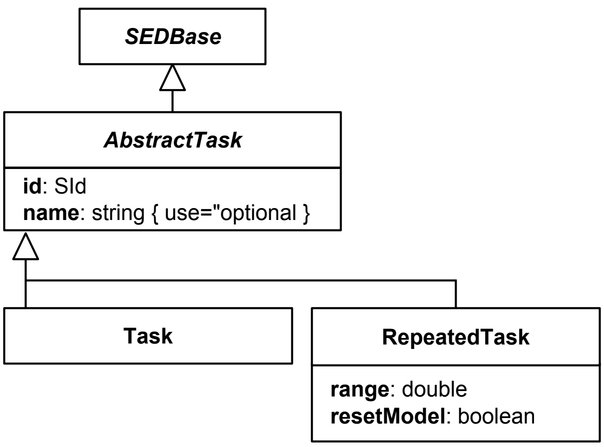

Figure 2.15: *The SED-ML Abstract Task class*

Table 2.19 shows all attributes and sub-elements for the abstractTask element.

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 2.19: *Attributes and nested elements for abstractTask. ^{o}denotes optional elements and attributes.*

2.4.8 Task

A task in SED-ML links a model to a certain simulation description via their respective identifiers (Figure 2.16), using the modelReference and the simulationReference. The task class receives the `id` and `name` attributes from the AbstractTask.

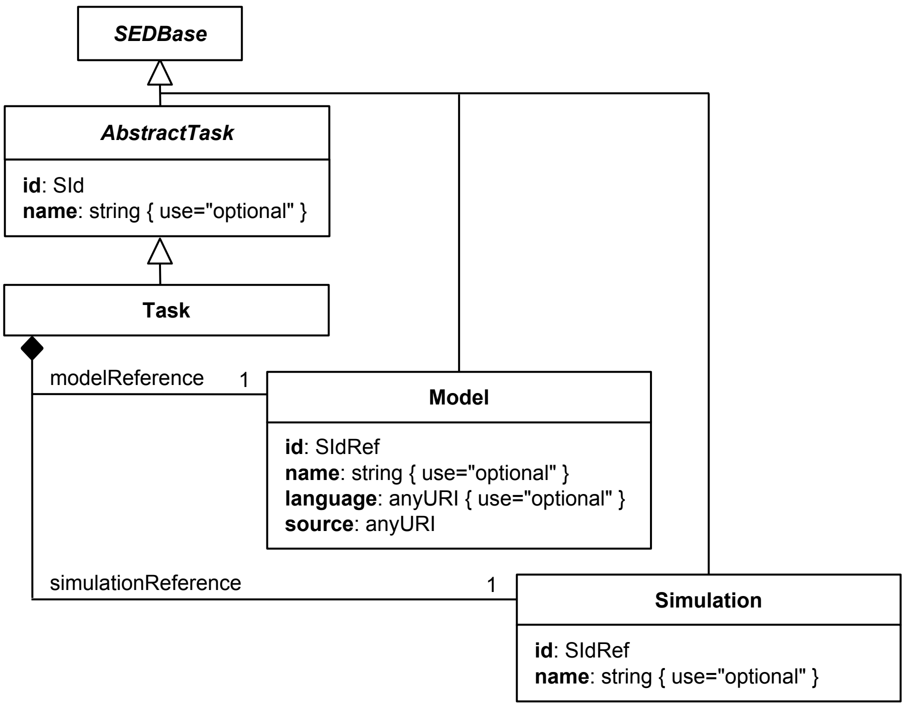

Figure 2.16: *The SED-ML Task class*

In SED-ML Level 1 Version 3 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 2.20 on the next page shows all attributes and sub-elements for the task element.

Listing 2.50 shows the use of the `task` element.

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

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
modelReference	page 26
simulationReference	page 27
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.20: *Attributes and nested elements for task. ^{o}denotes optional elements and attributes.*

```
</listOfTasks>
```

Listing 2.50: *The `task` element*

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

2.4.9 Repeated Task

The repeatedTask class (Figure 2.17 on the next page) 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).

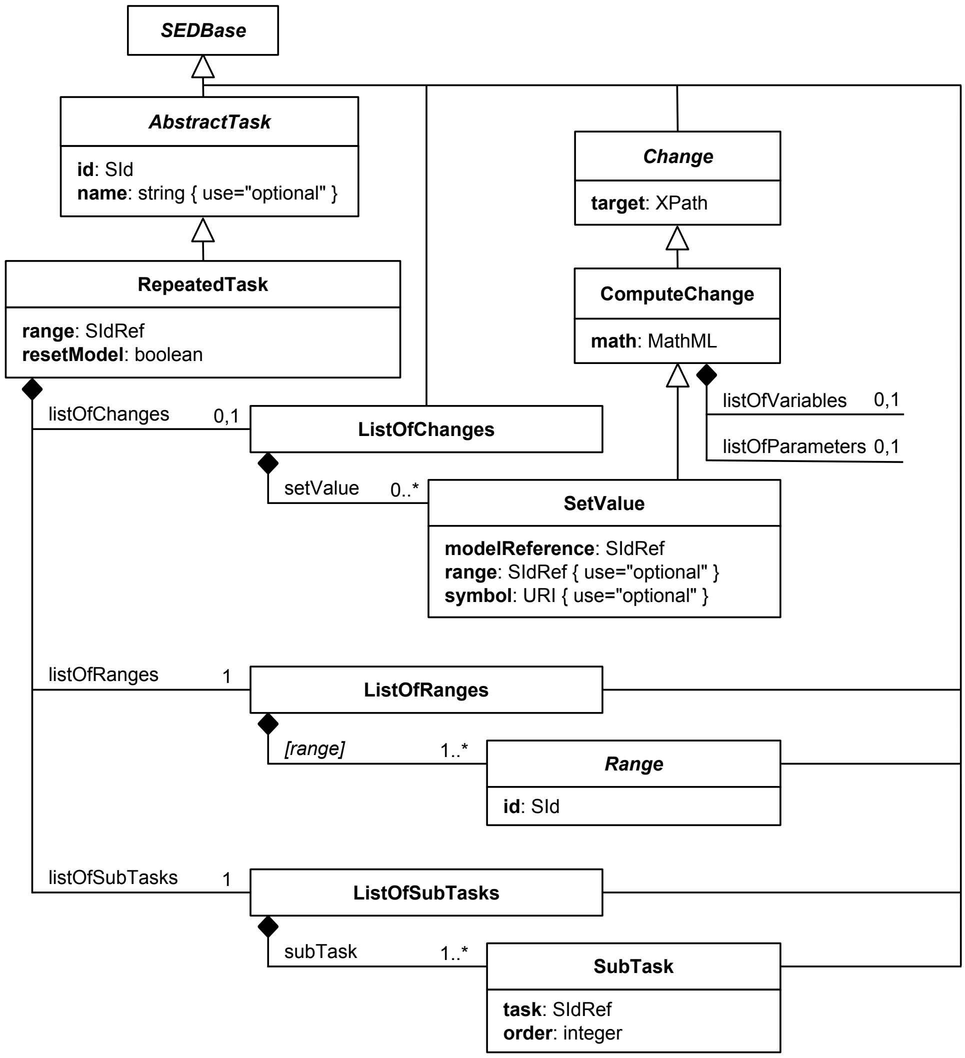

Figure 2.17: *The SED-ML RepeatedTask class*

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

Note that the order of activities within each iteration of a repeatedTask is as follows:

- the model is reset if specified by the `resetModel` attribute.
- any changes to the model specified by `setValue` elements are applied to the model.
- Finally, all `subTask` are executed in the order specified by the `order`.

Table 2.21 on the following page shows all attributes and sub-elements for the repeatedTask element.

Listing 2.51 shows the use of the `repeatedTask` element. In the example, `task1` is repeated three times, each time with a different value for a model parameter `w`.

```
<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>
```

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
range	page 50
resetModel	page 50
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfChanges^o	page 51
listOfRanges	page 50
listOfSubTask	page 51

