Frank T. Bergmann[1,2] / Jonathan Cooper[3] / Matthias König[4] / Ion Moraru[5] / David Nickerson[6] / Nicolas Le Novère[7] / Brett G. Olivier[1,2,8] / Sven Sahle[1] / Lucian Smith[9] / Dagmar Waltemath[10]

Simulation Experiment Description Markup Language (SED-ML) Level 1 Version 3 (L1V3)

[1] Modelling of Biol. Processes, BioQUANT/COS, Heidelberg University, Heidelberg, Germany. http://orcid.org/0000-0001-5553-4702, https://orcid.org/0000-0002-5293-5321.
[2] Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA, USA. http://orcid.org/0000-0001-5553-4702, https://orcid.org/0000-0002-5293-5321.
[3] Department of Computer Science, University of Oxford, Oxford, UK. https://orcid.org/0000-0001-6009-3542.
[4] Department of Biology, Humboldt University, Berlin, Germany, E-mail: koenigmx@hu-berlin.de. http://orcid.org/0000-0003-1725-179X.
[5] Department of Cell Biology, University of Connecticut, Connecticut, USA
[6] Auckland Bioengineering Institute, Auckland, New Zealand. https://orcid.org/0000-0003-4667-9779.
[7] Babraham Institute Cambridge, Cambridgeshire, UK. https://orcid.org/0000-0002-6309-7327.
[8] Systems Bioinformatics, AIMMS, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands. https://orcid.org/0000-0002-5293-5321.
[9] University of Washington, Seattle, WA, USA. https://orcid.org/0000-0001-7002-6386.
[10] University of Rostock, Rostock, Germany. https://orcid.org/0000-0002-5886-5563.

Abstract:
The creation of computational simulation experiments to inform modern biological research poses challenges to reproduce, annotate, archive, and share such experiments. Efforts such as SBML or CellML standardize the formal representation of computational models in various areas of biology. The Simulation Experiment Description Markup Language (SED-ML) describes what procedures the models are subjected to, and the details of those procedures. These standards, together with further COMBINE standards, describe models sufficiently well for the reproduction of simulation studies among users and software tools. The Simulation Experiment Description Markup Language (SED-ML) is an XML-based format that encodes, for a given simulation experiment, (i) which models to use; (ii) which modifications to apply to models before simulation; (iii) which simulation procedures to run on each model; (iv) how to post-process the data; and (v) how these results should be plotted and reported. SED-ML Level 1 Version 1 (L1V1) implemented support for the encoding of basic time course simulations. SED-ML L1V2 added support for more complex types of simulations, specifically repeated tasks and chained simulation procedures. SED-ML L1V3 extends L1V2 by means to describe which datasets and subsets thereof to use within a simulation experiment.

Keywords: Simulation experiment, computational modeling, reproducibility
DOI: 10.1515/jib-2017-0086
Received: December 31, 2017; **Accepted:** February 2, 2018

Matthias König is the corresponding author.

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

December 31, 2017

Editors

Frank T Bergmann	*Caltech, USA*
Jonathan Cooper	*University of Oxford, UK*
Matthias König	*Humboldt University Berlin, Germany*
Ion Moraru	*University of Connecticut, USA*
David Nickerson	*Auckland Bioengineering Institute, New Zealand*
Nicolas Le Novère	*Babraham Institute Cambridge, UK*
Brett G Olivier	*University Amsterdam, Netherlands*
Sven Sahle	*Heidelberg University, Germany*
Lucian Smith	*University of Washington, USA*
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 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 standardize the representation of computational models in various areas of biology, such as the Systems Biology Markup Language (SBML) [14], CellML [8] or NeuroML [11], resulted in an increase of the exchange and re-use of models. However, the description of models is not sufficient for the reproduction of simulation experiments and results. One also needs to describe the procedures the models are subjected to, i.e., the information that must be provided to allow the reproduction of simulation experiments among users and software tools. 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 for the reproduction of simulation experiments. SED-ML is a software-independent format encoded in XML not specific to particular simulation tools and independent of the underlying model language. SED-ML describes the minimum information of a simulation experiment as described by the Minimum Information About a Simulation Experiment (MIASE) [19].

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 [20]).

1.1 SED-ML overview

SED-ML specifies for a given simulation experiment

- what datasets to use (DataDescription);
- which models to use (Model);
- which modifications to apply to models before simulation (Change);
- which simulation procedures to run on each model (Simulation, and Task);
- what analysis results to plot or report and how to post-process the data (DataGenerator); and
- how these results should be presented (Output).

A SED-ML document contains the following main objects to describe this information: DataDescription, Model, Change, Simulation, Task, DataGenerator, and Output.

DataDescription

The DataDescription class allows to specify datasets for a simulation experiment. Such data can be used for instance for parametrization of model simulations or to plot data together with simulation results.

Model

The Model class allows to reference the models used in a simulation experiment.

The Change class allows to modify models (pre-processing), i.e., 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 class defines the simulation settings and the steps taken during simulation. These include the particular type of simulation, the algorithm, and the algorithm parameters used for the execution of the simulation.

Task

SED-ML uses the Task class to specify which Simulation is run with which Model.

DataGenerator

The DataGenerator class allows to encode post-processing of simulation results before the generation of outputs, e.g., one can normalize a variable, or apply post-processing like mean value calculation. In the definition of a DataGenerator, any addressable variable or parameter of any model or DataSource may be referenced, and new entities might be specified using MathML.

Output

The Output defines the output of the simulation experiment, which can be either a two dimensional plot (Plot2D), a three dimensional plot (Plot3D), or data table (Report). The Output is based on the post-processed simulation results in the DataGenerators.

This section provided a low level overview over a simulation experiment in SED-ML. For the detailed technical specification see Chapter 2.

1.2 Example simulation experiment

In this section an example simulation experiment in SED-ML for the repressilator model [9] is presented. The corresponding SED-ML is listed in Appendix A.1, the COMBINE Archive for this simulation experiment is available as `L1V3_repressilator.omex` from `http://sed-ml.org/`.

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) participate 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.

1.2.1 Time-course simulation

The first simulation experiment with the model reproduces the oscillation behavior of the model shown in Figure 1c of the reference publication [9]. This simulation experiment can be described as:

1. Import the repressilator model identified by the Unified Resource Identifier (URI) [3] `urn:miriam:biomodels.db:BIOMD0000000012`.
2. Select a deterministic simulation method for the numerical integration.
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 experiment in a simulation tool supporting SED-ML results in the output shown in Figure 1.1 and Figure 1.2.

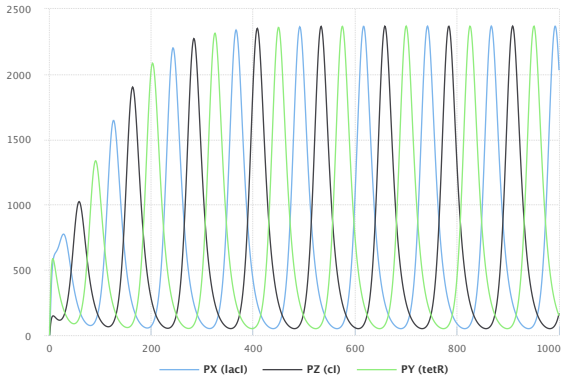

Figure 1.1: *Time-course simulation of the repressilator depicting repressor proteins lacI, tetR and cI. Simulation with SED-ML web tools [2].*

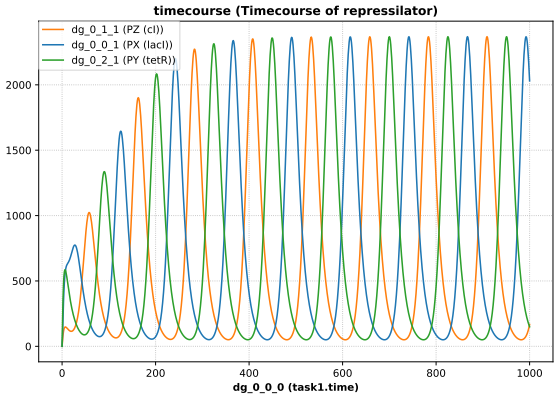

Figure 1.2: *Time-course simulation of the repressilator depicting repressor proteins lacI, tetR and cI. Simulation with tellurium [5].*

1.2.2 Applying pre-processing

A common step in a simulation experiment is the adjustment of model parameters before simulation. When changing the parameter values for the protein copies per promoter `tps_repr` and the leakiness in protein copies per promoter `tps_active` like stated below, the system's behavior switches from sustained oscillations to damped oscillations. The simulation experiment leading to that behavior is described as:

1. Import the model as in Section 1.2.1 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 1.3 on the following page and Figure 1.4 on the next page show the results of the simulation.

1.2.3 Applying post-processing

In a simulation experiment the raw numerical output of the simulation 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-plane), one performs 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)`.
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.

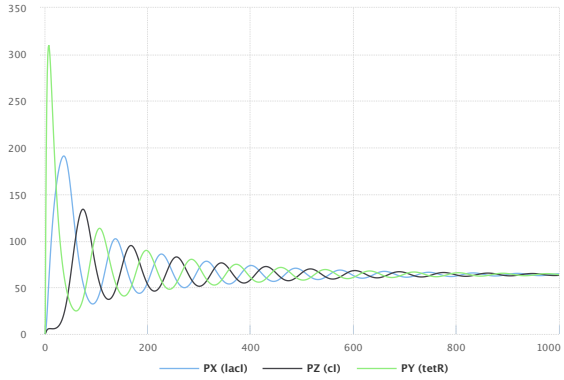

Figure 1.3: *Time-course simulation of the repressilator after changing parameters* `tps_repr` *and* `tps_active`*. Simulation with SED-ML web tools [2].*

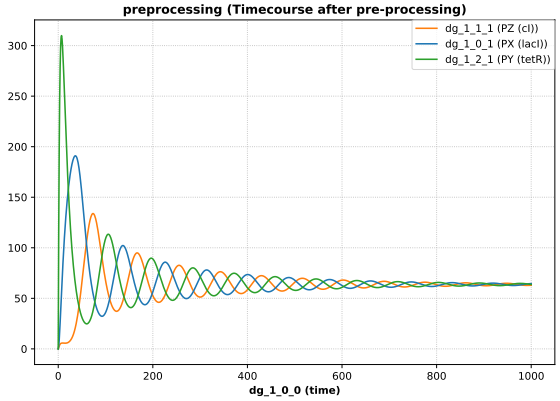

Figure 1.4: *Time-course simulation of the repressilator after changing parameters* `tps_repr` *and* `tps_active`*. Simulation with tellurium [5].*

Figure 1.5 and Figure 1.6 show the result of the simulation after post-processing of the output data.

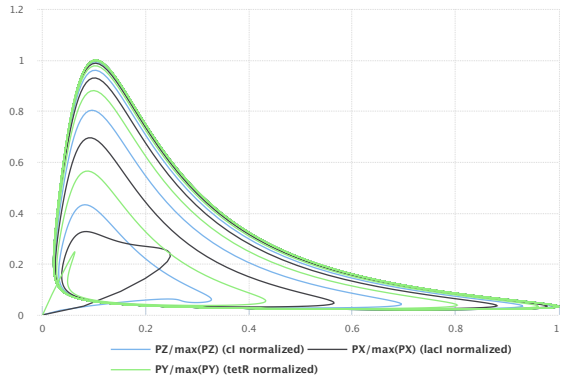

Figure 1.5: *Time-course simulation of the repressilator. Normalized* `lacI`*,* `tetR` *and* `cI` *in phase-plane. Simulation with SED-ML web tools [2].*

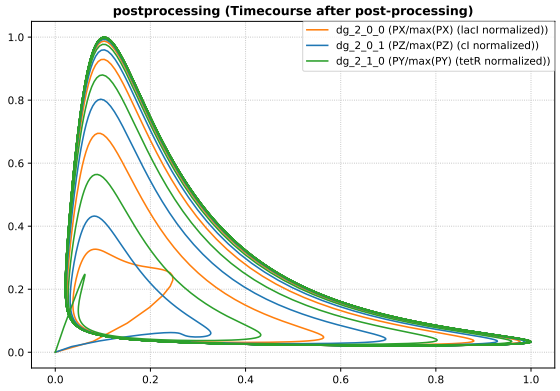

Figure 1.6: *Time-course simulation of the repressilator. Normalized* `lacI`*,* `tetR` *and* `cI` *in phase-plane. Simulation with tellurium [5].*

2. SED-ML technical specification

This document represents the technical specification of SED-ML Level 1 Version 3. The corresponding UML class diagram is shown in Figure 2.1. Example simulation experiments in SED-ML are provided in Appendix A. The XML Schema is provided in Appendix B. However, not all concepts of SED-ML can be captured using XML Schema alone. In such cases this specification is the normative document.

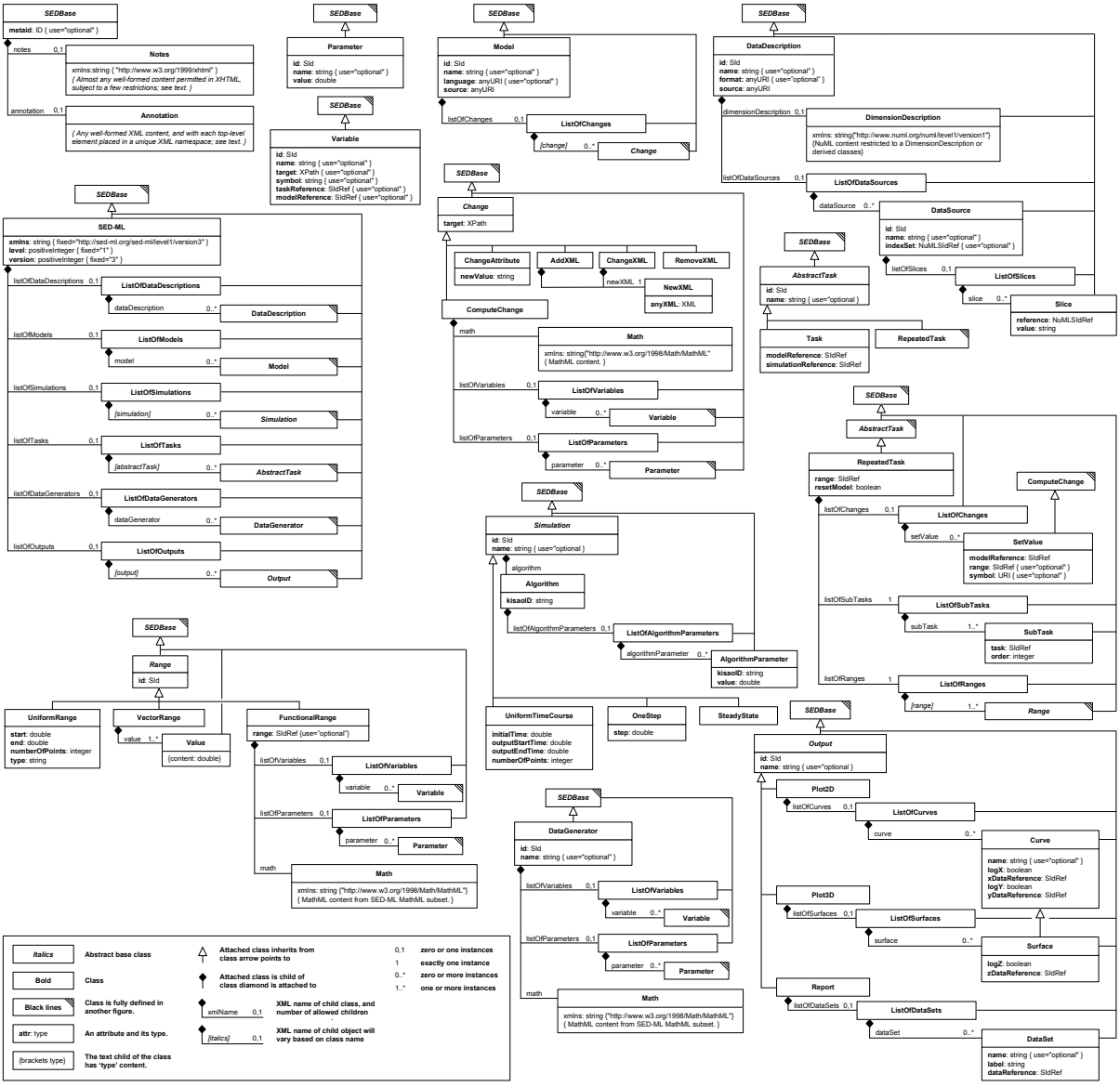

Figure 2.1: *The SED-ML Level 1 Version 3 UML class diagram*

2.1 General data types, attributes and classes

In this section concepts used repeatedly throughout the SED-ML specification are introduced. This includes primitive data types, classes (SedBase, Notes, Annotation, Parameter, Variable), attributes, and reference relations.

The main SED-ML components based on these general data types, attributes and classes are described in Section 2.2.

2.1.1 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 additional primitive types are defined by SED-ML itself: `ID`, `SId`, `SIdRef`, `XPath`, `MathML`, `anyURI`, `NuMLSId`, and `NuMLSIdRef`.

2.1.1.1 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 2.2. For a detailed description see the SBML specification on type `ID` [13].

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

Figure 2.2: *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.*

2.1.1.2 Type SId

The type `SId` is the type of the id attribute found on the majority of SED-ML components. `SId` is a data type derived from `string`, but with restrictions about the characters permitted and the sequences in which those characters may appear. The definition is shown in Figure 2.3. For a detailed description see the SBML specification on type `SId` [13].

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

Figure 2.3: *The definition of the type SId*

2.1.1.3 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 other words, a `SIdRef` value must be an existing identifier.

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.

2.1.1.4 Type XPath

Type `XPath` is used to identify nodes and attributes within an XML representation. `XPath` in SED-ML is an XPath version 1 expression which can be used to unambiguously identify an element or attribute in an XML file. The concept of XPath is described in Section 3.3.

2.1.1.5 Type MathML

Type MathML is used to describe mathematical expression in MathML. The concept of MathML and the allowed subset of MathML on a MathML attribute is described in Section 3.1.

2.1.1.6 Type anyURI

Type anyURI is used to reference model and data files, specify the language of models, the format of data files, for referencing implicit model variables, and in annotations. For a description of the uses of anyURI see Section 3.2.

2.1.1.7 Type NuMLSId

The type NuMLSId is the type of the id attribute found on NuML components. NuMLSId is a data type derived from SId, with the same restrictions about the characters permitted and the sequences in which those characters may appear as SId. The concept of NuML is described in Section 3.4.

2.1.1.8 Type NuMLSIdRef

Type NuMLSIdRef is used for all attributes that refer to identifiers of type NuMLSId in a model. This type is derived from NuMLSId, but with the restriction that the value of an attribute having type NuMLSIdRef must equal the value of some NuMLSId attribute. In other words, a NuMLSIdRef value must be an existing NuML identifier.

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

2.1.2 SEDBase

SEDBase is the base class of all SED-ML classes (Figure 2.4). The SEDBase class has the optional attribute metaid, and the two optional subelements notes and annotation.

SEDBase provides means to attach additional information on all other classes. That information can be specified by human readable Notes or custom Annotation.

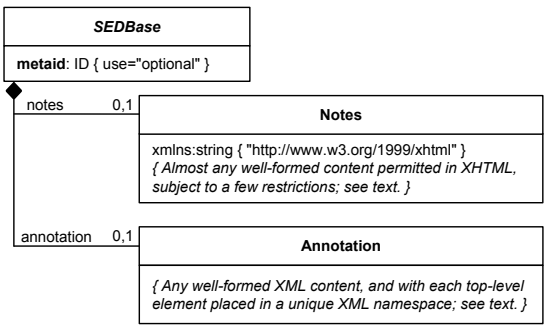

Figure 2.4: *The SEDBase class*

attribute	description
metaid^o	page 12
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

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.

In order to set either Notes or Annotation on a SED-ML class the metaid is required.

notes

The optional notes element stores Notes on SedBase.

annotation

The optional annotation element stores Annotation on SedBase.

2.1.3 Notes

A Notes is considered a human-readable description of the element it is assigned to. Instances of the Notes class may contain any valid XHTML [18]. The namespace URL for XHTML content inside the Notes class is http://www.w3.org/1999/xhtml, which may be declared either in the sedML element, or directly in the top level XHTML elements contained within the notes element. For details on of how to set the namespace and examples see the SBML specification [13].

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

attribute	description
xmlns:string "http://www.w3.org/1999/xhtml"	page 23
sub-elements	
well-formed content permitted in XHTML	

Table 2.2: *Attributes and nested elements for Notes. °denotes optional elements and attributes.*

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

Listing 2.1 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 2.1: *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.

2.1.4 Annotation

An Annotation is considered a computer-processable piece of information. Annotations may contain any valid XML content. For further guidelines on how to use annotations see the SBML specification [13]. The style of annotations in SED-ML is briefly described in Section 3.2.6.

Listing 2.2 shows the use of the annotation element. In the example, a model element is annotated with a reference to the original publication. The model contains an annotation that uses the model-qualifier isDescribedBy to link to the external resource http://identifiers.org/pubmed/10415827. In natural language the annotation content could be interpreted as "The model is described by the published article available from pubmed under the identifier 10415827".

```
<sedML>
    [..]
    <model id="model1" metaid="_001" language="urn:sedml:language:cellml" source="goldbeter1999a.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="http://identifiers.org/pubmed/10415827"/>
                </rdf:Bag>
                </bqmodel:isDescribedBy>
                </rdf:Description>
            </rdf:RDF>
        </annotation>
    </model>
```

```
16      [..]
17  </sedML>
```

Listing 2.2: *The annotation element*

2.1.5 Parameter

The Parameter class (Figure 2.5) is used to create named parameters with a constant value. The Parameter class introduces the required attributes `id` and `value`, and the optional attribute `name`.

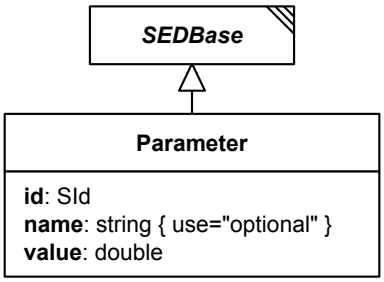

Figure 2.5: *The Parameter class*

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
value	page 14
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

Parameters can be used wherever a mathematical expression to compute a value is defined, e.g., in ComputeChange, FunctionalRange or DataGenerator. The Parameter definitions are local to the particular class defining them. By using Parameters rather than including numbers directly within a mathematical expression is that notes and annotations can be associated with them.

Listing 2.3 shows the use of the `parameter` element. In the example a parameter `p1` with the value `40` is defined.

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

Listing 2.3: *The definition of a parameter in SED-ML*

value

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

2.1.6 Variable

A Variable (Figure 2.6 on the next page) is a reference to an already existing entity, either to an object in one of the Models or to implicitly defined Symbols. The Variable class introduces the required attribute `id`, the optional attribute `name`, and the context dependent attributes `target`, `symbol`, `taskReference`, and `modelReference`.

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 a Symbol, rather than one explicitly appearing in the model, then the `symbol` attribute is used.

- 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.
- A Variable element must contain a taskReference if it occurs inside a `listOfVariables` inside a dataGenerator element. Only exception is if the Variable references a DataSource, in this case no taskReference is required.

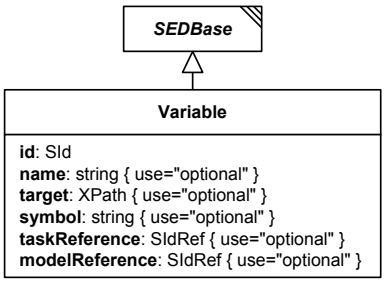

Figure 2.6: *The Variable class*

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
target	page 15
symbol	page 16
taskReference	page 19
modelReference	page 18
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

Table 2.4: *Attributes and nested elements for Variable. ^{o}denotes optional elements and attributes.*

- 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 2.4 shows the use of the `variable` element. In the example a variable `v1` is defined to compute a change on a model constituent (referenced by the `target` attribute on `computeChange`). The value of `v1` corresponds to the value of the targeted model constituent referenced by the `target` attribute. The second variable `v2` 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.

```
<sedML>
    <listOfModels>
        <model [..]>
            <listOfChanges>
                <computeChange target="TARGET ELEMENT OR ATTRIBUTE">
                <listOfVariables>
                   <variable id="v1" name="maximum velocity" target="XPath TO MODEL ELEMENT/ATTRIBUTE" />
                   [FURTHER VARIABLE DEFINITIONS]
                </listOfVariables>
                [..]
                </computeChange>
            </listOfChanges>
            [..]
        </model>
        [..]
    </listOfModels>
    <listOfDataGenerators>
        <dataGenerator [..]>
            <listOfVariables>
                <variable id="v2" name="time" taskReference="task1" symbol="urn:sedml:symbol:time" />
                [FURTHER VARIABLE DEFINITIONS]
            </listOfVariables>
        </dataGenerator>
    </listOfDataGenerators>
    [..]
</sedML>
```

Listing 2.4: *SED-ML `variable` definitions inside the `computeChange` element and inside the `dataGenerator` element*

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 `SId` of the targeted element. As of SED-ML Level 1 Version 3 this is used to refer to a DataSource in a Variable or to refer to ranges within a repeatedTask (see Listing 2.44).

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.

Listing 2.5 shows the use of the `target` attribute in a SED-ML file. In the example the `target` is used to reference a species with `id='PY'` in an SBML model.

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

Listing 2.5: *SED-ML `target` definition*

It should be noted that the identifiers and names inside the SED-ML document do not have to match the identifiers and names that the model and its constituents have in the model definition. In Listing 2.5, the variable with ID `v1` is defined. It is described as `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 2.6 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 2.5):

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

Listing 2.6: *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 2.7: *Species definition in the referenced model*

The XPath expression used in the `target` attribute unambiguously leads to the particular place in the SBML model, i.e., the species is to be found in the *sbml* element, and there inside the *listOfSpecies* (Listing 2.7).

symbol

The `symbol` element is used to refer to a Symbol. Symbols are predefined, implicit variables, which can be used 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 3.2.5.

Listing 2.8 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 2.2.5.6.

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

Listing 2.8: *SED-ML `symbol` definition*

taskReference

The `taskReference` element of data type `SIdRef` is used to reference a Task via a taskReference. The usage depends on the context the Variable is used in.

modelReference

The `modelReference` element of data type `SIdRef` is used to reference a Model via a modelReference. The usage depends on the context the Variable is used in.

2.1.7 General attributes

This section describes attributes which occur on multiple SED-ML classes, e.g., `id`, `name`, `math`, `kisaoID`, or `listOf*` constructs.

2.1.7.1 id

Most SED-ML classes have 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., every `id` must be unambiguous throughout a SED-ML document.

An example for an `id` is given in Listing 2.9. In the example the model has the `id` `m00001`.

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

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

2.1.7.2 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.

Listing 2.10 extends the model definition in Listing 2.9 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 2.10: *SED-ML `name` definition, e.g., for a model*

2.1.7.3 math

Some classes have a mandatory element `math` of data type MathML to encode mathematical expressions. Examples are the ComputeChange for pre-processing of Models or DataGenerator for post-processing of Task results. The available subset of mathematical functions and elements which can be used in the `math` element are listed in Section MathML.

2.1.7.4 kisaoID

Some classes, e.g., Algorithm and AlgorithmParameter, have a mandatory element `kisaoID` which references a term from the KiSAO ontology. The referenced term must be defined in the correct syntax, as defined by the regular expression `KISAO:[0-9]{7}`. The referenced KiSAO term should define the simulation Algorithm or AlgorithmParameter as precisely as possible.

2.1.7.5 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. 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 (with exception of `listOfRanges` and `listOfSubTasks` in a RepeatedTask, which are mandatory).

2.1.7.6 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`.

Listing 2.11 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 2.11: *SED-ML `listOfParameters` element*

2.1.7.7 listOfVariables

The Variable class is used 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 or to set up a DataGenerator. The `listOfVariables` is optional and may contain zero to many variables.

Listing 2.12 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 2.12: *SED-ML listOfVariables element*

2.1.8 Reference relations

The reference concept is used to refer to a particular element inside the SED-ML document. It may occur as an association between:

- two Models (modelReference)
- a Variable and a Model (modelReference)
- a Variable and an AbstractTask (taskReference)
- a Task and the simulated Model (modelReference)
- a Task and the Simulation (simulationReference)
- an Output and a DataGenerator (dataReference)

The definition of a Task requires a reference to a particular Model object (modelReference); furthermore, the Task object must be associated with a particular Simulation object (simulationReference).

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

a. The reference association might occur between a Variable object and a Model object, 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.

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.

2.1.8.1 modelReference

The modelReference is a reference used to refer to a particular Model via a `SIdRef`. 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` of a second Model. Circular constructs where a model `A` refers to a model `B` and `B` to `A` (directly or indirectly) are invalid.

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

The `modelReference` attribute of the `variable` element contains the id of a model that is defined in the document.

Listing 2.13 shows the use of the `modelReference` element. In 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`.

```
<model id="m0001" [..]>
    <listOfChanges>
        <computeChange>
```

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

Listing 2.13: *SED-ML* `modelReference` *attribute inside a variable definition of a* `computeChange` *element*

The modelReference is also used to indicate that a Model object is used in a particular Task. Listing 2.14 shows how this can be done for a sample SED-ML document.

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

Listing 2.14: *SED-ML* `modelReference` *definition inside a* `task` *element*

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

2.1.8.2 simulationReference

The simulationReference is used to refer to a particular Simulation via a `SIdRef`, e.g., in a Task.

Listing 2.14 shows the reference to a defined simulation for a sample SED-ML document. In the example, both tasks `t1` and `t2` use the simulation settings defined in `simulation1` to run the experiment.

2.1.8.3 taskReference

The taskReference is a reference used to refer to a particular AbstractTask via a `SIdRef`. The taskReference is used in SubTask to reference the respective subtask, or in Variable within a DataGenerator.

DataGenerator objects are created to apply post-processing to the simulation results before final output. For certain types of post-processing Variable objects need to be created. These link to a task defined within the listOfTasks from which the model that contains the variable of interest can be inferred. A taskReference association is used to realise that link from a Variable object inside a DataGenerator to an AbstractTask object. Listing 2.15 gives an example.

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

Listing 2.15: *SED-ML* `taskReference` *definition inside a* `dataGenerator` *element*

The example shows the definition of a variable `v1` in a `dataGenerator` element. The variable appears in the model that is used in task `t1`. The task definition of `t1` might look as shown in Listing 2.16.

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

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

Task `t1` references the model `model1`. Therefore we can conclude that the variable `v1` defined in Listing 2.15 targets an element of the model with ID `model1`. The targeting process itself will be explained in section 2.1.6 on page 15.

2.1.8.4 dataReference

The dataReference is a reference used to refer to a particular DataGenerator via a `SIdRef`, e.g., from an Output instance.

Four different types of dataReference exist in SED-ML Level 1 Version 3. They are used depending on the type of output for the simulation. A Plot2D has an `xDataReference` and a `yDataReference` assigned. A Plot3D has in addition a `zDataReference` assigned. To define a report, each data column has a `dataReference` assigned.

Listing 2.17 shows the reference to a defined data set for a sample SED-ML document. 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.

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

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

2.2 SED-ML Components

In this section the major components of SED-ML are described. The complete UML class diagram is given in Figure 2.1 on page 10, example simulation experiments are provided in Appendix A, the XML Schema is listed in Appendix B.

2.2.1 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 connected to the SED-ML class via `listOf*` constructs:

- DataDescription (for specification of external data),
- Model (for specification of models),
- Simulation (for specification of simulation setups),
- AbstractTask (for the linkage of models and simulation setups),
- DataGenerator (for the definition of post-processing),
- Output (for the specification of plots and reports).

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 required.

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

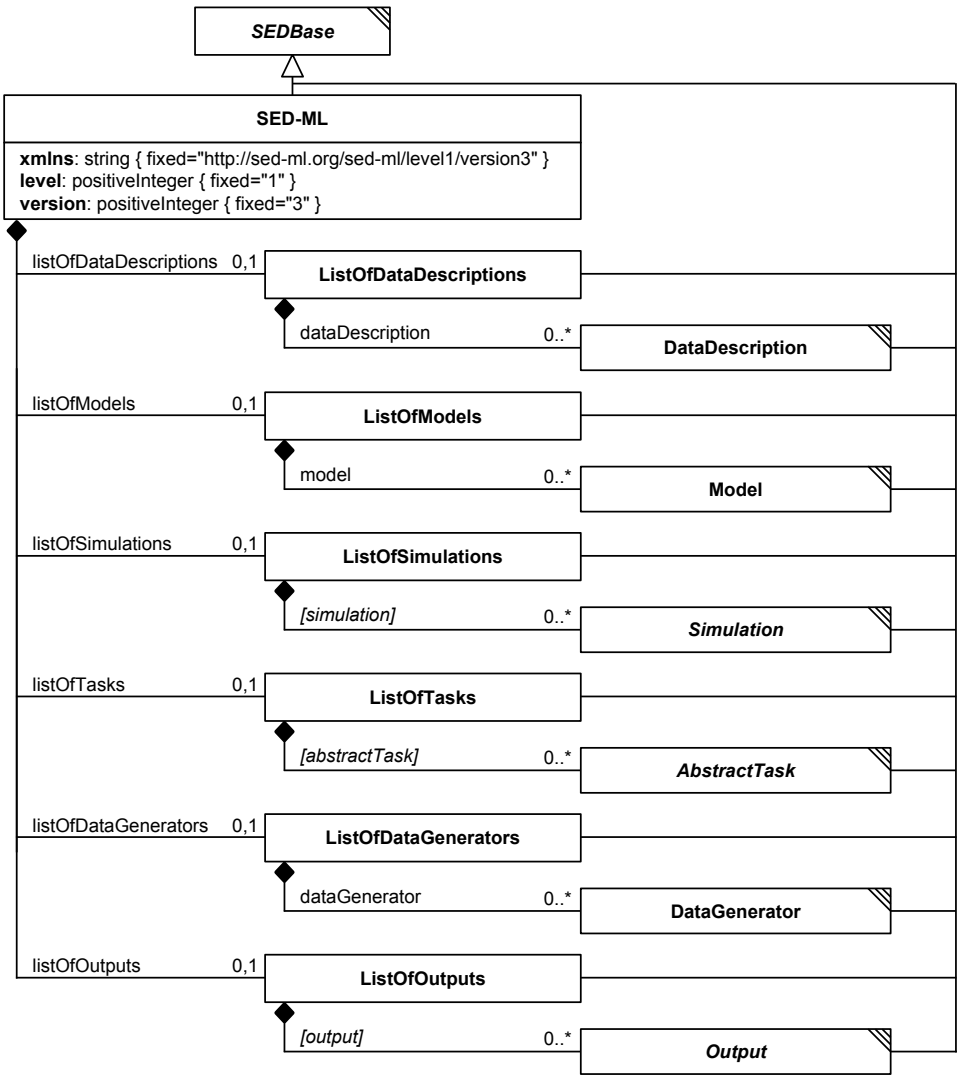

Figure 2.7: *The SED-ML class*

Table 2.5 shows all attributes and sub-elements for the SED-ML element.

attribute	description
metaid^o	page 12
xmlns	page 23
level	page 23
version	page 23
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfDataDescriptions^o	page 23
listOfModels^o	page 24
listOfSimulations^o	page 24
listOfTasks^o	page 24
listOfDataGenerators^o	page 24
listOfOutputs^o	page 25

Table 2.5: *Attributes and nested elements for SED-ML. ^{o}denotes optional elements and attributes.*

The basic XML structure of a SED-ML file is shown in listing 2.18.

```
<?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 2.18: *The SED-ML root element*

2.2.1.1 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. SED-ML notes use the XHTML namespace `http://www.w3.org/1999/xhtml`. Additional external namespaces might be used in annotations.

2.2.1.2 level

The current SED-ML `level` is `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 2.18.

2.2.1.3 version

The current SED-ML `version` is `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 2.18.

2.2.1.4 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 them have to be defined. The SED-ML document uses the `listOfDataDescriptions` container to define DataDescriptions for referencing external data (Figure 2.7 on the previous page). The `listOfDataDescriptions` is optional and may contain zero to many DataDescriptions.

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

```
<listOfDataDescriptions>
    <dataDescription id="Data1" name="Oscli Time Course Data" source="./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 2.19: *SED-ML listOfDataDescriptions element*

2.2.1.5 listOfModels

The models used in a simulation experiment are defined in the `listOfModels` container (Figure 2.7 on page 22). The `listOfModels` is optional and may contain zero to many Models. However, if a SED-ML document contains one or more Tasks, at least one Model must be defined to which the Task elements refer (see Section 2.1.8.1).