Table 2.21: *Attributes and nested elements for repeatedTask. ^{o}denotes optional elements and attributes.*

```
  <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 2.51: *The* `repeatedTask` *element*

2.4.9.1 `range`

The `repeatedTask` class has a required attribute `range` of type `SIdRef`. It specifies which range defined in the `listOfRanges` this repeated task iterates over. Listing 2.51 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.

2.4.9.2 `resetModel`

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 2.51 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.

2.4.9.3 `listOfRanges`

The listOfRanges defines the ranges used in the repeatedTask. The `listOfRanges` contains one or more range elements. Three different range types are permitted in the listOfRanges: UniformRange, VectorRange, and FunctionalRange.

Ranges are the iterative element of the repeated simulation experiment. Each range defines a collection of values to iterate over. The `id` attribute of the ranges can be used to refer to the current value of a range. When 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.

2.4.9.4 listOfChanges

The optional `listOfChanges` element contains setValue elements. These elements allow the modification of values in the model prior to the next iteration of the repeatedTask.

2.4.9.5 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 the `order` attribute on the subTask. To specify that one `subTask` should be executed before another its `order` attribute must have a lower number (c.f. Listing 2.52).

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

Listing 2.52: *The `subTask` element. In this example the task `task2` must be executed before `task1`.*

2.4.10 Range

The Range class is the abstract base class for the different types of ranges, i.e. UniformRange, VectorRange, andFunctionalRange (Figure 2.18).

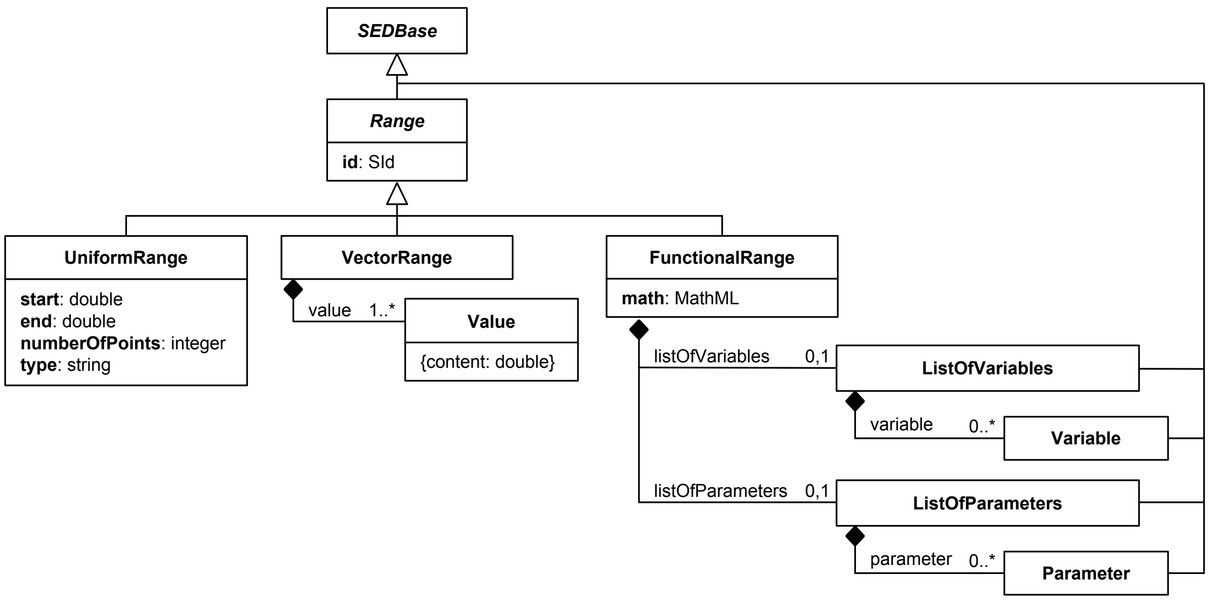

Figure 2.18: *The SED-ML Range class*

2.4.10.1 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 2.53: *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 2.54: *The* `UniformRange` *element with a logarithmic range.*

2.4.10.2 VectorRange

A `VectorRange` describes an ordered collection of real values, listing them explicitly within child `value` elements (Figure 2.18 on the previous page).

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 2.55: *The* `VectorRange` *element*

2.4.10.3 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 (Figure 2.18 on the preceding page). In this it is similar to the ComputeChange element, and shares some of the same child elements (but is no subclass of ComputeChange). It consists of an optional attribute `range`, two optional elements listOfVariables and listOfParameters, and a required element `math`.

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 referring to model variables or range values, which may then be used within the `math` 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>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <times/>
          <ci> w </ci>
          <ci> index </ci>
      </apply>
    </math>
</functionalRange>
```

Listing 2.56: *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">
    <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>
</functionalRange>
```

Listing 2.57: *A `functionalRange` element that returns `8` if `index` is smaller than `1`, `0.1` if `index` is between `4` and `6`, and `8` otherwise.*

2.4.11 `SetValue`

The SetValue class allows in a repeatedTask the modification of values in the model prior to the next execution of the subTasks. The changes to the model are hereby listed in the listOfChanges of the repeatedTask.

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 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.

The child `math` contains the expression computing the value by referring 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 2.58: *A `setValue` element setting `w` to the values of the range with id `current`.*

2.4.12 `DataGenerator`

The DataGenerator class prepares the raw simulation results for later output (Figure 2.19 on the next page). 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.

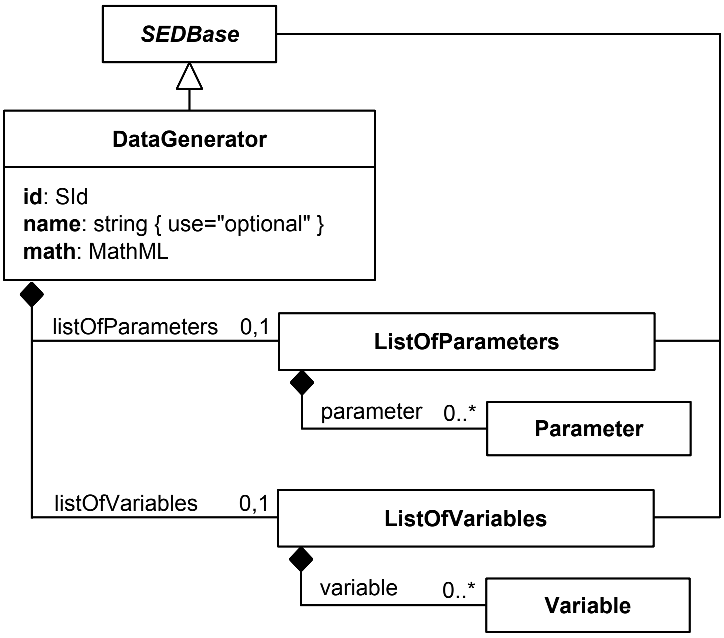

Figure 2.19: *The SED-ML DataGenerator class. Note that* `Parameter` *and* `Variable` *are subclasses of* SEDBase*; the respective inheritance connections are not shown in the figure.*

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 13. Variable and Parameter instances can be used to encode the mathematical expression.

Table 2.22 shows all attributes and sub-elements for the dataGenerator element.

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
listOfVariables^o	page 27
listOfParameters^o	page 29

Table 2.22: *Attributes and nested elements for dataGenerator. ^{o}denotes optional elements and attributes.*

Listing 2.59 shows the use of the `dataGenerator` element. In the example the `listOfDataGenerator` contains two `dataGenerator` elements. The first one, `d1`, refers to the task definition `task1` (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="task1"`). The model species with `id PX` is reused for the data generator `d2` without further post-processing.