Listing 2.20 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 2.20: *SED-ML listOfModels element*

2.2.1.6 listOfSimulations

The `listOfSimulations` element is the container for Simulation descriptions (Figure 2.7 on page 22). The `listOfSimulations` is optional and may contain zero to many Simulations. However, if the SED-ML document contains one or more Tasks, at least one Simulation element must be defined to which the Task elements refer (see Section 2.1.8.2).

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

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

Listing 2.21: *The SED-ML `listOfSimulations` element, containing two simulation setups*

2.2.1.7 listOfTasks

The `listOfTasks` element contains the defined tasks for the simulation experiment (Figure 2.7 on page 22). The `listOfTasks` is optional and may contain zero to many tasks, each of which is an instance of a subclass of AbstractTask.

Listing 2.22 shows the use of the `listOfTasks` element.

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

Listing 2.22: *The SED-ML `listOfTasks` element, defining one task*

2.2.1.8 listOfDataGenerators

The `listOfDataGenerators` container holds the dataGenerator definitions of a simulation experiment (Figure 2.7 on page 22). The `listOfDataGenerators` is optional and in general may contain zero to many DataGenerators.

In SED-ML, all variable and parameter values used in the Output class need to be defined as a DataGenerator beforehand.

Listing 2.23 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 2.23: *The* `listOfDataGenerators` *element, defining two data generators* time *and* LaCI repressor

2.2.1.9 `listOfOutputs`

The `listOfOutputs` container holds the Output definitions of a simulation experiment (Figure 2.7 on page 22). The `listOfOutputs` is optional and may contain zero to many outputs.

Listing 2.24 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 2.24: *The listOfOutput element*

2.2.2 DataDescription

The DataDescription class (Figure 2.8) allows to reference external data, and contains a description on how to access the data, in what format it is, and what subset of data to extract.

The DataDescription class introduces four attributes: the required attributes `id` and `source` and the optional attributes `format` and `name`. In addition two optional elements are defined: `dimensionDescription` and `listOfDataSources`.

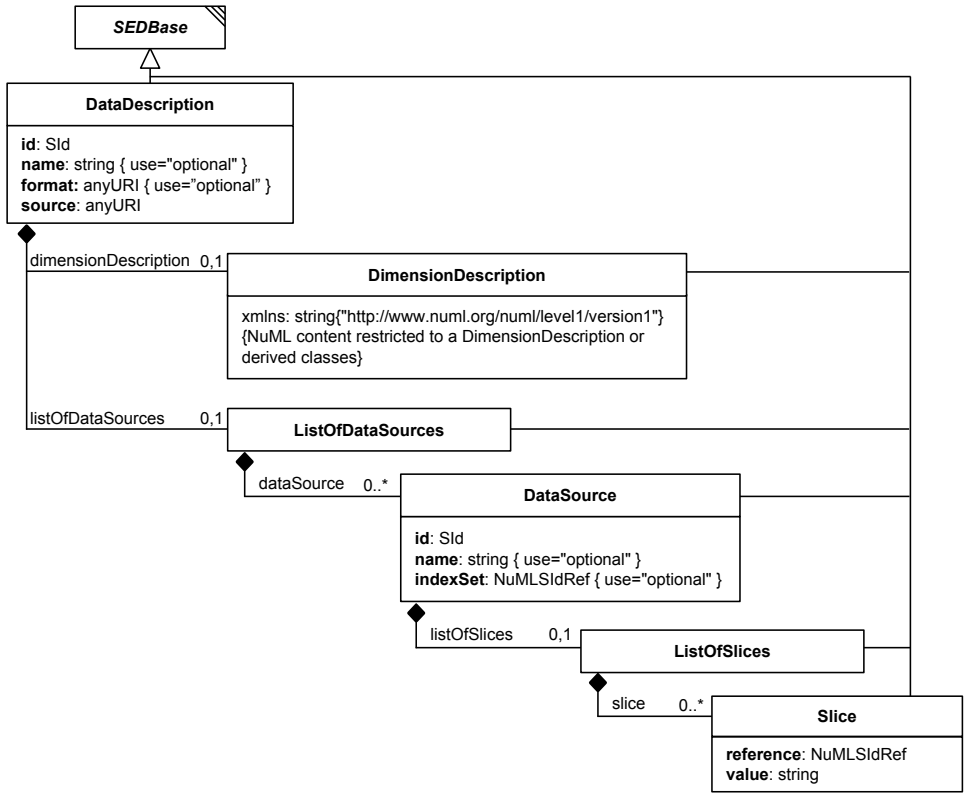

Figure 2.8: *The SED-ML DataDescription class*

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

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
source	page 26
format^o	page 26
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
dimensionDescription^o	page 26
listOfDataSources^o	page 26

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

Listing 2.25 shows the use of the `dataDescription` element.

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

Listing 2.25: *SED-ML `dataDescription` element*

source

The required `source` attribute of data type `anyURI` is used to specify the data file. The `source` attribute provides a location of a data file, analog to how the `source` attribute on the Model is handled. In order to resolve the `source` attribute, the same mechanisms are allowed as for the Model `source` element, i.e., via the local file system, a relative link, or an online resource.

format

The optional `format` attribute of data type `anyURI` is used to specify the format of the DataDescription. The allowed formats are defined in the format references, e.g., NuML (`urn:sedml:format:numl`) or CSV (`urn:sedml:format:csv`). If it is not explicitly defined the default value for `format` is `urn:sedml:format:numl`, referring to NuML representation of the data.

dimensionDescription

The optional `dimensionDescription` contains a DimensionDescription providing the dimension description of the data file. If the format is NuML (`urn:sedml:format:numl`) and a `dimensionDescription` is set, then the `dimensionDescription` must be identical to the `dimensionDescription` of the NuML file. If the format is not NuML, the `dimensionDescription` is required.

listOfDataSources

The optional `listOfDataSources` contains zero or more DataSource elements. A DataSource extracts chunks out of the external data provided by the outer DataDescription element.

2.2.3 DataDescription components

2.2.3.1 DimensionDescription

The DimensionDescription class (Figure 2.8 on the preceding page) defines the dimensions and data types of the external data provided by the outer DataDescription element. The DimensionDescription is a NuML container containing the dimension description of the dataset.

In the following example a nested NuML `compositeDescription` with `time` spanning one dimension and

`SpeciesIds` spanning a second dimension is given. This two dimensional space is then filled with `double` values representing concentrations.

```
<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" id="Concentration" name="Concentration" />
        </compositeDescription>
    </compositeDescription>
</dimensionDescription>
```

Listing 2.26: *SED-ML `dimensionDescription` element*

2.2.3.2 `DataSource`

The DataSource class (Figure 2.8 on page 25) extracts chunks out of the dataset provided by the outer DataDescription element. The DataSource class introduces three attributes: the required attribute `id` and the optional attributes `name`, `indexSet`, and `listOfSlices` (Figure 2.8 on page 25).

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

attribute	**description**
metaid^o	page 12
id	page 17
name^o	page 17
indexSet	page 28
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfSlices^o	page 28

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

DataSource elements can be used anywhere in the SED-ML Description. Specifically their `id` attribute can be referenced within the `listOfVariables` of DataGenerator, ComputeChange or SetValue objects. Here an example that references the DataSource `dataS1`:

```
<listOfDataDescriptions>
  <dataDescription id="data1" name="data file" source="./example.numl" format="urn:sedml:format:numl">
    <dimensionDescription>
      <compositeDescription indexType="double" name="Time">
        <compositeDescription indexType="string" name="SpeciesIds">
          <atomicDescription valueType="double" name="Values" />
        </compositeDescription>
      </compositeDescription>
    </dimensionDescription>
    <listOfDataSources>
      <dataSource id="dataS1">
        <listOfSlices>
          <slice reference="SpeciesIds" value="S1" />
        </listOfSlices>
      </dataSource>
      <dataSource id="dataTime" indexSet="Time" />
    </listOfDataSources>
  </dataDescription>
</listOfDataDescriptions>
<listOfDataGenerators>
  <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>
  ...
</listOfDataGenerators>
```

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 DataGenerator.

To indicate that the target of the Variable is an entity defined within the current SED-ML description (and not an Xpath expression) the hashtag (#) with the reference to an id is used.

In addition, this example uses the modelReference, in order to facilitate a mapping of the data with a given model.

Data may contain NA values. All calculations containing a NA value have NA as a result.

Since data elements defined via the DimensionDescription of the DataDescription or within the NuML file are either values or indices, the DataSource element provides two ways of addressing those elements, the indexSet and listOfSlices.

indexSet

The indexSet attribute allows to address all indices provided by NuML elements with **indexType**.

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

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

Similarly

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

extracts 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.

listOfSlices

The listOfSlices contains one or more Slice elements. The listOfSlices container holds the Slice definitions of a DataSource (Figure 2.8 on page 25). The listOfSlices is optional and may contain zero to many Slices.

2.2.3.3 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 2.8 on page 25).

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

attribute	**description**
metaid^o	page 12
reference	page 28
value	page 28
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

reference

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

value

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

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

isolates 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 can be multiple slice elements present, so it is possible 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>
```

2.2.4 `Model`

The Model class defines the models used in a simulation experiment (Figure 2.9).

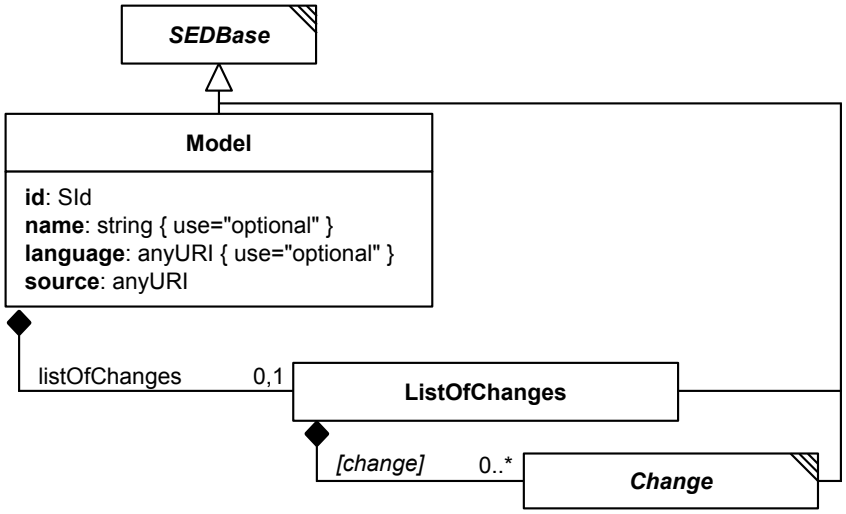

Figure 2.9: *The SED-ML Model class*

Each instance of the Model class has the mandatory attributes `id` and `source`, and the optional attributes `name`, `language`, and `listOfChanges`.

The optional attribute `language` defines the format the model is encoded in.

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 pre-processing before simulation. Those pre-processing steps are specified in the `listOfChanges` via the Change class.

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

Listing 2.27 on the next page shows the use of the `model` element. In the example the `listOfModels` contains three models: The first model `m0001` is the Repressilator model from BioModels Database

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
language^o	page 30
source	page 30
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfChanges^o	page 31

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

available from urn:miriam:biomodels.db:BIOMD0000000012. For the SED-ML simulation the model might undergo preprocessing steps described in the listOfChanges. Based on the description of the first model m0001, the second model m00012 is built, which is a modified version of the Repressilator model. m0002 refers to the model m001 in its source attribute. m0002 might then have additional changes applied to it on top of the changes defined in the pre-processing of m0001. The third model in the code example is a model in CellML representation. The model m0003 is available from the given URL in the source attribute. Again, it might have pre-processing steps applied before used in a simulation.

```
<listOfModels>
    <model id="m0001" language="urn:sedml:language:sbml"
        source="urn:miriam:biomodels.db:BIOMD0000000012">
        <listOfChanges>
            <change>
                [MODEL PRE-PROCESSING]
            </change>
        </listOfChanges>
    </model>
    <model id="m0002" language="urn:sedml:language:sbml" source="m0001">
        <listOfChanges>
            [MODEL PRE-PROCESSING]
        </listOfChange>
    </model>
    <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">
        [MODEL PRE-PROCESSING]
    </model>
</listOfModels>
```

Listing 2.27: *SED-ML* `model` *element*

language

The optional language attribute of data type anyURI is used to specify the format of the model. Example formats are SBML (urn:sedml:language:sbml) or CellML (urn:sedml:language:cellml). The supported languages are defined in the language references.

If it is not explicitly defined the default value for language 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 to decide whether or not one 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 Symbols in the Variable class, as the interpretation of Symbols depends on the language of the representation format.

source

To make a model accessible for the execution of a SED-ML file, the source must be specified through either an URI or a reference to an SId of an existing Model. The URI should follow the proposed URI Scheme for Model references.

An example for the definition of a model via an URI is given in Listing 2.28. The example defines one model m1 with the model source available from urn:miriam:biomodels.db:BIOMD0000000012. The MIRIAM URN can be resolved into the SBML model stored in BioModels Database under the identi-

fier `BIOMD0000000012` using the MIRIAM web service. The resulting URL is https://www.ebi.ac.uk/biomodels-main/BIOMD0000000012.

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

Listing 2.28: *The SED-ML `source` element, using the URI scheme*

An example for the definition of a model using an URL is given in Listing 2.29. 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 is used in the `source` attribute.

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

Listing 2.29: *The SED-ML `source` element, using a URL*

`listOfChanges`

The `listOfChanges` (Figure 2.9 on page 29) contains the Changes to be applied to a particular Model. The `listOfChanges` is optional and may contain zero to many Changes.

Listing 2.30 shows the use of the `listOfChanges` element.

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

Listing 2.30: *The SED-ML `listOfChanges` element, defining a change on a model*

2.2.5 Change

The Change class allows to describe changes applied to a model before simulation (Figure 2.10 on the following page). Changes can be of the following types:

- Changes on attributes of the model's XML representation (ChangeAttribute)
- 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, the ChangeAttribute, AddXML, ChangeXML, RemoveXML, and ComputeChange.

The Change class has the mandatory attribute target which defines the target of the change. The `target` attribute 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.

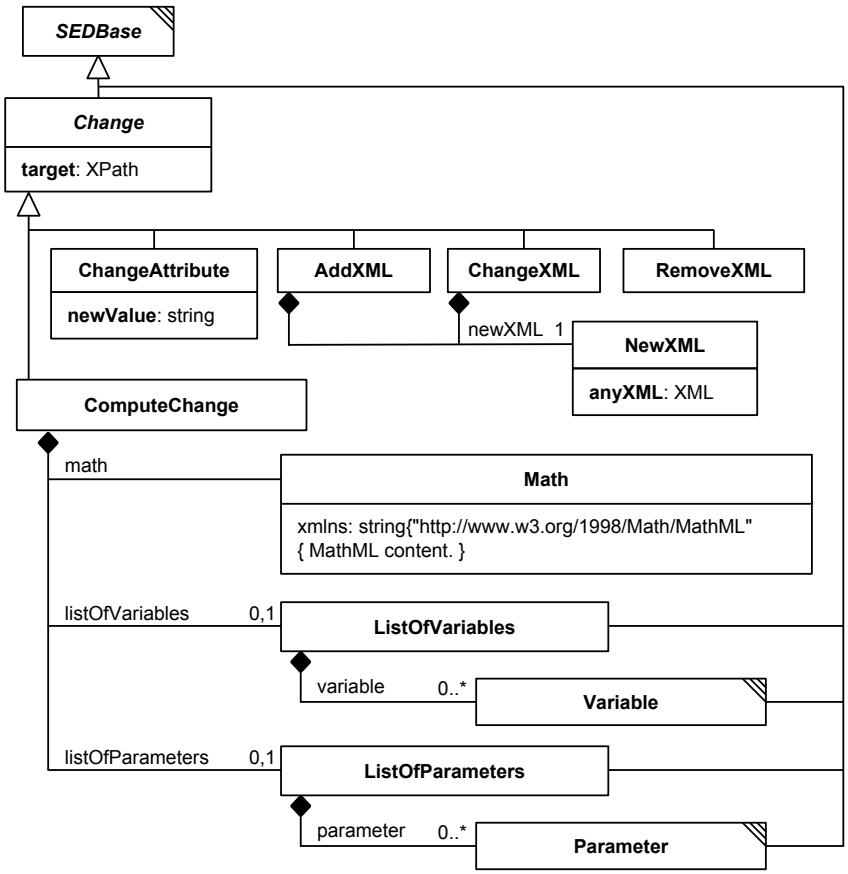

Figure 2.10: *The SED-ML Change class*

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

attribute	description
metaid^o	page 12
name^o	page 17
target	page 32
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

Table 2.10: *Attributes and nested elements for change. ^{o}denotes optional elements and attributes.*

target

The `target` attribute holds a valid XPath expression of data type `xpath` pointing to the XML element or XML attribute that is to undergo the defined changes.

2.2.5.1 NewXML

The `newXML` element provides a piece of XML code (Figure 2.10). `NewXML` must hold a valid piece of XML which after insertion into the original model must result in a valid model file (according to the model language specification as given by the `language` attribute of the model).

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 *is 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 *replaces* the XML element addressed by the XPath.

Examples are given in the relevant change class definitions.

2.2.5.2 AddXML

The AddXML class specifies a snippet of XML that is added as a child of the element selected by the XPath expression in the target attribute (Figure 2.10 on the previous 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 2.31. 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 2.31: *The addXML element with its newXML sub-element*

2.2.5.3 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 2.10 on the preceding page).

The XPath expression is specified in the required target attribute. 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 2.32. 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 2.32: *The changeXML element*

2.2.5.4 RemoveXML

The RemoveXML class can be used to delete XML elements or attributes in the model that are addressed by the XPath expression (Figure 2.10 on the previous page). 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 2.33. 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 2.33: *The removeXML element*

2.2.5.5 ChangeAttribute

The ChangeAttribute class allows to define updates on the XML attribute values of the corresponding model (Figure 2.10 on page 32). ChangeAttribute requires to specify the `target` of the change, i.e., the location of the addressed XML attribute, and also the `newValue` of that attribute. Note that the XPath expression in the `target` attribute must select a single attribute within the corresponding model.

The ChangeXML class covers the possibilities provided by the ChangeAttribute class, i.e, everything that can be expressed by a ChangeAttribute construct can also be expressed by 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.

Table 2.11 shows all attributes and sub-elements for the changeAttribute element.

attribute	**description**
metaid^o	page 12
name^o	page 17
target	page 32
newValue	page 34
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

newValue

The mandatory `newValue` attribute of data type `string` assignes a new value to the targeted XML attribute.

The example in Listing 2.34 shows the update of the value of two parameters inside an SBML model.

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

Listing 2.34: *The `changeAttribute` element and its `newValue` attribute*

2.2.5.6 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 2.10 on page 32).

The mathematical expression for the change is specified using the required `math` attribute of data type `MathML`. If used as an element of the ComputeChange class, it computes the change of the element or attribute addressed by the target attribute.

The computation can use the value of Variables via the optional element `listOfVariables`. Those variables can then be addressed by their respective `id`. A Variable used in a ComputeChange must carry a `modelReference` attribute but no `taskReference` attribute (Figure 2.10 on page 32). If the variable is referring to a DataSource neither the `modelReference` nor `taskReference` is required.

Additional Parameters via the optional element `listOfParameters`. Such Parameters are thereafter referred to by their `id`.

Note that when 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.

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

attribute	**description**
metaid^o	page 12
name^o	page 17
target	page 32
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfVariables^o	page 17
listOfParameters^o	page 17
math	page 17

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

Listing 2.35 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 2.35: *The computeChange element*

The example in Listing 2.35 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 uses 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: $S = S \times \frac{R^n}{K^n + R^n}$

2.2.6 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 2.11).

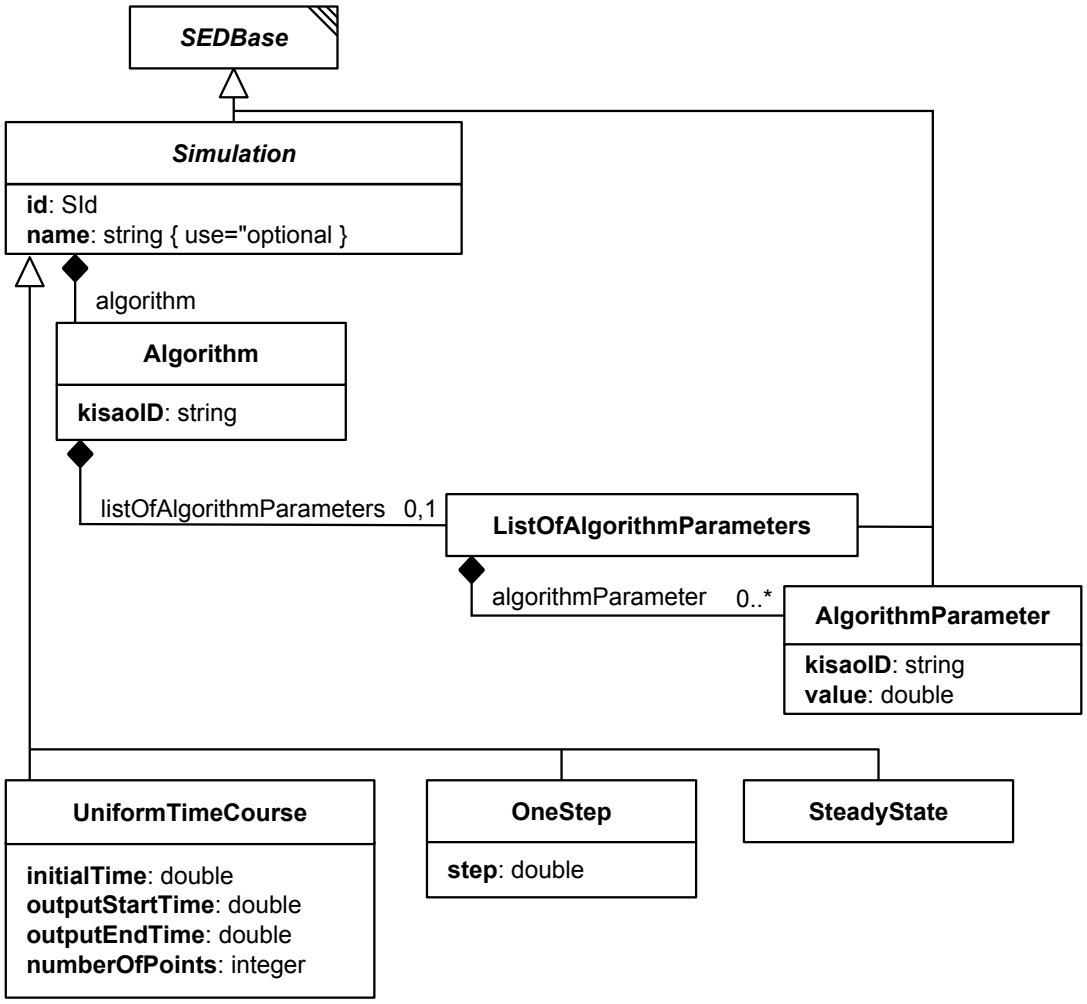

Figure 2.11: *The SED-ML Simulation class*

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.

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` describing the simulation Algorithm.

Table 2.13 on the next page shows all attributes and sub-elements for the simulation element.

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

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

Listing 2.36: *The SED-ML* `listOfSimulations` *element, defining two different UniformTimecourse simulations*

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
algorithm	page 39

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

algorithm

The mandatory attribute `algorithm` defines the simulation algorithms used for the execution of the simulation. The algorithms are defined via the Algorithm class.

2.2.6.1 UniformTimeCourse

Each instance of the UniformTimeCourse class has, in addition to the elements from Simulation, the mandatory elements `initialTime`, `outputStartTime`, `outputEndTime`, and `numberOfPoints` (Figure 2.11 on the previous page).

Table 2.14 shows all attributes and sub-elements for the uniformTimeCourse element.

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
initialTime	page 37
outputStartTime	page 38
outputEndTime	page 38
numberOfPoints	page 38
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
algorithm	page 39

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

Listing 2.37 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 2.37: *The SED-ML `uniformTimeCourse` element, defining a uniform time course simulation over 2500 time units with 1000 simulation points.*

initialTime

The attribute `initialTime` of type `double` represents what the time is at the start of the simulation, for purposes of output variables, and for calculating the `outputStartTime` and `outputEndTime`. In most cases, this will be `0.0`. The model must be set up such that `intialTime` is correct internally with respect to any output variables that may be produced. Listing 2.37 shows an example.

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, and describes the time (relative to the intialTime) that output is to be collected. To be valid the outputStartTime cannot be before initialTime. For an example, see Listing 2.37.

outputEndTime

The attribute outputEndTime of type double marks the end time of the simulation, relative to the initialTime. See Listing 2.37 for an example.

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, 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 2.37.

2.2.6.2 OneStep

The OneStep class 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 2.11 on page 36).

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

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
step	page 38
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
algorithm	page 39

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

Listing 2.38 shows the use of the oneStep element.

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

Listing 2.38: *The SED-ML oneStep element, specifying to apply the simulation algorithm for another output step of size 0.1.*

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 2.38 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.

2.2.6.3 `SteadyState`

The SteadyState represents a steady state computation (as for example implemented by NLEQ or Kinsolve). The SteadyState class has no additional elements than the elements from Simulation (Figure 2.11 on page 36).

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

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
algorithm	page 39

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

Listing 2.39 shows the use of the `steadyState` element.

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

Listing 2.39: *The SED-ML `steadyState` element, defining a steady state simulation with id `steady`.*

2.2.7 `Simulation` components

2.2.7.1 `Algorithm`

The Algorithm class has a mandatory element `kisaoID` which contains a KiSAO reference to the particular simulation algorithm used in the simulation. In addition, the Algorithm has an optional `listOfAlgorithmParameters`, a collection of algorithmParameter, which are used to parameterize the algorithm.

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

attribute	description
metaid^o	page 12
kisaoID	page 61
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfAlgorithmParameters^o	page 40

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

The example given in Listing 2.36, completed by algorithm definitions results in the code given in Listing 2.40. In the example, for both simulations a algorithm is defined. In the first simulation `s1` a deterministic approach is used (Euler forward method), in the second simulation `s2` a stochastic approach

is used (Stochsim nearest neighbor).

```
<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 2.40: *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.

listOfAlgorithmParameters

The `listOfAlgorithmParameters` contains the settings for the simulation algorithm used in a simulation (Figure 2.11 on page 36). It may list several instances of the AlgorithmParameter class. The `listOfAlgorithmParameters` is optional and may contain zero to many parameters.

Listing 2.41 shows the use of the `listOfAlgorithmParameters` element.

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

Listing 2.41: *SED-ML listOfAlgorithmParameters element*

2.2.7.2 *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 2.11 on page 36). 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 2.42: *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.

value

The `value` sets the value of the AlgorithmParameter.

2.2.8 AbstractTask

In SED-ML the subclasses of AbstractTask define which Simulations should be executed with which Models in the simulation experiment. AbstractTask is the base class of all SED-ML tasks, i.e. Task and RepeatedTask.

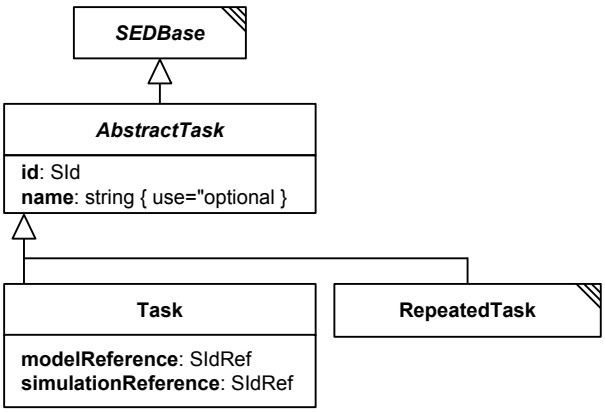

Figure 2.12: *The SED-ML Abstract Task class*

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

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

2.2.8.1 Task

A Task links a Model to a certain Simulation description via their respective identifiers (Figure 2.12), using the `modelReference` and the `simulationReference`. The task class receives the `id` and `name` attributes from AbstractTask.

In SED-ML 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 2.19 shows all attributes and sub-elements for the task element.

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
modelReference	page 18
simulationReference	page 19
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

Listing 2.43 on the next page shows the use of the `task` element. In the example, a simulation setting

`simulation1` is applied first to `model1` and then to `model2`.

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

Listing 2.43: *The `task` element*

2.2.8.2 *Repeated Task*

The RepeatedTask (Figure 2.13) provides a generic looping construct, allowing complex tasks to be composed from individual steps. The RepeatedTask performs a specified task (or sequence of tasks as defined in the `listOfSubTasks`) multiple times (where the exact number is specified through a Range construct as defined in `range`), while allowing specific quantities in the model to be altered at each iteration (as defined in the `listOfChanges`).

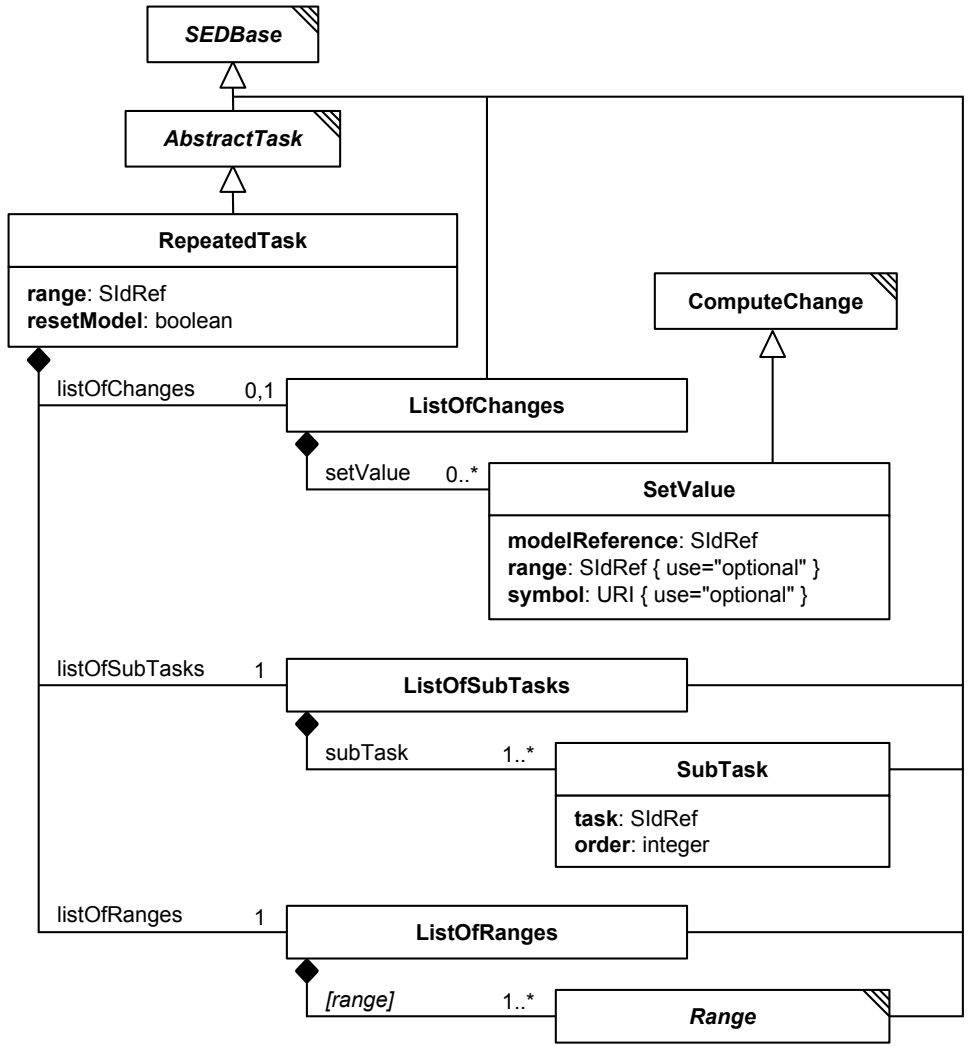

Figure 2.13: *The SED-ML RepeatedTask class*

The RepeatedTask inherits the required attribute `id` and optional attribute `name` from AbstractTask. Additionally it has the two required attributes `range` and `resetModel` and the child elements `listOfRanges`, `listOfChanges` and `listOfSubTasks`. Of these `listOf*` only `listOfChanges` is optional.

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 objects in the `listOfChanges` are applied to the Model.
- Finally, all subTasks in the `listOfSubtasks` are executed in the order specified by their `order` element.

Table 2.20 shows all attributes and sub-elements for the repeatedTask element.

attribute	**description**
metaid^o	page 12
id	page 17
name^o	page 17
range	page 43
resetModel	page 44
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfChanges^o	page 44
listOfSubTask	page 44
listOfRanges	page 44

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

Listing 2.44 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>
  <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 2.44: *The `repeatedTask` element*

range

The RepeatedTask has a required attribute `range` of type `SIdRef`. It specifies which `range` defined in the `listOfRanges` this repeated task iterates over. Listing 2.44 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.

resetModel

The repeatedTask 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 2.44 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.

listOfChanges

The optional `listOfChanges` element contains one or many SetValue elements. These elements allow the modification of values in the model prior to the next iteration of the RepeatedTask.

listOfSubTasks

The required `listOfSubTasks` contains one or more subTasks that specify which Tasks are performed in every iteration of the RepeatedTask. All subTasks 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 order in which to run multiple subTasks must be specified using the `order` attribute on the subTask.

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

Listing 2.45: *The* `subTask` *element. In this example the task* `task2` *must be executed before* `task1`*.*

listOfRanges

The `listOfRanges` defines one or more ranges used in the repeatedTask.

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 RepeatedTask its value is to be replaced with the current value of the Range.