```
<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 xmlns="http://www.w3.org/1998/Math/MathML">
            <math:ci>v1</math:ci>
        </math>
    </dataGenerator>
</listOfDataGenerators>
```

Listing 2.59: *Definition of two* `dataGenerator` *elements,* time *and* LaCI repressor

2.4.13 Output

The abstract Ouput class describes how the results of a simulation are presented. (Figure 2.20). The available output classes are plots (2D plot or 3D plot) and reports. The data used in the output plots or report is provided via dataGenerators.

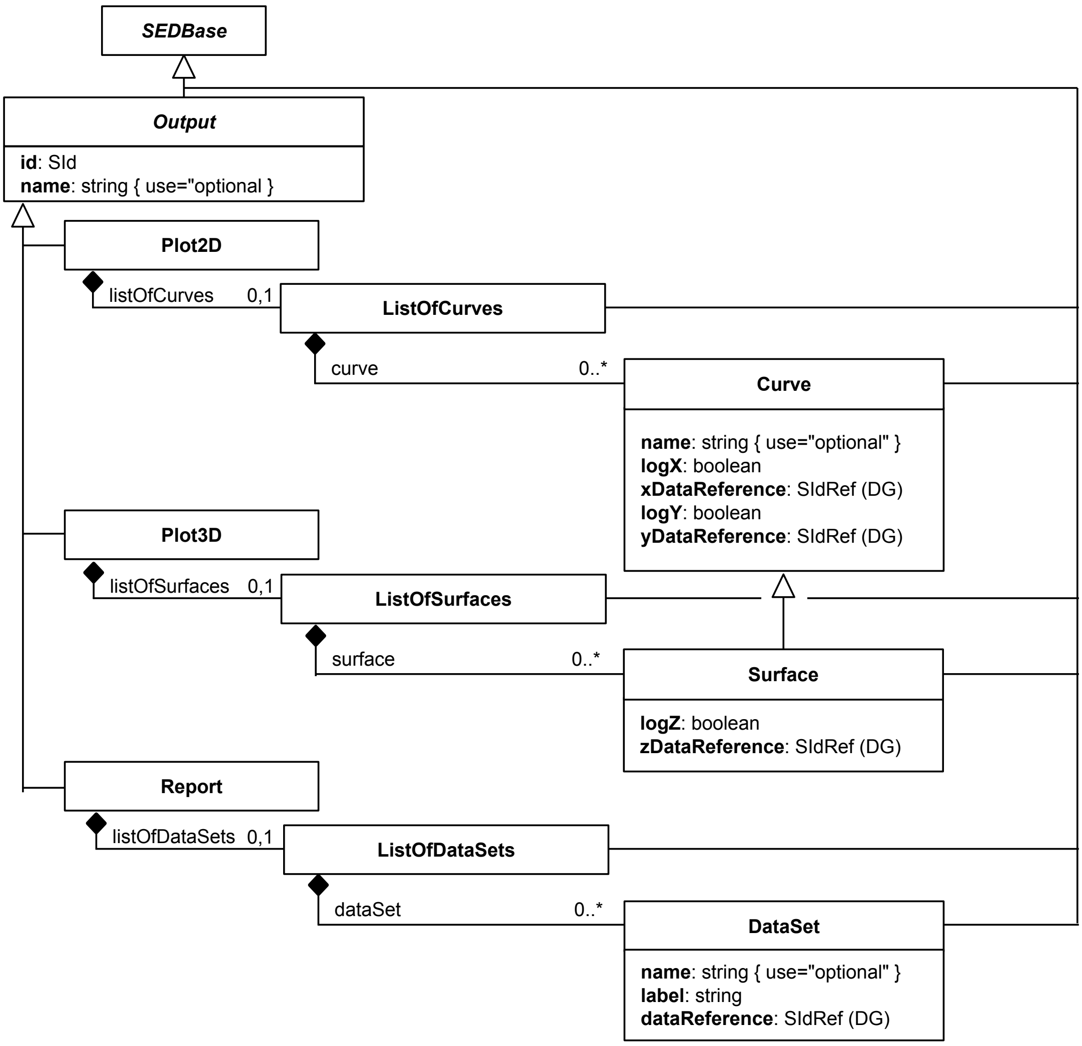

Figure 2.20: *The SED-ML Output class. Note that ListOfCurves, Curve, ListOfSurfaces, Surface, ListOfDataSets, DataSet and DataGenerator are subclasses of* SEDBase*; the respective inheritance connections are not shown in the figure.*

Note that even though the terms `Plot2D` and `Plot3D` 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 SED-ML Level 1 Version 3 alone.

2.4.13.1 Plot2D

The Plot2D class is used for two dimensional plot outputs (Figure 2.20 on the previous page). The Plot2D contains a number of `Curve` definitions, which define the curves which should be plotted in the the 2D plot. Table 2.23 shows all attributes and sub-elements for the plot2D element.

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

Table 2.23: *Attributes and nested elements for plot2D. ^{o}denotes optional elements and attributes.*

Listing 2.60 shows the use of the `listOfCurves` element. The example shows the definition of a `Plot2D` containing one `Curve` inside the `listOfCurves`.

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

Listing 2.60: *The `plot2D` element with the nested `listOfCurves` element.*

2.4.13.2 Plot3D

The `Plot3D` class is used for three dimensional plot outputs (Figure 2.20 on the preceding page). The `Plot3D` contains a number of `Surface` definitions. Table 2.24 shows all attributes and sub-elements for the plot3D element.

attribute	description
metaid^o	page 19
id	page 18
name^o	page 18
sub-elements	**description**
notes^o	page 19
annotation^o	page 20
listOfSurfaces^o	page 58

Table 2.24: *Attributes and nested elements for plot3D. ^{o}denotes optional elements and attributes.*

Listing 2.61 shows the use of the `plot3D` element. The example shows the definition of a `Surface` for the three dimensional plot inside the `listOfSurfaces`.

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

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

Listing 2.61: *The* `plot3D` *element with the nested* `listOfSurfaces` *element*

2.4.13.3 *Report*

The Report class defines a data table consisting of several single instances of the `DataSet` class (Figure 2.20 on page 55). Its output returns the simulation result processed via `DataGenerators` in actual *numbers*. The columns of the report table are defined by creating an instance of the DataSet class for each column. Table 2.25 shows all attributes and sub-elements for the report element.

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

Table 2.25: *Attributes and nested elements for report.* ^{o}*denotes optional elements and attributes.*

Listing 2.62 shows the use of the `listOfDataSets` element.

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

Listing 2.62: *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 not part of SED-ML Level 1 Version 3.

2.4.14 Output components

2.4.14.1 *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 2.20 on page 55). Table 2.26 on the next page shows all attributes and sub-elements for the curve element.

Listing 2.63 shows the use of the `curve` element. In the example a single curve is created. Results for the x-axis are generated by the dataGenerator `dg1`, results for the y-axis are generated by the dataGenerator `dg2`. Both `dg1` and `dg2` need to be defined in the listOfDataGenerators. The x-axis is plotted logarithmically.

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

Listing 2.63: *The SED-ML* `curve` *element, defining the output curve showing the result of simulation for the referenced dataGenerators*

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 2.26: *Attributes and nested elements for curve. ^{o}denotes optional elements and attributes.*

2.4.14.1.1 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 2.63.

2.4.14.1.2 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 2.63.

2.4.14.1.3 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 `SIdRef`. However, the valid values for the xDataReference are restricted to the IDs of already defined DataGenerator objects.

2.4.14.1.4 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 `SIdRef`. However, the valid values for the yDataReference are restricted to the IDs of already defined DataGenerator objects.

2.4.14.2 Surface

A surface is a three-dimensional figure representing a (processed) simulation result (Figure 2.20 on page 55). `Surface` is a subclass of `Curve` inheriting among others the elements `xDataReference`, `yDataReference`, `logX`, and `logY`.

Creating an instance of the Surface class requires the definition of data on three different axis. The aforementioned `xDataReference` and `yDataReference` attributes define the dataGenerators for 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 2.27 on the following page shows all attributes and sub-elements for the surface element.

Listing 2.64 shows the use of the `surface` element. In the example a single surface is created. Results shown on the x-axis are generated by the data generator `dg1`, results on the y-axis by dataGenerator `dg2`, results on the z-axis by dataGenerator `dg3`. All used dataGenerators, i.e. `dg1`, `dg2` and `dg3`, must be defined in the listOfDataGenerators.

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

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 59
zDataReference	page 59
sub-elements	**description**
notes^o	page 19
annotation^o	page 20

Table 2.27: *Attributes and nested elements for surface. ^{o}denotes optional elements and attributes.*

```
    [FURTHER SURFACE DEFINITIONS]
</listOfSurfaces>
```

Listing 2.64: *The SED-ML `surface` element, defining the output showing the result of the referenced task*

2.4.14.2.1 `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 2.64.

2.4.14.2.2 `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 `SIdRef`. However, the valid values for the zDataReference are restricted to the IDs of already defined DataGenerator objects.

2.4.14.3 `DataSet`

The DataSet class holds definitions of data to be used in the Report class (Figure 2.20 on page 55). DataSets are labeled references to instances of the DataGenerator class.

Table 2.28 shows all attributes and sub-elements for the dataSet element.

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

Table 2.28: *Attributes and nested elements for dataSet. ^{o}denotes optional elements and attributes.*

2.4.14.3.1 `label`

Each data set in a Report must have 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.

2.4.14.3.2 `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 dataSet in the report.

Listing 2.65 shows the use of the `dataSet` element. The example shows the definition of a dataSet. The referenced dataGenerator `dg1` must be defined in the listOfDataGenerators.

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

Listing 2.65: *The SED-ML* `dataSet` *element, defining a data set containing the result of the referenced task*

3. COMBINE archive

A COMBINE archive [2] is a single file that supports the exchange of all the information necessary for a modeling and simulation experiment in biology. A COMBINE archive file is a ZIP container that includes a manifest file, listing the content of the archive, an optional metadata file adding information about the archive and its content, and the files describing the model. The content of a COMBINE Archive consists of files encoded in COMBINE standards whenever possible, but may include additional files defined by an Internet Media Type. Several tools that support the COMBINE Archive are available, either as independent libraries or embedded in modeling software.

The COMBINE archive is described at `http://co.mbine.org/documents/archive` and in [2].

COMBINE archives are the recommended means for distributing simulation experiment descriptions in SED-ML, the respective data and model files, and the simulation results (figures and reports).

4. Acknowledgements

The SED-ML specification is developed with the input of many people. The following individuals served as past SED-ML Editors and contributed to SED-ML specifications. Their efforts helped shape what SED-ML is today.

- Richard Adams (editor, 2011-2012)
- Frank Bergmann (editor, 2011-2014)
- Jonathan Cooper (editor, 2012-2015)
- Nicolas Le Novère (editorial advisor, 2011-2012, 2013)
- Andrew Miller (editor, 2011-2012)
- Ion Moraru (editor, 2014-2016)
- Sven Sahle (editor, 2014-2016)
- Herbert Sauro

Moreover, we would like to thank all the participants of the meetings where SED-ML has been discussed as well as the members of the SED-ML community.

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).

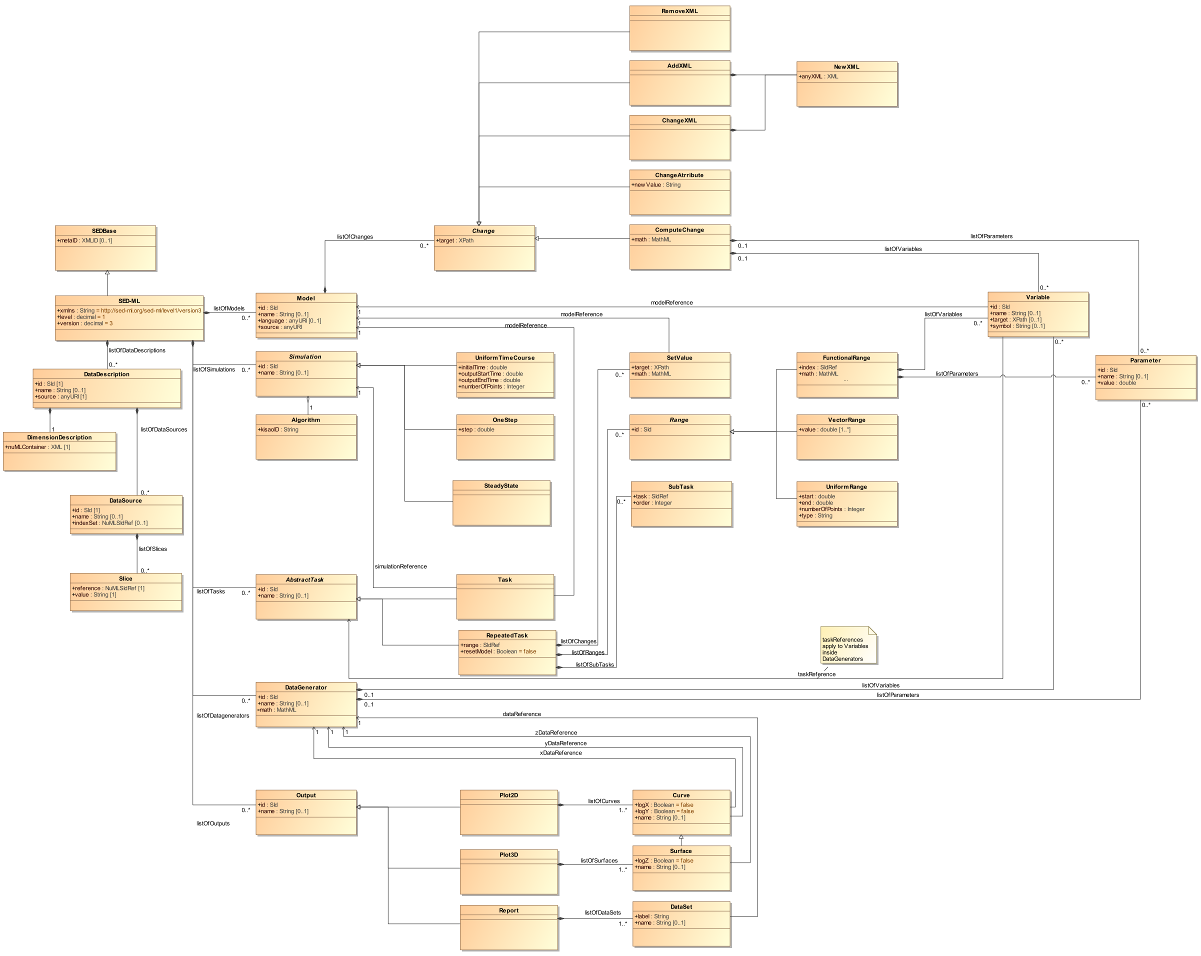

Figure A.1: *The SED-ML UML class diagram*

B. XML Schema

Listing B.1 shows the full SED-ML XML Schema.