2.2.9 Task components

2.2.9.1 SubTask

A SubTask (Figure 2.13 on page 42) defines the subtask which is executed in every iteration of the enclosing RepeatedTask. The SubTask has a required attribute `task` that references the `id` of another AbstractTask. The order in which to run multiple subTasks must be specified via the required attribute `order`.

task

The required element `task` of data type `SIdRef` specifies the AbstractTask executed by this SubTask.

order

The required attribute `order` of data type `integer` specifies the order in which to run multiple subTasks in the `listOfSubTasks`. To specify that one subTask should be executed before another its `order` attribute must have a lower number (e.g. in Listing 2.45).

2.2.9.2 SetValue

The SetValue class (Figure 2.13 on page 42) allows the modification of the model prior to the next execution of the subTasks. The changes to the model are defined in the `listOfChanges` of the RepeatedTask.

SetValue inherits from the ComputeChange class, which allows it to compute arbitrary expressions involving a number of variables and parameters. SetValue has a mandatory `modelReference` attribute, and the 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 `math` contains the expression computing the value by referring to optional parameters, variables or ranges. Again as for functionalRange, variable references retrieve always the current value of the model variable or range at the current iteration of the enclosing repeatedTask.

```
<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 2.46: *A `setValue` element setting `w` to the values of the range with id `current`.*

2.2.9.3 Range

The Range class is the abstract base class for the different types of ranges, i.e. UniformRange, VectorRange, and FunctionalRange (Figure 2.14).

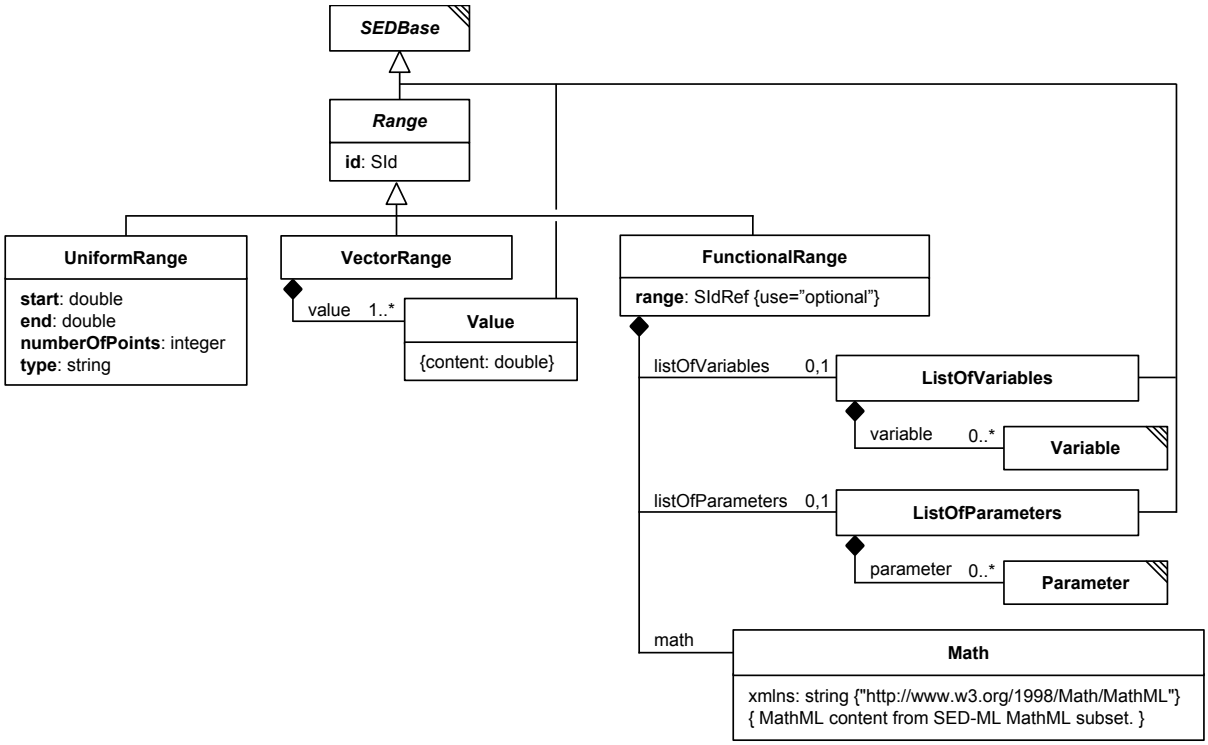

Figure 2.14: *The SED-ML Range class*

2.2.9.3.1 UniformRange

The UniformRange (Figure 2.14) allows the definition of a Range with uniformly spaced values. In this it is quite similar to what is used in the UniformTimeCourse. The UniformRange is defined via three mandatory attributes: `start`, the start value; `end`, the end value and `numberOfPoints` which defines defines the number of points in addition to the start value (the actual items in the range are `numberOfPoints`+1). 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, the following UniformRange will produce `101` values uniformly spaced on the interval $[0, 10]$ in ascending order.

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

Listing 2.47: *The `UniformRange` element*

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 2.48: *The `UniformRange` element with a logarithmic range.*

2.2.9.3.2 `VectorRange`

The VectorRange (Figure 2.14 on the previous page) describes an ordered collection of real values, listing them explicitly within child `value` elements.

For example, the range below iterates over the values 1, 4 and 10 in that order.

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

Listing 2.49: *The `VectorRange` element*

2.2.9.3.3 `Value`

The Value (Figure 2.14 on the preceding page) describes a single value, e.g., the Values in a VectorRange.

2.2.9.3.4 `FunctionalRange`

The FunctionalRange (Figure 2.14 on the previous page) constructs a range through calculations that determine the next value based on the value(s) of other range(s) or model variables. In this it is 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` of type SIdRef 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`, the 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`.

The `math` encompasses the mathematical expression that is used to compute the value for the FunctionalRange 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 2.50: *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 2.51: *A* `functionalRange` *element that returns* `8` *if* `index` *is smaller than* `1`*,* `0.1` *if* `index` *is between* `4` *and* `6`*, and* `8` *otherwise.*

2.2.10 DataGenerator

The DataGenerator class prepares the raw simulation results for later output (Figure 2.15 on the following 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.

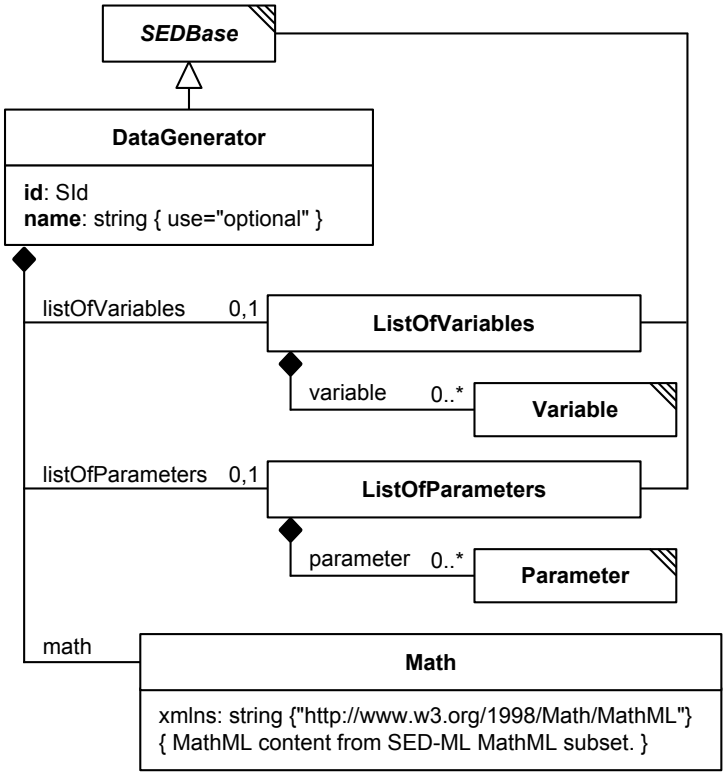

Figure 2.15: *The SED-ML DataGenerator class. Note that* `Parameter` *and* `Variable` *are subclasses of* SEDBase*; the respective inheritance connections are not shown in the figure.*

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 required `math` element contains a mathML expression for the calculation of the DataGenerator. Variable and Parameter instances can be used to encode the mathematical expression.

Table 2.21 shows all attributes and sub-elements for the dataGenerator element.

attribute	**description**
metaid^o	page 12
id	page 17
name^o	page 17
sub-elements	**description**
math	page 17
notes^o	page 13
annotation^o	page 13
listOfVariables^o	page 14
listOfParameters^o	page 14

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

Listing 2.52 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 2.52: *Definition of two* `dataGenerator` *elements,* time *and* LaCI repressor

2.2.11 Output

The abstract Output class describes how the results of a simulation are presented (Figure 2.16 on the next page). The available output classes are plots (Plot2D and Plot3D) and reports (Report). The data used in Outputs is provided via DataGenerators.

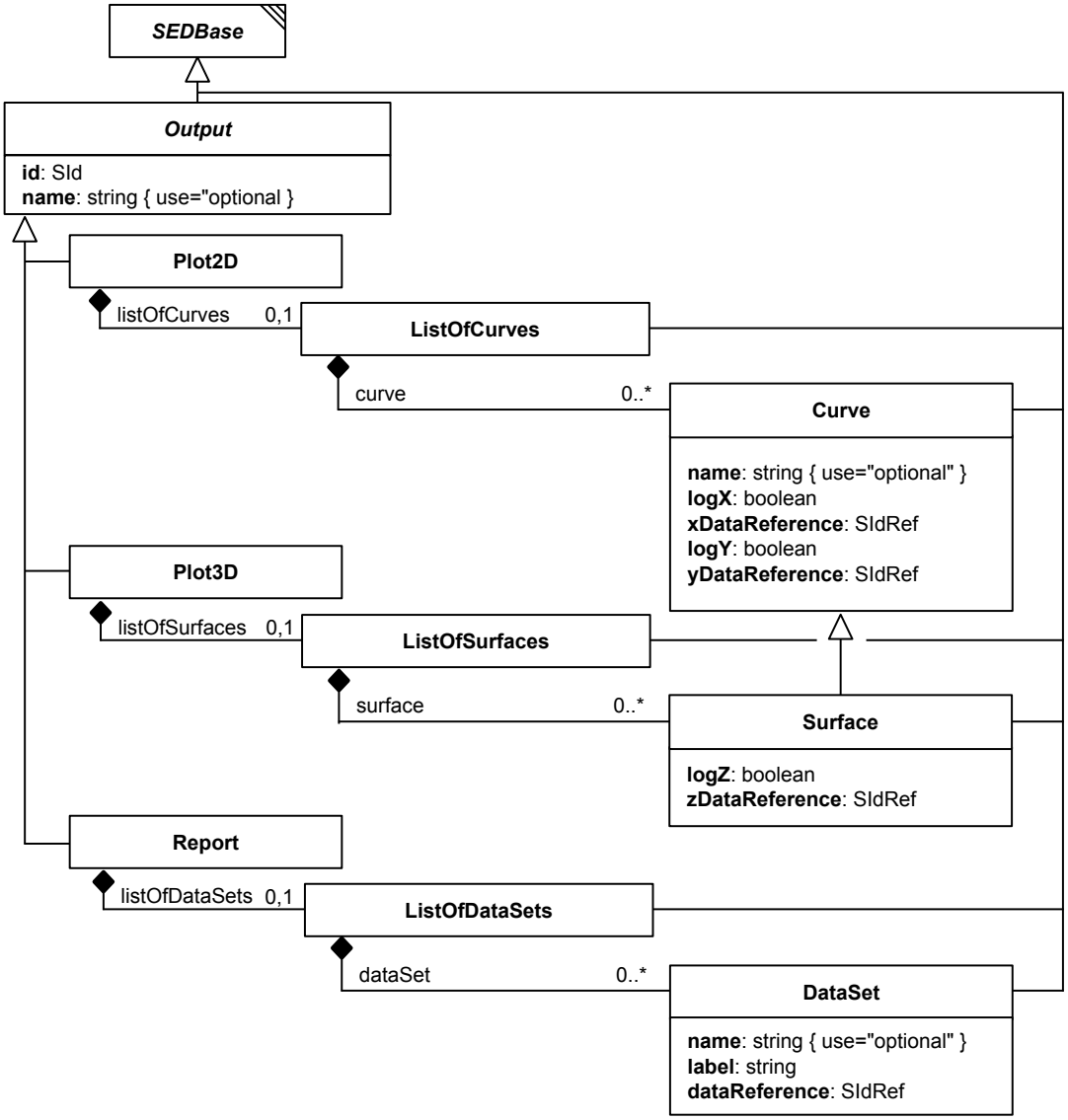

Figure 2.16: *The SED-ML Output class. Note that ListOfCurves, Curve, ListOfSurfaces, Surface, ListOfDataSets, DataSet and DataGenerator are subclasses of* SEDBase*; the respective inheritance connections are not shown in the figure.*

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.

2.2.11.1 `Plot2D`

The Plot2D class is used for two dimensional plot outputs (Figure 2.16). The Plot2D contains a number of Curve definitions in the `listOfCurves`, defining the curves to be plotted in the the 2D plot. Table 2.22 on the next page shows all attributes and sub-elements for the plot2D element.

Listing 2.53 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 2.53: *The `plot2D` element with the nested `listOfCurves` element.*

attribute	description
metaid^o	page 12
name^o	page 17
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfCurves^o	page 52

Table 2.22: *Attributes and nested elements for plot2D. ^{o}denotes optional elements and attributes.*

2.2.11.2 Plot3D

The Plot3D class is used for three dimensional plot outputs (Figure 2.16 on the previous page). The Plot3D contains a number of Surface definitions in the `listOfSurfaces`. Table 2.23 shows all attributes and sub-elements for the plot3D element.

attribute	description
metaid^o	page 12
name^o	page 17
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfSurfaces^o	page 53

Table 2.23: *Attributes and nested elements for plot3D. ^{o}denotes optional elements and attributes.*

Listing 2.54 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 2.54: *The `plot3D` element with the nested `listOfSurfaces` element*

2.2.11.3 Report

The Report class defines a data table consisting of several single instances of the `DataSet` in the `listOfDataSets` (Figure 2.16 on the preceding page). 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 for each column.

Table 2.24 on the next page shows all attributes and sub-elements for the report element.

Listing 2.55 shows the use of the `listOfDataSets` element.

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

Listing 2.55: *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

attribute	description
metaid^o	page 12
name^o	page 17
sub-elements	**description**
notes^o	page 13
annotation^o	page 13
listOfDataSets^o	page 54

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

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.

2.2.12 `Output` components

In this section the Output components Curve, Surface, and DataSet are described.

2.2.12.1 `Curve`

A Curve is a two-dimensional Output component representing a (processed) simulation result (Figure 2.16 on page 50). Zero or more Curve instances define a plot2D (Figure 2.16 on page 50). A curve needs a dataGenerator reference to refer to the data that will be plotted on the x-axis, using the `xDataReference`. A second dataGenerator reference is needed to refer to the data that will be plotted on the y-axis, using the `yDataReference`. Table 2.25 shows all attributes and sub-elements for the curve element.

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
logX	page 52
xDataReference	page 53
logY	page 53
yDataReference	page 53
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

Listing 2.56 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 2.56: *The SED-ML `curve` element, defining the output curve showing the result of simulation for the referenced dataGenerators*

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 2.56.

xDataReference

The xDataReference is a mandatory attribute of the Curve object. Its content refers to a dataGenerator which denotes the DataGenerator object that is used to generate the output on the x-axis of a Curve in a plot2D. The xDataReference data type is SIdRef. However, the valid values for the xDataReference are restricted to the id of already defined DataGenerators.

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 2.56.

yDataReference

The yDataReference is a mandatory attribute of the Curve object. Its content refers to a dataGenerator which denotes the DataGenerator object that is used to generate the output on the y-axis of a Curve in a plot2D. The yDataReference data type is SIdRef. However, the valid values for the yDataReference are restricted to the id of already defined DataGenerators.

2.2.12.2 Surface

A Curve is a three-dimensional Output component representing a (processed) simulation result (Figure 2.16 on page 50). Zero or more Surface instances define a plot3D (Figure 2.16 on page 50).

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 2.26 shows all attributes and sub-elements for the surface element.

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
logX	page 52
xDataReference	page 53
logY	page 53
yDataReference	page 53
logZ	page 54
zDataReference	page 54
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

Listing 2.57 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" />
    [FURTHER SURFACE DEFINITIONS]
```

```
</listOfSurfaces>
```

Listing 2.57: *The SED-ML* `surface` *element, defining the output showing the result of the referenced task*

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 2.57.

zDataReference

The `zDataReference` is a mandatory attribute of the Surface object. Its content refers to a dataGenerator which denotes the DataGenerator object that is used to generate the output on the z-axis of a plot3D. The `zDataReference` data type is `SIdRef`. However, the valid values for the `zDataReference` are restricted to the `id` of already defined DataGenerators.

2.2.12.3 DataSet

The DataSet class holds definitions of data to be used in the Report class (Figure 2.16 on page 50). DataSets are labeled references to instances of the DataGenerator class.

Table 2.27 shows all attributes and sub-elements for the dataSet element.

attribute	description
metaid^o	page 12
id	page 17
name^o	page 17
dataReference	page 54
label	page 54
sub-elements	**description**
notes^o	page 13
annotation^o	page 13

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

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.

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 dataGenerator fills the according dataSet in the report.

Listing 2.58 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 2.58: *The SED-ML* `dataSet` *element, defining a data set containing the result of the referenced task*

3. Concepts used in SED-ML

3.1 MathML

SED-ML encodes mathematical expressions using a subset of MathML 2.0 [4]. 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.

SED-ML files can use mathematical expressions to encode for example pre-processing steps applied to the computational model (ComputeChange), or post processing steps applied to the raw simulation data before output (DataGenerator).

SED-ML classes reference MathML expressions via the element `math` of data type `MathML`.

3.1.1 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`

3.1.2 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. 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.

3.1.3 MathML functions

The only aggregate MathML functions available in SED-ML are `min`, `max`, `sum`, and `product`. These represent the only exceptions. At this point SED-ML does not define a complete algebra of vector values.

min

The `min` of a variable represents the smallest value the simulation experiment for that variable (Listing 3.1).

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

Listing 3.1: *Example for the use of the MathML* `min` *function.*

max

The `max` of a variable represents the largest value the simulation experiment for that variable (Listing 3.2).

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

Listing 3.2: *Example for the use of the MathML* `max` *function.*

sum

The `sum` of a variable represents the sum of all values of the variable returned by the simulation experiment (Listing 3.3).

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

Listing 3.3: *Example for the use of the MathML* `sum` *function.*

product

The `product` of a variable represents the multiplication of all values of the variable returned by the simulation experiment (Listing 3.4).

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

Listing 3.4: *Example for the use of the MathML* `product` *function.*

3.1.4 NA values

NA (not available) values can occur within a simulation experiment. Examples are missing values in a DataSource or simulation results with NA values. All math operations encoded in MathML in SED-ML are well defined on NA values.

NA values in a Curve or Surface should be ignored during plotting.

3.2 URI scheme

URIs are used in SED-ML as a mechanism

- to reference models (3.2.1 Model references)
- to reference data files (3.2.2 Data references)
- to specify the language of the referenced model (3.2.3 Language references)

- to specify the format of the referenced dataset (3.2.4 Data format references)
- to enable addressing implicit model variables (3.2.5 Symbols)

In addition, annotation of SED-ML elements should use a standardised URI Annotations Scheme to ensure long-time availability of information that can unambiguously be identified.

3.2.1 Model references

The URI of a model should preferably point to a public, consistent location that provides the model description file. References to curated, open model bases are recommended, such as the BioModels Database. However, any resource registered with MIRIAM resources[1] can easily be referenced.

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. 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[2]. 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 does not specify how to resolve the URNs. However, MIRIAM Resources offers web services to do so[3]. 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`.

For additional information see the `source` attribute on Model.

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, the recommended means is the COMBINE archive.

3.2.2 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.

For additional information see the `source` attribute on DataDescription.

An alternative means to obtain a data file may be to provide a single resource containing necessary data files and the SED-ML file is the COMBINE archive.

3.2.3 Language references

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.

To specify the language a model is encoded in, a set of pre-defined SED-ML URNs can be used (Table 3.1 on the following page). 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 spe-

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

cific as defining a model being in a particular version of a language, e.g., SBML Level 3 Version 1 as `urn:sedml:language:sbml.level-3.version-1`.

For additional information see the `language` attribute on Model.

Language	URN
CellML (generic)	`urn:sedml:language:cellml`
CellML 1.0	`urn:sedml:language:cellml.1_0`
CellML 1.1	`urn:sedml:language:cellml.1_1`
NeuroML (generic)	`urn:sedml:language:neuroml`
NeuroML Version 1.8.1 Level 1	`urn:sedml:language:neuroml.version-1_8_1.level-1`
NeuroML Version 1.8.1 Level 2	`urn:sedml:language:neuroml.version-1_8_1.level-2`
SBML (generic)	`urn:sedml:language:sbml`
SBML Level 1 Version 1	`urn:sedml:language:sbml.level-1.version-1`
SBML Level 1 Version 2	`urn:sedml:language:sbml.level-1.version-2`
SBML Level 2 Version 1	`urn:sedml:language:sbml.level-2.version-1`
SBML Level 2 Version 2	`urn:sedml:language:sbml.level-2.version-2`
SBML Level 2 Version 3	`urn:sedml:language:sbml.level-2.version-3`
SBML Level 2 Version 4	`urn:sedml:language:sbml.level-2.version-4`
SBML Level 3 Version 1	`urn:sedml:language:sbml.level-3.version-1`
SBML Level 3 Version 2	`urn:sedml:language:sbml.level-3.version-2`
VCML (generic)	`urn:sedml:language:vcml`

Table 3.1: *Predefined model language URNs. The latest list of language URNs is available from http://sed-ml.org/.*

3.2.4 Data format references

To help software decide whether or not it supports a SED-ML file, the information on the dataDescription encoding for each referenced dataDescription can be provided through the `format` attribute.

To specify the format of a dataDescription, a set of pre-defined SED-ML URNs can be used (Table 3.2). The structure of SED-ML format URNs is `urn:sedml:format:`*`name.version`*.

If it is not explicitly defined the default value for `format` is `urn:sedml:format:numl`, referring to NuML representation of the data. However, the use of the `format` attribute is strongly encouraged.

For additional information see the `format` attribute on DataDescription and the description of individual formats and their use in SED-ML below.

Data Format	URN
NuML (generic)	`urn:sedml:format:numl`
NuML Level 1 Version 1	`urn:sedml:format:numl.level-1.version-1`
CSV	`urn:sedml:format:csv`
TSV	`urn:sedml:format:tsv`

Table 3.2: *Predefined dataDescription format URNs. The latest list of format URNs is available from http://sed-ml.org/.*

3.2.4.1 NuML (Numerical Markup Language)

NuML is an exchange format for numerical data. Data in the NuML format (`urn:sedml:format:numl`) is defined via `resultComponents` with a single dataset corresponding to a single `resultComponent`. In the case that a NuML file consists of multiple `resultComponents` the first `resultComponent` contains the data used in the DataDescription. There is currently no mechanism in SED-ML to reference the additional `resultComponents`.

If a `dimensionDescription` is set on the DataDescription, than this `dimensionDescription` must be

identical to the dimensionDescription of the NuML file.

3.2.4.2 CSV (Comma Separated Values)

Data in the CSV format (urn:sedml:format:csv) must follow the following rules when used in combination with SED-ML:

- Each record is one line - Line separator may be LF (0x0A) or CRLF (0x0D0A), a line separator may also be embedded in the data (making a record more than one line but still acceptable).
- Fields are separated with commas.
- Embedded commas - Field must be delimited with double-quotes.
- Leading and trailing whitespace is ignored - Unless the field is delimited with double-quotes in that case the whitespace is preserved.
- Embedded double-quotes - Embedded double-quote characters must be doubled, and the field must be delimited with double-quotes.
- Embedded line-breaks - Fields must be surounded by double-quotes.
- Always Delimiting - Fields may always be delimited with double quotes, the delimiters will be parsed and discarded by the reading applications.
- The first record is the header record defining the unique column ids
- Lines starting with "#" are treated as comment lines and ignored
- Empty lines are allowed and ignored
- For numerical data the "." decimal separator is used
- The following strings are interpreted as NaN: "", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN", "N/A", "NA", "NULL", "NaN", "nan".

A dataset in CSV is always encoding two dimensional data.

When using data in the CSV format SED-ML, the dimensionDescription is required on the DataDescription.

The dimensionDescription must consist of an outer compositeDescription with indexType="integer" which allows to reference the rows of the CSV by index and a inner compositeDescription which allows to reference the columns of the CSV by their column header id. Within the inner compositeDescription exactly one atomicDescription must exist. All data in the CSV must have the same type which is defined via the valueType on the atomicDescription.

Below an example of the required dimensionDescription for a CSV is provided. In the example the time and S1 columns are read from the CSV file

```
# ./example.csv
time, S1, S2
0.0, 10.0, 0.0
0.1, 9.9, 0.1
0.2, 9.8, 0.2
```

Listing 3.5: *Example CSV*

```

<dataDescription id="datacsv" name="Example CSV dataset" source="./example.csv" format="
    urn:sedml:format:csv">
  <dimensionDescription>
    <compositeDescription indexType="integer" name="Index">
      <compositeDescription indexType="string" name="ColumnIds">
        <atomicDescription valueType="double" name="Values" />
      </compositeDescription>
    </compositeDescription>
  </dimensionDescription>
  <listOfDataSources>
    <dataSource id="dataTime">
      <listOfSlices>
        <slice reference="ColumnIds" value="time" />
      </listOfSlices>
    </dataSource>
```

```
    <dataSource id="dataS1">
      <listOfSlices>
        <slice reference="ColumnIds" value="S1" />
      </listOfSlices>
    </dataSource>
  </listOfDataSources>
  ...
</dataDescription>
```

Listing 3.6: *SED-ML* `dimensionDescription` *element for the example.csv*

3.2.4.3 TSV (Tab Separated Values)

The format TSV (`urn:sedml:format:tsv`) is defined identical to CSV with the exceptions listed below

- Fields are separated with tabs instead of commas.
- Embedded tab - Field must be delimited with double-quotes (embedded comma field must not be delimited with double quotes)

3.2.5 Symbols

Some variables used in a simulation 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, whereas time is not explicitly defined.

SED-ML can refer to such implicit variables via the Symbol concept. Such implicit variables are defined using the SED-ML URN scheme `urn:sedml:symbol:`*`implicitVariable`*.

For example, to refer in a SED-ML file to the definition of time, the URN `urn:sedml:symbol:time` is used.

Table 3.3 lists the predefined symbols in SED-ML.

Language	URN	Definition
SBML	`urn:sedml:symbol:time`	Time in SBML is an intrinsic model variable that is addressable in model equations via a csymbol `time`.

Table 3.3: *Predefined symbols in SED-ML. The latest list of symbols is available from* `http://sed-ml.org`.

3.2.6 Annotation Scheme

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 standardized biomodels.net qualifier. The list of qualifiers, as well as further information about their usage, is available from `http://www.biomodels.net/qualifiers/`.

3.3 XPath

XPath is a language for finding and referencing 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 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).

3.4 NuML

The Numerical Markup Language (NuML) aims to standardize the exchange and archiving of numerical results. Additional information including the NuML specification is available from `https://github.com/NuML/NuML`.

NuML constructs are used in SED-ML for referencing external data sets in the DataDescription class. NuML is used to define the DimensionDescription of external datasets in the DataDescription. In addition, `NuMLSIds` are used for retrieving subsets of data via either the `indexSet` element in the DataSource or within the Slice class.

3.5 KiSAO

The Kinetic Simulation Algorithm Ontology (KiSAO [7]) is used in SED-ML to specify simulation algorithms and algorithmParameters. 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/`.

3.6 COMBINE archive

A COMBINE archive [1] 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 [1].

COMBINE archives are the recommended means for distributing simulation experiment descriptions in SED-ML, the respective data and model files, and the Outputs of the simulation experiment (figures and

reports). All SED-ML specification examples in Appendix A are available as COMBINE archive from http://sed-ml.org.

3.7 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 https://github.com/sed-ml.

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. 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).

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 (see Section 3.2.3 for more information).

All specification examples listed below are available as Combine Archives from `http://sed-ml.org/` under the `*.omex` file name for the respective example.

Additional SED-ML examples are available at `http://sed-ml.org/`.

A.1 Example simulation experiment (`L1V3_repressilator.omex`)

This example lists the SED-ML for the example in the introduction (Section 1.2).

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created by phraSED-ML version v1.0.7 with libSBML version 5.15.0. -->
<sedML xmlns="http://sed-ml.org/sed-ml/level1/version3" level="1" version="3">
  <listOfSimulations>
    <uniformTimeCourse id="sim1" initialTime="0" outputStartTime="0" outputEndTime="1000" numberOfPoints=
        "1000">
      <algorithm kisaoID="KISAO:0000019"/>
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml.level-3.version-1" source="urn:miriam:biomodels.
        db:BIOMD0000000012"/>
    <model id="model2" language="urn:sedml:language:sbml.level-3.version-1" source="model1">
      <listOfChanges>
        <changeAttribute target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='ps_0']/
            @value" newValue="1.3e-05"/>
        <changeAttribute target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='ps_a']/
            @value" newValue="0.013"/>
      </listOfChanges>
    </model>
  </listOfModels>
  <listOfTasks>
    <task id="task1" modelReference="model1" simulationReference="sim1"/>
    <task id="task2" modelReference="model2" simulationReference="sim1"/>
  </listOfTasks>
  <listOfDataGenerators>
    <!-- timecourse -->
    <dataGenerator id="dg_0_0_0" name="task1.time">
      <listOfVariables>
        <variable id="task1_____time" symbol="urn:sedml:symbol:time" taskReference="task1"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task1_____time </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_0_0_1" name="PX (lacI)">
      <listOfVariables>
        <variable id="task1_____PX" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PX
            ']" taskReference="task1" modelReference="model1"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task1_____PX </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_0_1_1" name="PZ (cI)">
      <listOfVariables>
        <variable id="task1_____PZ" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PZ
            ']" taskReference="task1" modelReference="model1"/>
```

```
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task1_____PZ </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_0_2_1" name="PY (tetR)">
      <listOfVariables>
        <variable id="task1_____PY" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PY
            ']" taskReference="task1" modelReference="model1"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task1_____PY </ci>
      </math>
    </dataGenerator>
    <!-- pre-processing -->
    <dataGenerator id="dg_1_0_0" name="time">
      <listOfVariables>
        <variable id="task2_____time" symbol="urn:sedml:symbol:time" taskReference="task2"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task2_____time </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_1_0_1" name="PX (lacI)">
      <listOfVariables>
        <variable id="task2_____PX" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PX
            ']" taskReference="task2" modelReference="model2"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task2_____PX </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_1_1_1" name="PZ (cI)">
      <listOfVariables>
        <variable id="task2_____PZ" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PZ
            ']" taskReference="task2" modelReference="model2"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task2_____PZ </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_1_2_1" name="PY (tetR)">
      <listOfVariables>
        <variable id="task2_____PY" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PY
            ']" taskReference="task2" modelReference="model2"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> task2_____PY </ci>
      </math>
    </dataGenerator>
    <!-- post-processing -->
    <dataGenerator id="dg_2_0_0" name="PX/max(PX) (lacI normalized)">
      <listOfVariables>
        <variable id="task1_____PX" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PX
            ']" taskReference="task1" modelReference="model1"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <divide/>
          <ci> task1_____PX </ci>
          <apply>
            <csymbol definitionURL="http://sed-ml.org/#max" encoding="text">max</csymbol>
            <ci> task1_____PX </ci>
          </apply>
        </apply>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_2_0_1" name="PZ/max(PZ) (cI normalized)">
      <listOfVariables>
        <variable id="task1_____PZ" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PZ
            ']" taskReference="task1" modelReference="model1"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <divide/>
          <ci> task1_____PZ </ci>
          <apply>
            <csymbol definitionURL="http://sed-ml.org/#max" encoding="text">max</csymbol>
            <ci> task1_____PZ </ci>
          </apply>
        </apply>
      </math>
    </dataGenerator>
    <dataGenerator id="dg_2_1_0" name="PY/max(PY) (tetR normalized)">
      <listOfVariables>
        <variable id="task1_____PY" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='PY
            ']" taskReference="task1" modelReference="model1"/>
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
```

```
        <apply>
          <divide/>
          <ci> task1_____PY </ci>
          <apply>
            <csymbol definitionURL="http://sed-ml.org/#max" encoding="text">max</csymbol>
            <ci> task1_____PY </ci>
          </apply>
        </apply>
      </math>
    </dataGenerator>
  </listOfDataGenerators>
  <listOfOutputs>
    <plot2D id="timecourse" name="Timecourse of repressilator">
      <listOfCurves>
        <curve id="plot_0__plot_0_0_0__plot_0_0_1" logX="false" logY="false" xDataReference="dg_0_0_0"
             yDataReference="dg_0_0_1"/>
        <curve id="plot_0__plot_0_0_0__plot_0_1_1" logX="false" logY="false" xDataReference="dg_0_0_0"
             yDataReference="dg_0_1_1"/>
        <curve id="plot_0__plot_0_0_0__plot_0_2_1" logX="false" logY="false" xDataReference="dg_0_0_0"
             yDataReference="dg_0_2_1"/>
      </listOfCurves>
    </plot2D>
    <plot2D id="preprocessing" name="Timecourse after pre-processing">
      <listOfCurves>
        <curve id="plot_1__plot_1_0_0__plot_1_0_1" logX="false" logY="false" xDataReference="dg_1_0_0"
             yDataReference="dg_1_0_1"/>
        <curve id="plot_1__plot_1_0_0__plot_1_1_1" logX="false" logY="false" xDataReference="dg_1_0_0"
             yDataReference="dg_1_1_1"/>
        <curve id="plot_1__plot_1_0_0__plot_1_2_1" logX="false" logY="false" xDataReference="dg_1_0_0"
             yDataReference="dg_1_2_1"/>
      </listOfCurves>
    </plot2D>
    <plot2D id="postprocessing" name="Timecourse after post-processing">
      <listOfCurves>
        <curve id="plot_2__plot_2_0_0__plot_2_0_1" logX="false" logY="false" xDataReference="dg_2_0_0"
             yDataReference="dg_2_0_1"/>
        <curve id="plot_2__plot_2_1_0__plot_2_0_0" logX="false" logY="false" xDataReference="dg_2_1_0"
             yDataReference="dg_2_0_0"/>
        <curve id="plot_2__plot_2_0_1__plot_2_1_0" logX="false" logY="false" xDataReference="dg_2_0_1"
             yDataReference="dg_2_1_0"/>
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing A.1: *SED-ML document for example simulation experiment.*

A.2 Simulation experiments with dataDescriptions

The DataDescription provides means to use external datasets in simulation experiments. In this section simulation experiments using the dataDescription are presented.

A.2.1 Plotting data with simulations (L1V3_plotting-data-numl.omex)

This example demonstrates the use of the DataDescription and DataSource to load external data in SED-ML. In the example a model is simulated (using a uniformTimeCourse simulation) and the simulation results are plotted. In addition data is plotted using the dataDescription and DataSource), extracting the `S1` and `time` column from it and renders it. The listed example uses data encoded in NuML as format (`urn:sedml:format:numl`).