```
<xs:schema targetNamespace="http://sed-ml.org/sed-ml/level1/version3" xmlns="http://sed-ml.org/sed-ml/level1/version3"
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:math="http://www.w3.org/1998/Math/MathML"
    elementFormDefault="qualified">
    <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 SED-ML 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 attributes -->
    <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>

    <!-- SED ML Top level element -->
    <xs:element name="sedML">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfDataDescriptions" minOccurs="0" />
                        <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="3" />
                </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]{7}" />
        </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>

    <!-- The model(s) to simulate/analyse -->
    <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>

    <!-- Model pre-processing changes -->
    <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:complexType name="ComputeChange">
        <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 name="computeChange" type="ComputeChange"/>


    <!-- The simulation/analysis algorithms to use -->
    <xs:element name="algorithm">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfAlgorithmParameters" minOccurs="0"/>
                    </xs:sequence>
                    <xs:attribute name="kisaoID" type="KisaoType" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="algorithmParameter">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="kisaoID" type="KisaoType" use="required"/>
                    <xs:attribute name="value" type="xs:string" use="required"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="Simulation">
        <xs:complexContent>
            <xs:extension base="SEDBase">
                <xs:sequence>
                    <xs:element ref="algorithm" />
                </xs:sequence>
                <xs:attributeGroup ref="idGroup" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:element name="uniformTimeCourse">
        <xs:complexType>
```

```
            <xs:complexContent>
                <xs:extension base="Simulation">
                    <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="oneStep">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="Simulation">
                    <xs:attribute name="step" type="xs:double" use="required"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="steadyState">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="Simulation">
                    <!-- There is actually no difference from the base type here -->
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <!-- The various task elements inherit from AbstractTask -->
    <xs:complexType name="AbstractTask">
        <xs:complexContent>
            <xs:extension base="SEDBase">
                <xs:attributeGroup ref="idGroup" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:element name="task">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="AbstractTask">
                    <xs:attribute name="simulationReference" type="SId" use="required" />
                    <xs:attribute name="modelReference" type="SId" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="repeatedTask">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="AbstractTask">
                    <xs:sequence>
                        <xs:element ref="listOfRanges"/>
                        <xs:element name="listOfChanges" type="repeatedTaskListOfChanges"
                            minOccurs="0"/>
                        <xs:element ref="listOfSubTasks"/>
                    </xs:sequence>
                    <xs:attribute name="range" type="SId" use="required"/>
                    <xs:attribute name="resetModel" type="SId" use="required"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <!-- Child elements of repeatedTask -->
    <xs:complexType name="Range">
        <xs:complexContent>
            <xs:extension base="SEDBase">
                <xs:attributeGroup ref="idGroup"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:simpleType name="LogOrLinear">
        <xs:restriction base="xs:string">
            <xs:enumeration value="log"/>
            <xs:enumeration value="linear"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:element name="uniformRange">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="Range">
                    <xs:attribute name="start" type="xs:double"/>
```

```
                <xs:attribute name="end" type="xs:double"/>
                <xs:attribute name="numberOfPoints" type="xs:integer"/>
                <xs:attribute name="type" type="LogOrLinear"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="vectorRange">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="Range">
                <xs:sequence>
                    <xs:element name="value" type="xs:double" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="functionalRange">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="Range">
                <xs:sequence>
                    <xs:element ref="listOfVariables" minOccurs="0" />
                    <xs:element ref="listOfParameters" minOccurs="0" />
                    <xs:element ref="math:math" />
                </xs:sequence>
                <xs:attribute name="range" type="SId" use="optional"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="setValue">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="ComputeChange">
                <xs:attribute name="modelReference" type="SId" use="required"/>
                <xs:attribute name="range" type="SId" use="optional"/>
                <xs:attribute name="symbol" type="xs:string" use="optional"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<xs:element name="subTask">
    <xs:complexType>
        <xs:complexContent>
            <xs:extension base="SEDBase">
                <xs:attribute name="task" type="SId" use="required"/>
                <xs:attribute name="order" type="xs:integer" use="optional"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:element>

<!-- Post-processing using a 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>

<!-- Simulation experiment outputs -->
<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="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 name="label" use="required" type="xs:string" />
                    <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="listOfAlgorithmParameters">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="algorithmParameter" 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:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="task" />
                        <xs:element ref="repeatedTask" />
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="listOfDataDescriptions">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="dataDescription"/>
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="dataDescription">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="dimensionDescription"/>
                        <xs:element ref="listOfDataSources"/>
                    </xs:sequence>
                    <xs:attribute name="source" type="xs:anyURI" use="required" />
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="dimensionDescription">
        <xs:complexType>
            <xs:sequence>
                <xs:any namespace="http://www.numl.org/numl/level1/version1"
                    processContents="skip" minOccurs="0" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="listOfDataSources">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="dataSource"/>
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="dataSource">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfSlices" minOccurs="0" maxOccurs="unbounded" />
                    </xs:sequence>
                    <xs:attribute name="indexSet" type="SId" use="optional" />
                    <xs:attributeGroup ref="idGroup" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
```

```
    </xs:element>

    <xs:element name="listOfSlices">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="slice"/>
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="slice">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:attribute name="reference" type="SId" use="required" />
                    <xs:attribute name="value" type="xs:string" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="listOfSimulations">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="uniformTimeCourse"/>
                        <xs:element ref="oneStep"/>
                        <xs:element ref="steadyState"/>
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="listOfOutputs">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="plot2D" />
                        <xs:element ref="plot3D" />
                        <xs:element ref="report" />
                    </xs:choice>
                </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>

    <xs:element name="listOfChanges">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element ref="changeAttribute" />
                        <xs:element ref="changeXML" />
                        <xs:element ref="addXML" />
                        <xs:element ref="removeXML" />
                        <xs:element ref="computeChange" />
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:element name="listOfRanges">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:choice maxOccurs="unbounded">
                        <xs:element ref="uniformRange" />
                        <xs:element ref="vectorRange" />
                        <xs:element ref="functionalRange" />
                    </xs:choice>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <xs:complexType name="repeatedTaskListOfChanges">
        <xs:complexContent>
            <xs:extension base="SEDBase">
                <xs:sequence>
                    <xs:element ref="setValue" minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <xs:element name="listOfSubTasks">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="subTask" maxOccurs="unbounded" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

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

C. Examples

This appendix presents selected SED-ML examples. These examples are only illustrative and do not intend to demonstrate the full capabilities of SED-ML. For a more comprehensive view of the SED-ML features refer to the specification (Chapter 2). Additional SED-ML examples are available from `http://sed-ml.org/`.

The presented examples use models encoded in SBML and CellML. SED-ML is not restricted to those formats, but can be used with models encoded in formats serialized in XML. A list of formats known to have been used 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) TODO: add reference.

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 [16].

A second model is defined in line 13 of the example, using `model1` as a source and applying additional 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 lines 24-65. Those are the prerequisite for defining the outputs of the simulation. The first dataGenerator named `time` collects the simulation time. `tim1` in line 33 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 line 41 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 (lines 67-72) 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.

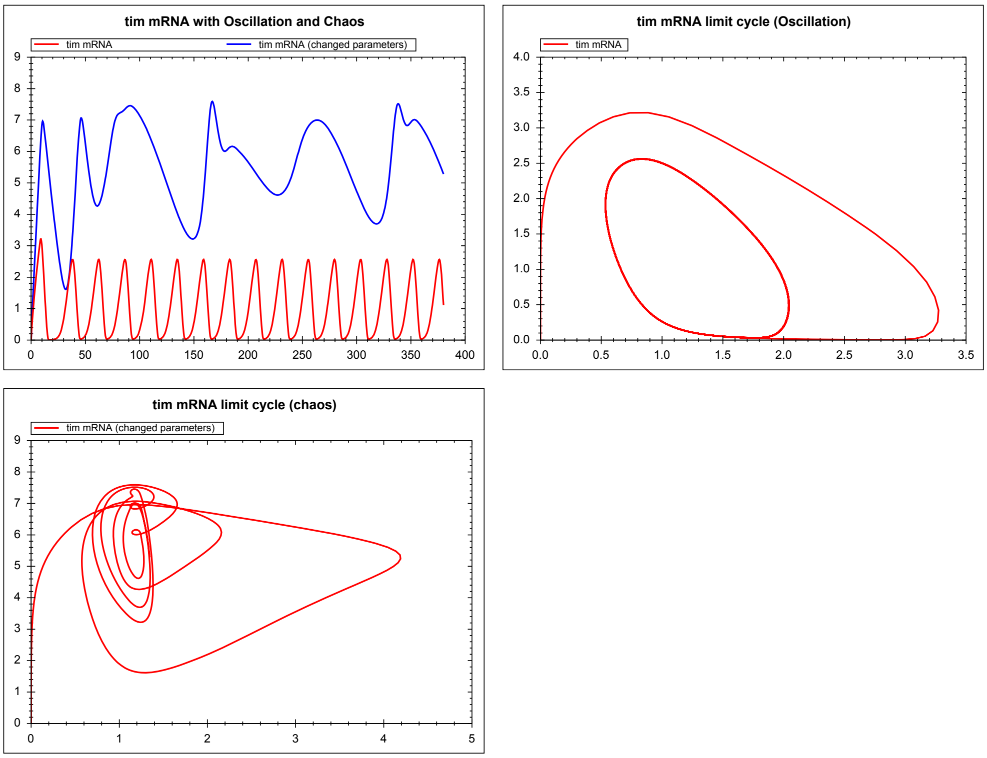

Figure C.1: *The simulation result gained from the simulation description given in Listing C.1*

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.4092.21172 see http://libsedml.sf.net -->
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V1.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3" level="1"
 version="3">
  <listOfSimulations>
    <uniformTimeCourse id="simulation1" initialTime="0" outputStartTime="0" outputEndTime="380" numberOfPoints="1000">
      <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 [18].

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 `dataGenerators` 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.

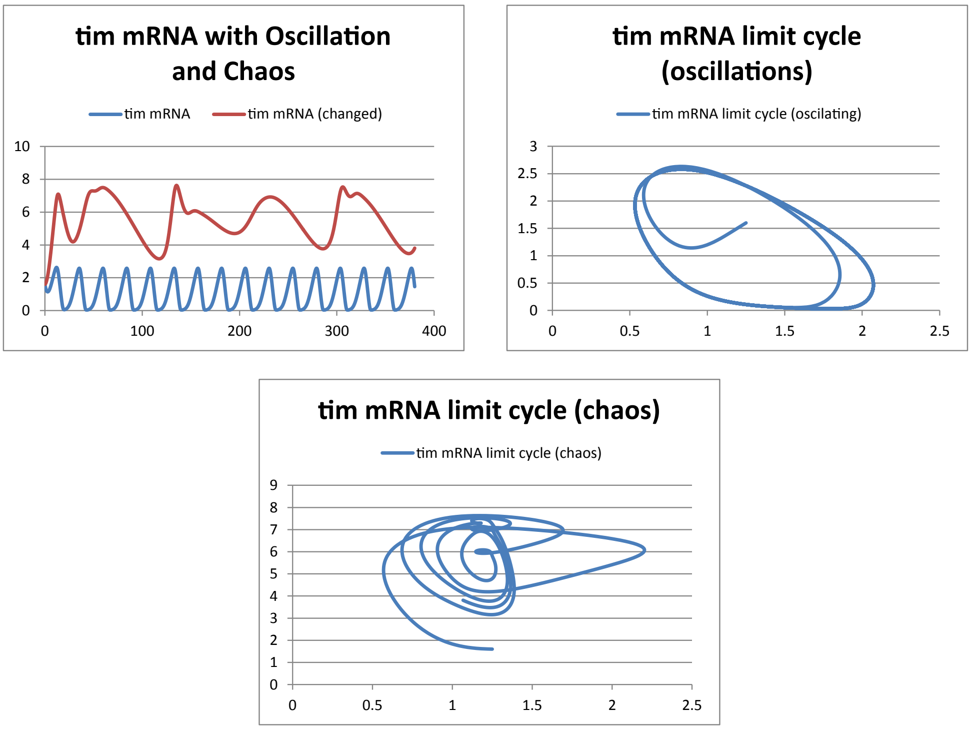

Figure C.2: *The simulation result gained from the simulation description given in Listing C.2*

```xml
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V1.xsd"
 xmlns:math="http://www.w3.org/1998/Math/MathML" xmlns="http://sed-ml.org/sed-ml/level1/version3" level="
      1" version="3">
 <notes><p xmlns="http://www.w3.org/1999/xhtml">Comparing Limit Cycles and strange attractors for
        oscillation in Drosophila</p></notes>
 <listOfSimulations>
   <uniformTimeCourse id="simulation1"
    initialTime="0" outputStartTime="0" outputEndTime="380"
    numberOfPoints="1000" >
      <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>
    </plot2D>
  </listOfOutputs>
</sedML>
```

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

C.3 IkappaB-NF-kappaB Signaling (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 [16].

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.

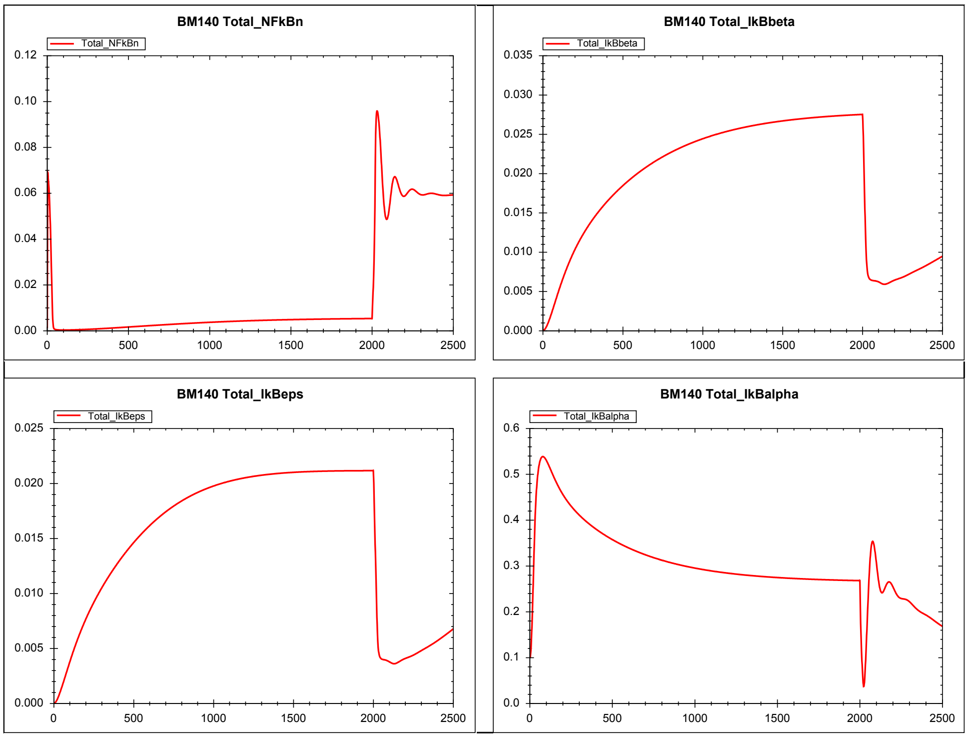

Figure C.3: *The simulation result gained from the simulation description given in Listing C.3*