The corresponding example using CSV (`urn:sedml:format:csv`) as format to encode the data is available as `L1V3_plotting-data-csv.omex`.

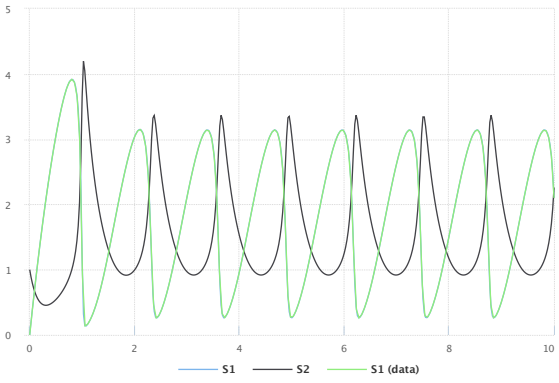

Figure A.1: *The simulation result from the simulation description given in Listing A.2. Simulation with SED-ML web tools [2].*

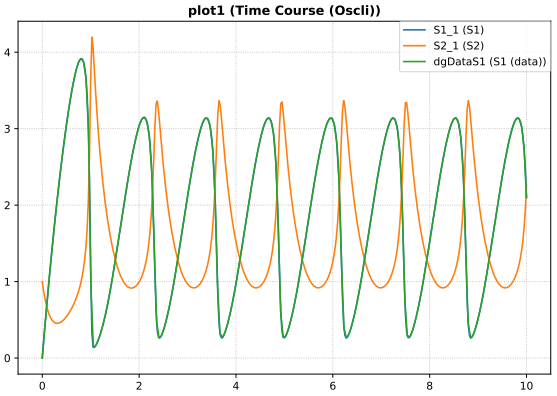

Figure A.2: *Simulation with tellurium [5].*

```
<?xml version="1.0" encoding="utf-8"?>
<sedML level="1" version="3" xmlns="http://sed-ml.org/sed-ml/level1/version3">
    <listOfDataDescriptions>
        <dataDescription id="Data1" name="oscillator data" source="./oscli.numl" format="
             urn:sedml:format: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="400">
            <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="./oscli.xml"/>
    </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">
                <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
                    "/>
                <curve id="curve3" logX="false" logY="false" xDataReference="dgDataTime" yDataReference="
                    dgDataS1"/>
            </listOfCurves>
        </plot2D>
    </listOfOutputs>
</sedML>
```

Listing A.2: *SED-ML document using DataSource and DataDescription*

A.3 Simulation experiments with repeatedTasks

The RepeatedTask makes it possible to encode a large number of different simulation experiments. In this section several such simulation experiments are presented.

A.3.1 Time course parameter scan (`L1V3_repeated-scan-oscli.omex`)

In this example a repeatedTask is used to run repeated uniformTimeCourse simulations with a deterministic simulation algorithm. Within the repeatedTask after each run the parameter value is changed, resulting in a time course parameter scan.

NOTE: This example produces three dimensional results (time, species concentration, multiple repeats). SED-ML Level 1 Version 3 does not include a way to post-process 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.

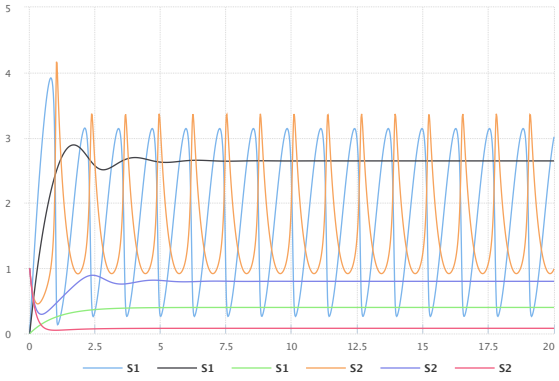

Figure A.3: *The simulation result gained from the simulation description given in Listing A.3. Simulation with SED-ML web tools [2].*

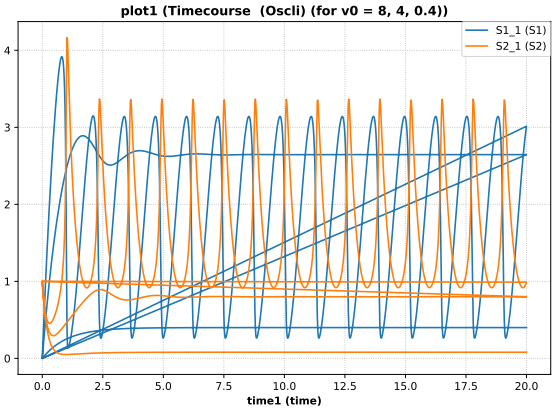

Figure A.4: *Simulation with tellurium [5].*

```
<?xml version="1.0" encoding="utf-8"?>
<sedML 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="./oscli.xml" />
  </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" symbol="urn:sedml:symbol:time" taskReference="task1" />
      </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 A.3: *SED-ML document implementing the one dimensional time course parameter scan*

A.3.2 Steady state parameter scan (`L1V3_repeated-steady-scan-oscli.omex`)

In this example a repeatedTask is used in combination with a steadyState simulation task (performing a steady state computation). On each repeat a parameter is varied resulting in a steady state parameter scan.

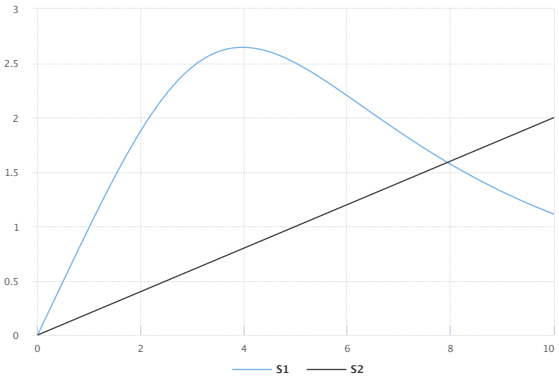

Figure A.5: *The simulation result from the simulation description given in Listing A.4. Simulation with SED-ML web tools [2].*

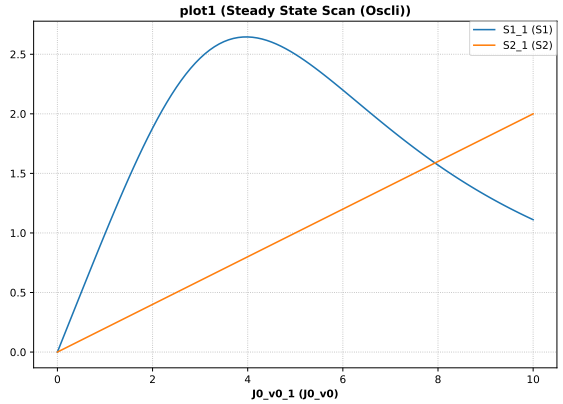

Figure A.6: *Simulation with tellurium [5].*

```
<?xml version="1.0" encoding="utf-8"?>
```

```xml
<!-- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
<sedML 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="./oscli.xml" />
  </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 A.4: *SED-ML document implementing the one dimensional steady state parameter scan*

A.3.3 Stochastic simulation (L1V3_repeated-stochastic-runs.omex)

In this example a repeatedTask is used to run a stochastic simulation multiple times. Running just one stochastic trace does not provide a complete picture of the behavior of a system. A large number of such traces is needed. This example demonstrates the basic use case of running ten traces of a simulation by using a repeatedTask which runs ten uniform time course simulations (each performing a stochastic simulation run).

NOTE: This example produces three dimensional results (time, species concentration, multiple repeats). While SED-ML Level 1 Version 3 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.

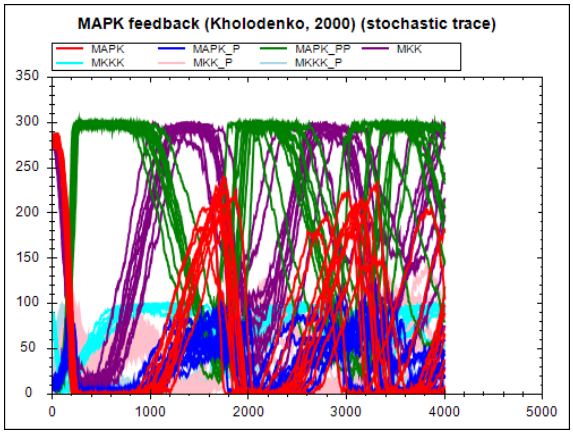

Figure A.7: *The simulation result from the simulation description given in Listing A.5. Simulation with SED-ML web tools [2].*

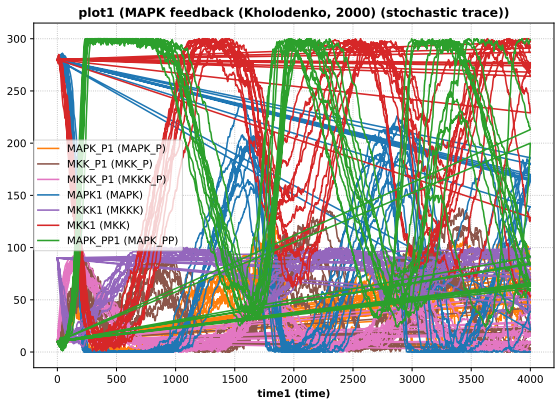

Figure A.8: *Simulation with tellurium [5].*

```
<?xml version="1.0" encoding="utf-8"?>
<sedML 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="./BorisEJB.xml" />
  </listOfModels>
  <listOfTasks>
    <task id="task0" modelReference="model1" simulationReference="timecourse1" />
    <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 A.5: *SED-ML document implementing repeated stochastic runs*

A.3.4 Simulation perturbation (`L1V3_oscli-nested-pulse.omex`)

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 a repeatedTask is used iterating a oneStep task (that advances an ODE integration to the next output step). During the steps a single parameter is modified effectively causing the oscillations of a model to stop. Once the value is reset the oscillations recover.

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

```
<?xml version="1.0" encoding="utf-8"?>
<sedML 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="./oscli.xml" />
  </listOfModels>
  <listOfTasks>
    <task id="task0" modelReference="model1" simulationReference="stepper" />
    <repeatedTask id="task1" resetModel="false" range="index">
```

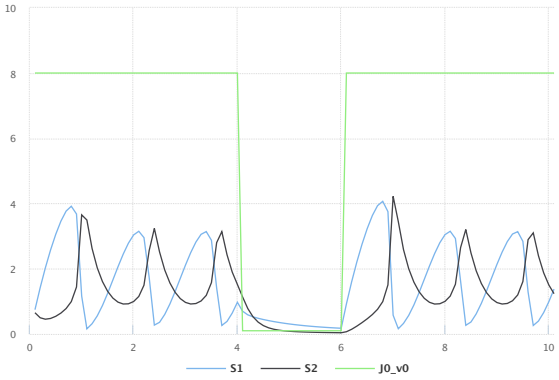

Figure A.9: *The simulation result from the simulation description given in Listing A.6. Simulation with SED-ML web tools [2].*

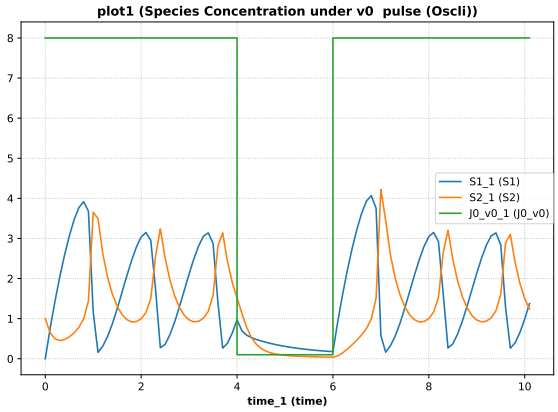

Figure A.10: *Simulation with tellurium [5].*

```
      <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" symbol="urn:sedml:symbol:time" taskReference="task1" />
      </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 A.6: *SED-ML document implementing the perturbation experiment*

A.3.5 2D steady state parameter scan (`L1V3_parameter-scan-2d.omex`)

This example uses a repeatedTask which runs over another repeatedTask which performs a steady state computation. Each repeated simulation task modifies a different parameter.

NOTE: This example produces three dimensional results (time, species concentration, multiple repeats). While SED-ML Level 1 Version 3 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.

```
<?xml version="1.0" encoding="utf-8"?>
<sedML 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="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>
```

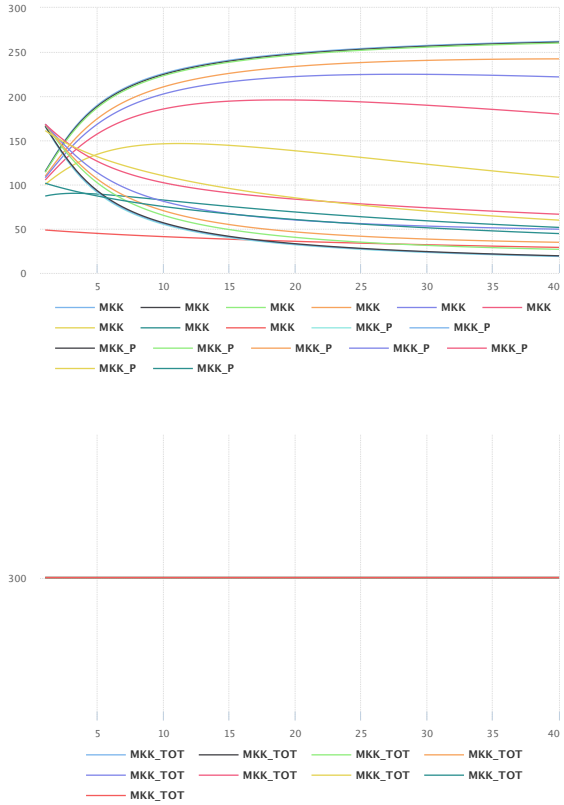

Figure A.11: *The simulation result gained from the simulation description given in Listing A.7. Simulation with SED-ML web tools [2].*

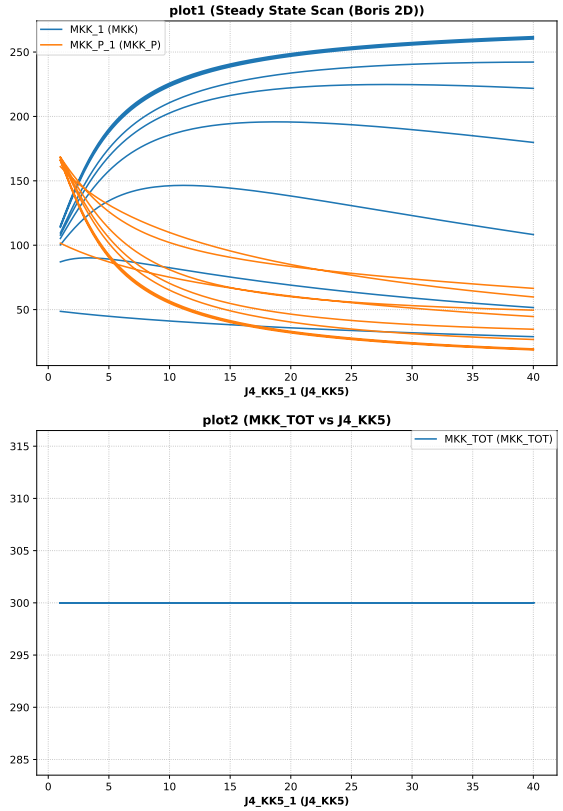

Figure A.12: *Simulation with tellurium [5].*

```
      <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>
    <dataGenerator id="MKK_PP_1" name="MKK_PP_1">
      <listOfVariables>
        <variable id="MKK_PP_1" name="MKK_PP" taskReference="task1" target="/sbml:sbml/sbml:model/
            sbml:listOfSpecies/sbml:species[@id='MKK_PP']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <ci> MKK_PP_1 </ci>
      </math>
    </dataGenerator>
    <dataGenerator id="MKK_TOT" name="MKK_TOT">
      <listOfVariables>
        <variable id="MKK" name="MKK" taskReference="task1" target="/sbml:sbml/sbml:model/
            sbml:listOfSpecies/sbml:species[@id='MKK']" />
        <variable id="MKK_P" name="MKK_P" taskReference="task1" target="/sbml:sbml/sbml:model/
            sbml:listOfSpecies/sbml:species[@id='MKK_P']" />
        <variable id="MKK_PP" name="MKK_PP" taskReference="task1" target="/sbml:sbml/sbml:model/
            sbml:listOfSpecies/sbml:species[@id='MKK_PP']" />
      </listOfVariables>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <plus/>
          <ci> MKK </ci>
          <ci> MKK_P </ci>
          <ci> MKK_PP </ci>
        </apply>
      </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>
    <plot2D id="plot2" name="MKK_TOT vs J4_KK5">
      <listOfCurves>
        <curve id="curve3" logX="false" logY="false" xDataReference="J4_KK5_1" yDataReference="MKK_TOT" /
            >
      </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" />
        <dataSet id="col_4" dataReference="MKK_PP_1" label="MKK_PP_1" />
        <dataSet id="col4" dataReference="MKK_TOT" label="MKK_TOT" />
      </listOfDataSets>
    </report>
  </listOfOutputs>
</sedML>
```

Listing A.7: *SED-ML document implementing the two dimensional steady state parameter scan*

A.4 Simulation experiments with different model languages

SED-ML allows to specify models in various languages, e.g., SBML [14] and CellML [8] (see Section 3.2.3 for more information). This section demonstrates the same simulation experiment with the model either in SBML (Appendix A.4.1) or in CellML (Appendix A.4.2).

A.4.1 Van der Pol oscillator in SBML (`L1V3_vanderpol-sbml.omex`)

The following example provides a SED-ML description for the simulation of the Van der Pol oscillator in SBML [14]. The time-course and the behavior in the phase plane are plotted. The mathematical model and the performed simulation experiment are identical to Appendix A.4.2.