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

```
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V1.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3
     " level="1"
 version="3">
  <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 Simulation Experiments with 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.

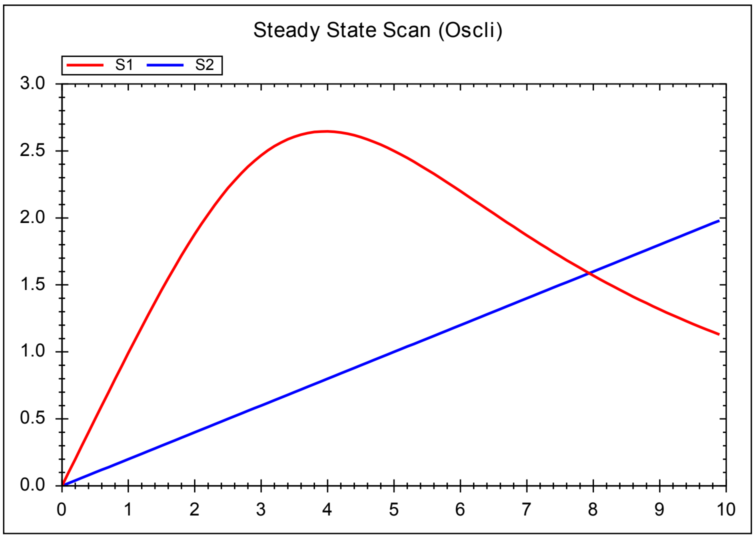

Figure C.4: *The simulation result gained from the simulation description given in Listing C.4*

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V2.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3" level="1"
 version="3">
  <listOfSimulations>
    <steadyState id="steady1">
      <algorithm kisaoID="KISAO:0000282" />
    </steadyState>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="http://sourceforge.net/p/libsedml/code/119/tree/trunk/Samples/models/oscli.xml?format=raw" />
  </listOfModels>
  <listOfTasks>
    <task id="task0" modelReference="model1" simulationReference="steady1" />
    <repeatedTask id="task1" resetModel="true" range="current">
      <listOfRanges>
        <uniformRange id="current" start="0" end="10" numberOfPoints="100" type="linear" />
      </listOfRanges>
      <listOfChanges>
        <setValue target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J0_v0']"
          range="current" modelReference="model1">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <ci> current </ci>
          </math>
        </setValue>
      </listOfChanges>
      <listOfSubTasks>
        <subTask order="1" task="task0" />
      </listOfSubTasks>
    </repeatedTask>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="J0_v0_1" name="J0_v0">
      <listOfVariables>
        <variable id="J0_v0" name="J0_v0" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J0_v0']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> J0_v0 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S1_1" name="S1">
      <listOfVariables>
        <variable id="S1" name="S1" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='S1']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S2_1" name="S2">
      <listOfVariables>
        <variable id="S2" name="S2" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='S2']" />
      </listOfVariables>
```

```
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S2 </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="Steady State Scan (Oscli)">
      <listOfCurves>
        <curve id="curve1" logX="false" logY="false" xDataReference="J0_v0_1" yDataReference="S1_1" />
        <curve id="curve2" logX="false" logY="false" xDataReference="J0_v0_1" yDataReference="S2_1" />
      </listOfCurves>
    </plot2D>
    <report id="report1" name="Steady State Values">
      <listOfDataSets>
        <dataSet id="col1" dataReference="J0_v0_1" label="J0_v0" />
        <dataSet id="col2" dataReference="S1_1" label="S1" />
        <dataSet id="col3" dataReference="S2_1" label="S2" />
      </listOfDataSets>
    </report>
  </listOfOutputs>
</sedML>
```

Listing C.4: *SED-ML document implementing the one dimensional steady state parameter scan*

C.4.2 Simulation Perturbation

Often it is interesting to see how the dynamic behavior of a model changes when some perturbations are applied to the model. In this example we include one repeated task that makes repeated use of a oneStep task (that advances an ODE integration to the next output step). During the steps one parameter is modified effectively causing the oscillations of a model to stop. Once the value is reset the oscillations recover.

Note: In the example below we use a functionalRange, although the same result could also be achieved using the setValue element directly.

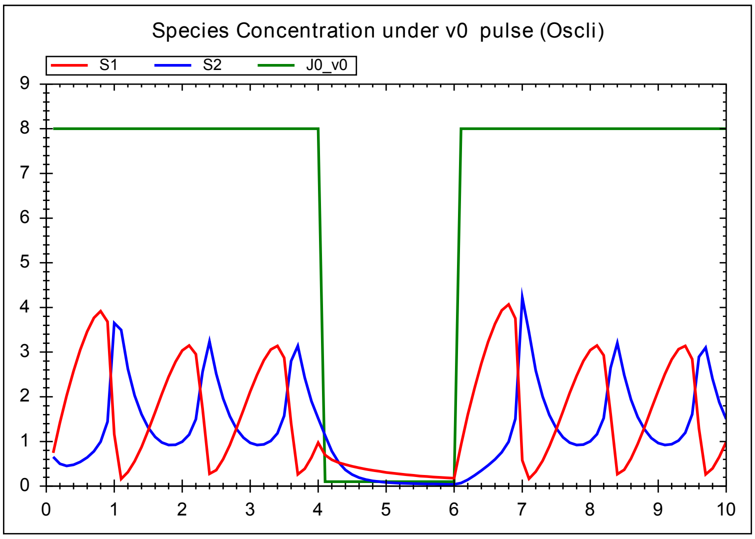

Figure C.5: *The simulation result gained from the simulation description given in Listing C.5*

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V2.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3
     " level="1"
 version="3">
  <listOfSimulations>
    <oneStep id="stepper" step="0.1">
      <algorithm kisaoID="KISAO:0000019" />
    </oneStep>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="http://sourceforge.net/p/libsedml/code
        /119/tree/trunk/Samples/models/oscli.xml?format=raw" />
  </listOfModels>
  <listOfTasks>
```

```
    <task id="task0" modelReference="model1" simulationReference="stepper" />
    <repeatedTask id="task1" resetModel="false" range="index">
      <listOfRanges>
        <uniformRange id="index" start="0" end="10" numberOfPoints="100" type="linear" />
        <functionalRange id="current" range="index">
          <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>
        </functionalRange>
      </listOfRanges>
      <listOfChanges>
        <setValue target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J0_v0']"
          range="current" modelReference="model1">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <ci> current </ci>
          </math>
        </setValue>
      </listOfChanges>
      <listOfSubTasks>
        <subTask order="1" task="task0" />
      </listOfSubTasks>
    </repeatedTask>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time_1" name="time">
      <listOfVariables>
        <variable id="time" name="time" taskReference="task1" target="time" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> time </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="J0_v0_1" name="J0_v0">
      <listOfVariables>
        <variable id="J0_v0" name="J0_v0" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfParameters/sbml:parameter[@id='J0_v0']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> J0_v0 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S1_1" name="S1">
      <listOfVariables>
        <variable id="S1" name="S1" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='S1']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S2_1" name="S2">
      <listOfVariables>
        <variable id="S2" name="S2" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='S2']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S2 </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
```

```
    <plot2D id="plot1" name="Species Concentration under v0  pulse (Oscli)">
      <listOfCurves>
        <curve id="curve1" logX="false" logY="false" xDataReference="time_1" yDataReference="S1_1" />
        <curve id="curve2" logX="false" logY="false" xDataReference="time_1" yDataReference="S2_1" />
        <curve id="curve3" logX="false" logY="false" xDataReference="time_1" yDataReference="J0_v0_1" />
      </listOfCurves>
    </plot2D>
    <report id="report1" name="Species Concentration under v0  pulse (Oscli)">
      <listOfDataSets>
        <dataSet id="col0" dataReference="time_1" label="time" />
        <dataSet id="col1" dataReference="J0_v0_1" label="J0_v0" />
        <dataSet id="col2" dataReference="S1_1" label="S1" />
        <dataSet id="col3" dataReference="S2_1" label="S2" />
      </listOfDataSets>
    </report>
  </listOfOutputs>
</sedML>
```

Listing C.5: *SED-ML document implementing the perturbation experiment*

C.4.3 Repeated Stochastic Simulation

NOTE: This example produces three dimensional results (time, species concentration, multiple repeats). While Level 1 Version 2 does not include a way to post-processing these values. So it is left to the implementation on how to display them. One example would be to flatten the values by overlaying them onto the desired plot.

Running just one stochastic trace does not provide a complete picture of the behavior of a system. A large number of traces are needed to provide a result. This example demonstrates the basic use case of running ten traces of a simulation to. This is achieved by running on repeatedTask running ten uniform time course simulations (each of which performing a stochastic simulation run).

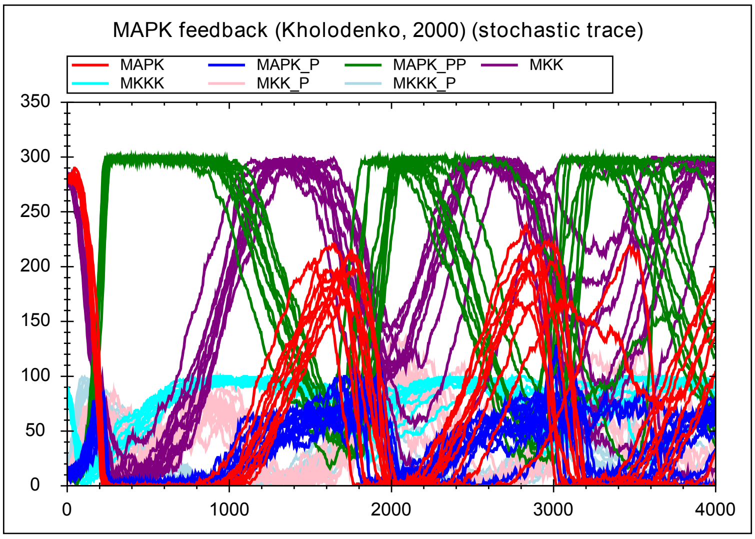

Figure C.6: *The simulation result gained from the simulation description given in Listing C.6*

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V2.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3
     " level="1"
 version="3">
  <listOfSimulations>
    <uniformTimeCourse id="timecourse1" initialTime="0" outputStartTime="0" outputEndTime="4000"
         numberOfPoints="1000">
      <algorithm kisaoID="KISAO:0000241" />
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="E:\Users\fbergmann\Documents\sbml
         models\borisejb.xml" />
  </listOfModels>
  <listOfTasks>
    <task id="task0" modelReference="model1" simulationReference="timecourse1" />
```

```xml
    <repeatedTask id="task1" resetModel="true" range="current">
      <listOfRanges>
        <uniformRange id="current" start="0" end="10" numberOfPoints="10" type="linear" />
      </listOfRanges>
      <listOfSubTasks>
        <subTask order="1" task="task0" />
      </listOfSubTasks>
    </repeatedTask>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time1" name="time">
      <listOfVariables>
        <variable id="time" taskReference="task1" symbol="urn:sedml:symbol:time" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> time </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MAPK1" name="MAPK">
      <listOfVariables>
        <variable id="MAPK" name="MAPK" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MAPK']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MAPK </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MAPK_P1" name="MAPK_P">
      <listOfVariables>
        <variable id="MAPK_P" name="MAPK_P" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MAPK_P']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MAPK_P </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MAPK_PP1" name="MAPK_PP">
      <listOfVariables>
        <variable id="MAPK_PP" name="MAPK_PP" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MAPK_PP']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MAPK_PP </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKK1" name="MKK">
      <listOfVariables>
        <variable id="MKK" name="MKK" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MKK']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKK </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKK_P1" name="MKK_P">
      <listOfVariables>
        <variable id="MKK_P" name="MKK_P" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MKK_P']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKK_P </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKKK1" name="MKKK">
      <listOfVariables>
        <variable id="MKKK" name="MKKK" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MKKK']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKKK </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKKK_P1" name="MKKK_P">
      <listOfVariables>
        <variable id="MKKK_P" name="MKKK_P" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MKKK_P']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKKK_P </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="MAPK feedback (Kholodenko, 2000) (stochastic trace)">
      <listOfCurves>
        <curve id="curve1" logX="false" logY="false" xDataReference="time1" yDataReference="MAPK1" />
        <curve id="curve2" logX="false" logY="false" xDataReference="time1" yDataReference="MAPK_P1" />
        <curve id="curve3" logX="false" logY="false" xDataReference="time1" yDataReference="MAPK_PP1" />
        <curve id="curve4" logX="false" logY="false" xDataReference="time1" yDataReference="MKK1" />
```

```
        <curve id="curve5" logX="false" logY="false" xDataReference="time1" yDataReference="MKKK1" />
        <curve id="curve6" logX="false" logY="false" xDataReference="time1" yDataReference="MKK_P1" />
        <curve id="curve7" logX="false" logY="false" xDataReference="time1" yDataReference="MKKK_P1" />
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing C.6: *SED-ML document implementing repeated stochastic runs*

C.4.4 One Dimensional Time Course Parameter Scan

NOTE: This example produces three dimensional results (time, species concentration, multiple repeats). While Level 1 Version 2 does not include a way to post-processing these values. So it is left to the implementation on how to display them. One example would be to flatten the values by overlaying them onto the desired plot.

Here one repeatedTask runs repeated uniform time course simulations (performing a deterministic simulation run) after each run the parameter value is changed.

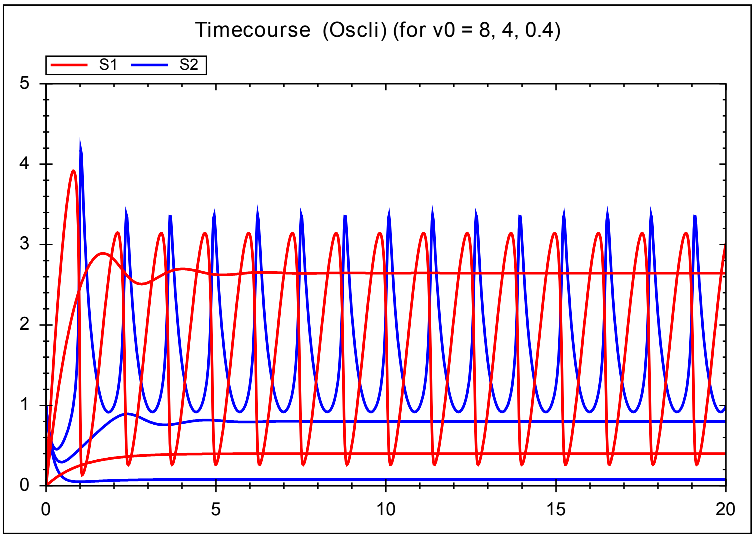

Figure C.7: *The simulation result gained from the simulation description given in Listing C.7*

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V2.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3
     " level="1"
 version="3">
  <listOfSimulations>
    <uniformTimeCourse id="timecourse1" initialTime="0" outputStartTime="0" outputEndTime="20"
         numberOfPoints="1000">
      <algorithm kisaoID="KISAO:0000019" />
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="http://sourceforge.net/p/libsedml/code
         /119/tree/trunk/Samples/models/oscli.xml?format=raw" />
  </listOfModels>
  <listOfTasks>
    <task id="task0" modelReference="model1" simulationReference="timecourse1" />
    <repeatedTask id="task1" resetModel="true" range="current">
      <listOfRanges>
        <vectorRange id="current">
          <value>8</value>
          <value>4</value>
          <value>0.4</value>
        </vectorRange>
      </listOfRanges>
      <listOfChanges>
        <setValue target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J0_v0']"
          range="current" modelReference="model1">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <ci> current </ci>
```

```
          </math>
        </setValue>
      </listOfChanges>
      <listOfSubTasks>
        <subTask order="1" task="task0" />
      </listOfSubTasks>
    </repeatedTask>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time1" name="time">
      <listOfVariables>
        <variable id="time" name="time" taskReference="task1" target="time" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> time </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="J0_v0_1" name="J0_v0">
      <listOfVariables>
        <variable id="J0_v0" name="J0_v0" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfParameters/sbml:parameter[@id='J0_v0']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> J0_v0 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S1_1" name="S1">
      <listOfVariables>
        <variable id="S1" name="S1" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='S1']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S2_1" name="S2">
      <listOfVariables>
        <variable id="S2" name="S2" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='S2']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S2 </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="Timecourse  (Oscli) (for v0 = 8, 4, 0.4)">
      <listOfCurves>
        <curve id="curve1" logX="false" logY="false" xDataReference="time1" yDataReference="S1_1" />
        <curve id="curve2" logX="false" logY="false" xDataReference="time1" yDataReference="S2_1" />
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing C.7: *SED-ML document implementing the one dimensional time course parameter scan*

C.4.5 Two Dimensional Steady State Parameter Scan

NOTE: This example produces three dimensional results (time, species concentration, multiple repeats). While Level 1 Version 2 does not include a way to post-processing these values. So it is left to the implementation on how to display them. One example would be to flatten the values by overlaying them onto the desired plot.

Here a repeatedTask runs over another repeatedTask which runs over a oneStep task (performing a steady state computation). Each repeated simulation task modifies a different parameter.

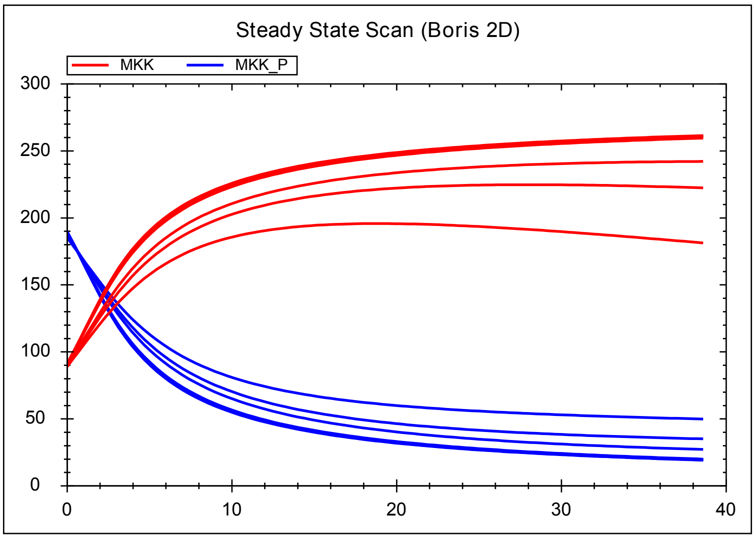

Figure C.8: *The simulation result gained from the simulation description given in Listing C.8*

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://sed-ml.org/ sed-ml-L1-V2.xsd" xmlns="http://sed-ml.org/sed-ml/level1/version3
      " level="1"
 version="3">
  <listOfSimulations>
    <steadyState id="steady1">
      <algorithm kisaoID="KISAO:0000282" />
    </steadyState>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="E:\Users\fbergmann\Documents\sbml
         models\borisejb.xml" />
  </listOfModels>
  <listOfTasks>
    <task id="task0" modelReference="model1" simulationReference="steady1" />
    <repeatedTask id="task1" resetModel="false" range="current">
      <listOfRanges>
        <vectorRange id="current">
          <value>1</value>
          <value>5</value>
          <value>10</value>
          <value>50</value>
          <value>60</value>
          <value>70</value>
          <value>80</value>
          <value>90</value>
          <value>100</value>
        </vectorRange>
      </listOfRanges>
      <listOfChanges>
        <setValue target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J1_KK2']"
          range="current" modelReference="model1">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <ci> current </ci>
          </math>
        </setValue>
      </listOfChanges>
      <listOfSubTasks>
        <subTask order="1" task="task2" />
      </listOfSubTasks>
    </repeatedTask>
    <repeatedTask id="task2" resetModel="false" range="current1">
      <listOfRanges>
        <uniformRange id="current1" start="1" end="40" numberOfPoints="100" type="linear" />
      </listOfRanges>
      <listOfChanges>
        <setValue target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J4_KK5']"
          range="current1" modelReference="model1">
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <ci> current1 </ci>
          </math>
        </setValue>
      </listOfChanges>
      <listOfSubTasks>
        <subTask order="1" task="task0" />
      </listOfSubTasks>
```

```
    </repeatedTask>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="J4_KK5_1" name="J4_KK5">
      <listOfVariables>
        <variable id="J4_KK5" name="J4_KK5" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfParameters/sbml:parameter[@id='J4_KK5']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> J4_KK5 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="J1_KK2_1" name="J1_KK2">
      <listOfVariables>
        <variable id="J1_KK2" name="J1_KK2" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfParameters/sbml:parameter[@id='J1_KK2']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> J1_KK2 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKK_1" name="MKK">
      <listOfVariables>
        <variable id="MKK" name="MKK" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MKK']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKK </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKK_P_1" name="MKK_P">
      <listOfVariables>
        <variable id="MKK_P" name="MKK_P" taskReference="task1" target="/sbml:sbml/sbml:model/
             sbml:listOfSpecies/sbml:species[@id='MKK_P']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKK_P </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="Steady State Scan (Boris 2D)">
      <listOfCurves>
        <curve id="curve1" logX="false" logY="false" xDataReference="J4_KK5_1" yDataReference="MKK_1" />
        <curve id="curve2" logX="false" logY="false" xDataReference="J4_KK5_1" yDataReference="MKK_P_1" /
             >
      </listOfCurves>
    </plot2D>
    <report id="report1" name="Steady State Values (Boris2D)">
      <listOfDataSets>
        <dataSet id="col0" dataReference="J4_KK5_1" label="J4_KK5" />
        <dataSet id="col1" dataReference="J1_KK2_1" label="J1_KK2" />
        <dataSet id="col2" dataReference="MKK_1" label="MKK" />
        <dataSet id="col3" dataReference="MKK_P_1" label="MKK_P" />
      </listOfDataSets>
    </report>
  </listOfOutputs>
</sedML>
```

Listing C.8: *SED-ML document implementing the one dimensional steady state parameter scan*

C.5 Referencing external data

This example demonstrates the use of the data sources in a basic SED-ML description. In this example a model is simulated (using a uniform time course simulation), that simulation result is plotted in one plot, a second plot obtains a stored result (using the data sources), extracts the `S1` and `time` column from it and renders it.

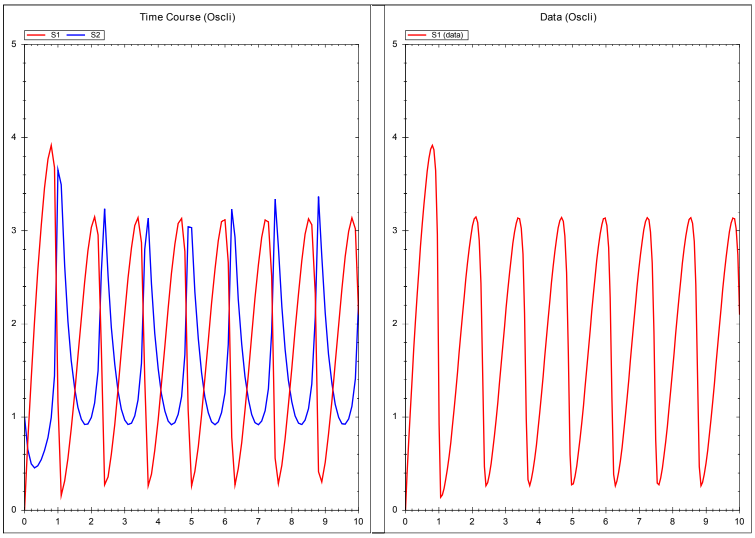

Figure C.9: *The simulation result gained from the simulation description given in Listing C.9*

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Written by libSedML v1.1.5335.19015 see http://libsedml.sf.net -->
<sedML level="1" version="3" xmlns="http://sed-ml.org/sed-ml/level1/version3">
  <listOfDataDescriptions>
    <dataDescription id="Data1" name="Oscli Time Course Data" source="http://svn.code.sf.net/p/libsedml/code/trunk/Samples/data/oscli.numl">
      <dimensionDescription>
        <compositeDescription indexType="double" id="time" name="time" xmlns="http://www.numl.org/numl/level1/version1">
          <compositeDescription indexType="string" id="SpeciesIds" name="SpeciesIds">
            <atomicDescription valueType="double" name="Concentrations" />
          </compositeDescription>
        </compositeDescription>
      </dimensionDescription>
      <listOfDataSources>
        <dataSource id="dataS1">
          <listOfSlices>
            <slice reference="SpeciesIds" value="S1" />
          </listOfSlices>
        </dataSource>
        <dataSource id="dataTime" indexSet="time" />
      </listOfDataSources>
    </dataDescription>
  </listOfDataDescriptions>
  <listOfSimulations>
    <uniformTimeCourse id="sim1" initialTime="0" outputStartTime="0" outputEndTime="10" numberOfPoints="100">
      <algorithm kisaoID="KISAO:0000019">
        <listOfAlgorithmParameters>
          <algorithmParameter kisaoID="KISAO:0000209" value="1E-06" />
          <algorithmParameter kisaoID="KISAO:0000211" value="1E-12" />
          <algorithmParameter kisaoID="KISAO:0000415" value="10000" />
        </listOfAlgorithmParameters>
      </algorithm>
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="http://sourceforge.net/p/libsedml/code/119/tree/trunk/Samples/models/oscli.xml?format=raw" />
  </listOfModels>
  <listOfTasks>
    <task id="task1" modelReference="model1" simulationReference="sim1" />
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time_1" name="time">
      <listOfVariables>
        <variable id="time" name="time" taskReference="task1" symbol="urn:sedml:symbol:time" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> time </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S1_1" name="S1">
      <listOfVariables>
        <variable id="S1" name="S1" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='S1']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
```

```xml
        <ci> S1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="S2_1" name="S2">
      <listOfVariables>
        <variable id="S2" name="S2" taskReference="task1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='S2']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> S2 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dgDataS1" name="S1 (data)">
      <listOfVariables>
        <variable id="varS1" modelReference="model1" target="#dataS1" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> varS1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dgDataTime" name="Time">
      <listOfVariables>
        <variable id="varTime" modelReference="model1" target="#dataTime" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> varTime </ci>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="plot1" name="Time Course (Oscli)">
      <listOfCurves>
        <curve id="curve1" logX="false" logY="false" xDataReference="time_1" yDataReference="S1_1" />
        <curve id="curve2" logX="false" logY="false" xDataReference="time_1" yDataReference="S2_1" />
      </listOfCurves>
    </plot2D>
    <plot2D id="plot2" name="Data (Oscli)">
      <listOfCurves>
        <curve id="curve3" logX="false" logY="false" xDataReference="dgDataTime" yDataReference="dgDataS1" />
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing C.9: *SED-ML document using DataSource and DataDescription*

Bibliography

[1] 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.

[2] F. T. Bergmann, R. Adams, S. Moodie, J. Cooper, M. Glont, M. Golebiewski, M. Hucka, C. Laibe, A. K. Miller, D. P. Nickerson, B. G. Olivier, N. Rodriguez, H. M. Sauro, M. Scharm, S. Soiland-Reyes, D. Waltemath, F. Yvon, and N. Le Novère. Combine archive and omex format: one file to share all information to reproduce a modeling project. *BMC bioinformatics*, 15:369, Dec. 2014.

[3] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax, 2005.

[4] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1 (Second Edition), 2006.

[5] D. Carlisle, P. Ion, R. Miner, and N. Poppelier. Mathematical Markup Language (MathML) version 2.0. *W3C Recommendation*, 21, 2001.

[6] J. Clarke and S. DeRose. XML Path Language (XPath) Version 1.0, 1999.

[7] M. Courtot, N. Juty, C. Knüpfer, D. Waltemath, A. Dräger, A. andFinney, M. Golebiewski, S. Hoops, S. Keating, D. 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, Oct. 2011.

[8] A. A. Cuellar, C. M. Lloyd, P. F. Nielson, M. D. B. Halstead, D. P. Bullivant, D. P. Nickerson, and P. J. Hunter. An overview of CellML 1.1, a biological model description language. *Simulation*, 79(12):740–747, 2003.

[9] J. Dada, I. Spasić, N. Paton, and P. Mendes. SBRML: a markup language for associating systems biology data with models. *Bioinformatics (Oxford, England)*, 26(7):932–938, April 2010.

[10] M. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, Jan. 2000.

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

[12] 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.

[13] 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.

[14] M. Hucka, F. Bergmann, S. Hoops, S. Keating, S. Sahle, and D. Wilkinson. The Systems Biology Markup Language (SBML): Language Specification for Level 3 Version 1 Core (Release 1 Candidate). *Nature Precedings*, January 2010.

[15] 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.

[16] 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.

[17] 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.

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

[19] OMG. *UML 2.2 Superstructure and Infrastructure*, February 2009.

[20] 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.

[21] W3C. XML Schema Part 1: Structures Second Edition. W3C Recommendation, October 2004.

[22] D. Waltemath, R. Adams, D. Beard, F. Bergmann, U. Bhalla, R. Britten, V. Chelliah, M. 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. Snoep, D. Tolle, O. Wolkenhauer, and N. Le Novère. Minimum information about a simulation experiment (MIASE). *PLoS Comput Biol*, 7:e1001122, 2011.

[23] D. Waltemath, R. Adams, F. T. Bergmann, M. Hucka, F. Kolpakov, A. K. Miller, I. I. Moraru, D. Nickerson, S. Sahle, J. L. Snoep, et al. Reproducible computational biology experiments with sed-ml-the simulation experiment description markup language. *BMC systems biology*, 5(1):198, 2011.