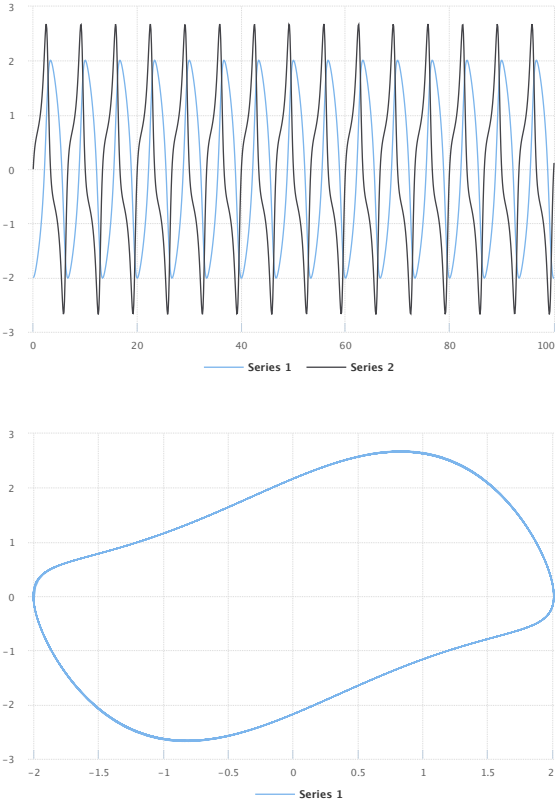

Figure A.13: *The simulation result gained from the simulation description given in Listing A.8. Simulation with SED-ML web tools [2].*

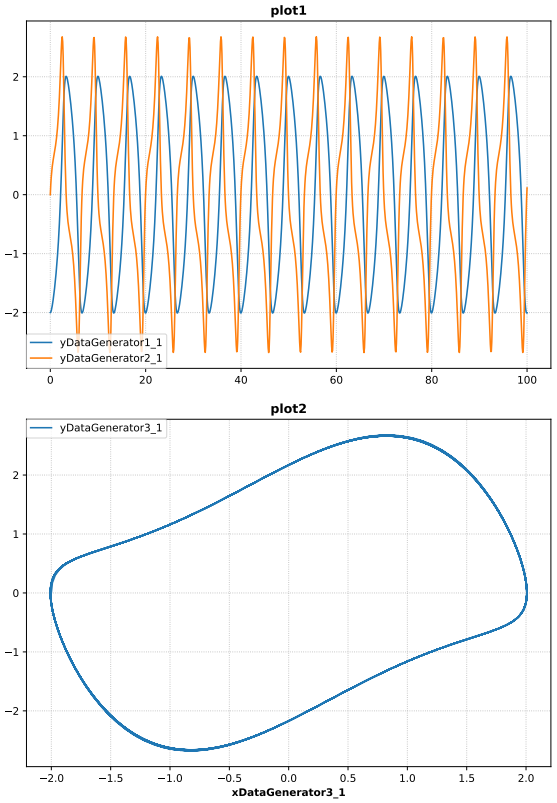

Figure A.14: *Simulation with tellurium [5].*

```
<?xml version='1.0' encoding='UTF-8'?>
<sedML level="1" version="3" xmlns="http://sed-ml.org/sed-ml/level1/version3">
    <listOfSimulations>
        <uniformTimeCourse id="simulation1" initialTime="0" numberOfPoints="1000" outputEndTime="100"
             outputStartTime="0">
            <algorithm kisaoID="KISAO:0000019">
                <listOfAlgorithmParameters>
                    <algorithmParameter kisaoID="KISAO:0000211" value="1e-07"/>
                    <algorithmParameter kisaoID="KISAO:0000475" value="BDF"/>
                    <algorithmParameter kisaoID="KISAO:0000481" value="true"/>
                    <algorithmParameter kisaoID="KISAO:0000476" value="Newton"/>
                    <algorithmParameter kisaoID="KISAO:0000477" value="Dense"/>
                    <algorithmParameter kisaoID="KISAO:0000480" value="0"/>
                    <algorithmParameter kisaoID="KISAO:0000415" value="500"/>
                    <algorithmParameter kisaoID="KISAO:0000467" value="0"/>
                    <algorithmParameter kisaoID="KISAO:0000478" value="Banded"/>
                    <algorithmParameter kisaoID="KISAO:0000209" value="1e-07"/>
                    <algorithmParameter kisaoID="KISAO:0000479" value="0"/>
                </listOfAlgorithmParameters>
            </algorithm>
```

```
        </uniformTimeCourse>
    </listOfSimulations>
    <listOfModels>
        <model id="model" language="urn:sedml:language:sbml" source="vanderpol-sbml.xml"/>
    </listOfModels>
    <listOfTasks>
        <repeatedTask id="repeatedTask" range="once" resetModel="true">
            <listOfRanges>
                <vectorRange id="once">
                    <value> 1 </value>
                </vectorRange>
            </listOfRanges>
            <listOfSubTasks>
                <subTask order="1" task="task1"/>
            </listOfSubTasks>
        </repeatedTask>
        <task id="task1" modelReference="model" simulationReference="simulation1"/>
    </listOfTasks>
    <listOfDataGenerators>
        <dataGenerator id="xDataGenerator1_1">
            <listOfVariables>
                <variable id="xVariable1_1" taskReference="task1" symbol="urn:sedml:symbol:time" />
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> xVariable1_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="yDataGenerator1_1">
            <listOfVariables>
                <variable id="yVariable1_1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species
                    [@id='x']" taskReference="repeatedTask" modelReference="model"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> yVariable1_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="xDataGenerator2_1">
            <listOfVariables>
                <variable id="xVariable2_1" taskReference="task1" symbol="urn:sedml:symbol:time" />
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> xVariable2_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="yDataGenerator2_1">
            <listOfVariables>
                <variable id="yVariable2_1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species
                    [@id='y']" taskReference="repeatedTask" modelReference="model"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> yVariable2_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="xDataGenerator3_1">
            <listOfVariables>
                <variable id="xVariable3_1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species
                    [@id='x']" taskReference="repeatedTask" modelReference="model"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> xVariable3_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="yDataGenerator3_1">
            <listOfVariables>
                <variable id="yVariable3_1" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species
                    [@id='y']" taskReference="repeatedTask" modelReference="model"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> yVariable3_1 </ci>
            </math>
        </dataGenerator>
    </listOfDataGenerators>
    <listOfOutputs>
        <plot2D id="plot1">
            <listOfCurves>
                <curve id="curve1_1" logX="false" logY="false" xDataReference="xDataGenerator1_1"
                    yDataReference="yDataGenerator1_1"/>
                <curve id="curve2_1" logX="false" logY="false" xDataReference="xDataGenerator2_1"
                    yDataReference="yDataGenerator2_1"/>
            </listOfCurves>
        </plot2D>
        <plot2D id="plot2">
            <listOfCurves>
                <curve id="curve3_1" logX="false" logY="false" xDataReference="xDataGenerator3_1"
                    yDataReference="yDataGenerator3_1"/>
            </listOfCurves>
        </plot2D>
    </listOfOutputs>
```

```
</sedML>
```

Listing A.8: *Van der Pol Model (SBML) Simulation Description in SED-ML*

A.4.2 Van der Pol oscillator in CellML (`L1V3_vanderpol-cellml.omex`)

The following example provides a SED-ML description for the simulation of the Van der Pol model in CellML [8]. The time-course and the behavior in the phase plane are plotted. The mathematical model and the performed simulation experiment are identical to Appendix A.4.1.

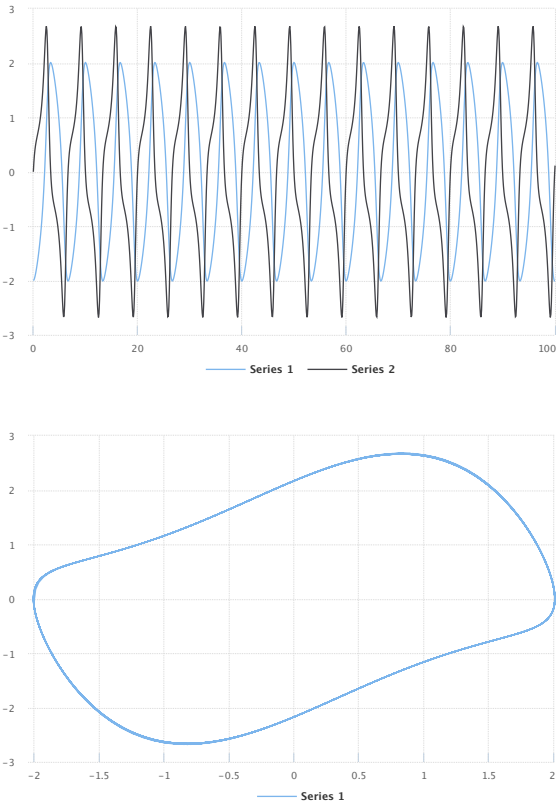

Figure A.15: *The simulation result gained from the simulation description given in Listing A.9. Simulation with SED-ML web tools [2].*

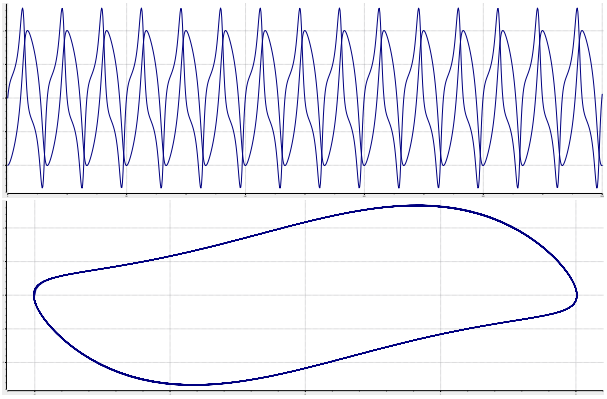

Figure A.16: *Simulation with OpenCOR [10].*

```
<?xml version='1.0' encoding='UTF-8'?>
<sedML level="1" version="3" xmlns="http://sed-ml.org/sed-ml/level1/version3" xmlns:cellml="http://www.
     cellml.org/cellml/1.0#">
    <listOfSimulations>
        <uniformTimeCourse id="simulation1" initialTime="0" numberOfPoints="1000" outputEndTime="100"
             outputStartTime="0">
            <algorithm kisaoID="KISAO:0000019">
                <listOfAlgorithmParameters>
                    <algorithmParameter kisaoID="KISAO:0000211" value="1e-07"/>
                    <algorithmParameter kisaoID="KISAO:0000475" value="BDF"/>
                    <algorithmParameter kisaoID="KISAO:0000481" value="true"/>
                    <algorithmParameter kisaoID="KISAO:0000476" value="Newton"/>
                    <algorithmParameter kisaoID="KISAO:0000477" value="Dense"/>
                    <algorithmParameter kisaoID="KISAO:0000480" value="0"/>
                    <algorithmParameter kisaoID="KISAO:0000415" value="500"/>
                    <algorithmParameter kisaoID="KISAO:0000467" value="0"/>
                    <algorithmParameter kisaoID="KISAO:0000478" value="Banded"/>
                    <algorithmParameter kisaoID="KISAO:0000209" value="1e-07"/>
                    <algorithmParameter kisaoID="KISAO:0000479" value="0"/>
                </listOfAlgorithmParameters>
            </algorithm>
        </uniformTimeCourse>
    </listOfSimulations>
    <listOfModels>
```

```
        <model id="model" language="urn:sedml:language:cellml.1_0" source="vanderpol-model.cellml"/>
    </listOfModels>
    <listOfTasks>
        <repeatedTask id="repeatedTask" range="once" resetModel="true">
            <listOfRanges>
                <vectorRange id="once">
                    <value> 1 </value>
                </vectorRange>
            </listOfRanges>
            <listOfSubTasks>
                <subTask order="1" task="task1"/>
            </listOfSubTasks>
        </repeatedTask>
        <task id="task1" modelReference="model" simulationReference="simulation1"/>
    </listOfTasks>
    <listOfDataGenerators>
        <dataGenerator id="xDataGenerator1_1">
            <listOfVariables>
                <variable id="xVariable1_1" target="/cellml:model/cellml:component[@name='main']/
                    cellml:variable[@name='t']" taskReference="repeatedTask"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> xVariable1_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="yDataGenerator1_1">
            <listOfVariables>
                <variable id="yVariable1_1" target="/cellml:model/cellml:component[@name='main']/
                    cellml:variable[@name='x']" taskReference="repeatedTask"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> yVariable1_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="xDataGenerator2_1">
            <listOfVariables>
                <variable id="xVariable2_1" target="/cellml:model/cellml:component[@name='main']/
                    cellml:variable[@name='t']" taskReference="repeatedTask"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> xVariable2_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="yDataGenerator2_1">
            <listOfVariables>
                <variable id="yVariable2_1" target="/cellml:model/cellml:component[@name='main']/
                    cellml:variable[@name='y']" taskReference="repeatedTask"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> yVariable2_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="xDataGenerator3_1">
            <listOfVariables>
                <variable id="xVariable3_1" target="/cellml:model/cellml:component[@name='main']/
                    cellml:variable[@name='x']" taskReference="repeatedTask"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> xVariable3_1 </ci>
            </math>
        </dataGenerator>
        <dataGenerator id="yDataGenerator3_1">
            <listOfVariables>
                <variable id="yVariable3_1" target="/cellml:model/cellml:component[@name='main']/
                    cellml:variable[@name='y']" taskReference="repeatedTask"/>
            </listOfVariables>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <ci> yVariable3_1 </ci>
            </math>
        </dataGenerator>
    </listOfDataGenerators>
    <listOfOutputs>
        <plot2D id="plot1">
            <listOfCurves>
                <curve id="curve1_1" logX="false" logY="false" xDataReference="xDataGenerator1_1"
                    yDataReference="yDataGenerator1_1"/>
                <curve id="curve2_1" logX="false" logY="false" xDataReference="xDataGenerator2_1"
                    yDataReference="yDataGenerator2_1"/>
            </listOfCurves>
        </plot2D>
        <plot2D id="plot2">
            <listOfCurves>
                <curve id="curve3_1" logX="false" logY="false" xDataReference="xDataGenerator3_1"
                    yDataReference="yDataGenerator3_1"/>
            </listOfCurves>
        </plot2D>
    </listOfOutputs>
```

```
101 </sedML>
```

Listing A.9: *Van der Pol Model (CellML) Simulation Description in SED-ML*

A.5 Reproducing publication results

SED-ML allows to describe simulation experiments from publications in a reproducible manner. This section provides such examples.

A.5.1 Le Loup model (L1V3_leloup-sbml.omex)

The following example provides a SED-ML description for the simulation of the model based on the publication [16].

The model is referenced by its SED-ML id `model1` and refers to the model with the MIRIAM URN `urn:miriam:biomodels.db:BIOMD0000000021`. A second model is defined in 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 dataGenerators are defined, which are the prerequisite for defining the simulation output. The first dataGenerator with `id` `time` collects the simulation time. `tim1` maps on the `Mt` entity in the model that is used in `task1` which in the model `model1`. The dataGenerator named `per_tim1` 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 curves and provides the time course of the simulation using the tim mRNA concentrations from both tasks. The second plot shows the `per_tim` concentration against the `tim` concentration for the oscillating model. The third plot shows the same plot for the chaotic model. The resulting three plots are depicted in Figure A.17 and A.18 .

```
<?xml version="1.0" encoding="utf-8"?>
<sedML  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">
```

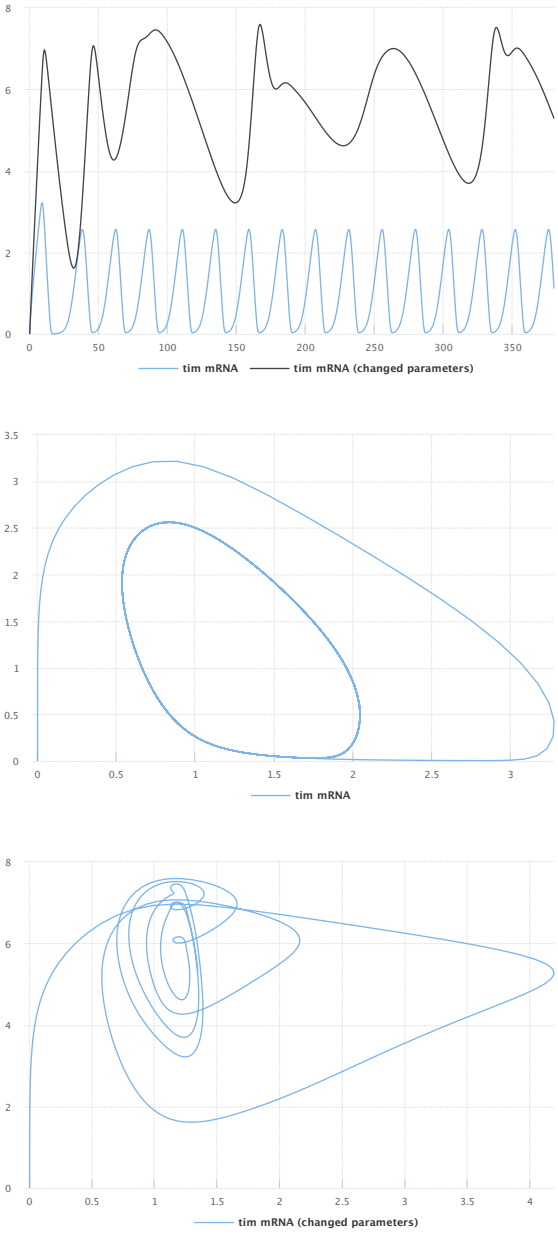

Figure A.17: *The simulation result gained from the simulation description given in Listing A.10. Simulation with SED-ML web tools [2].*

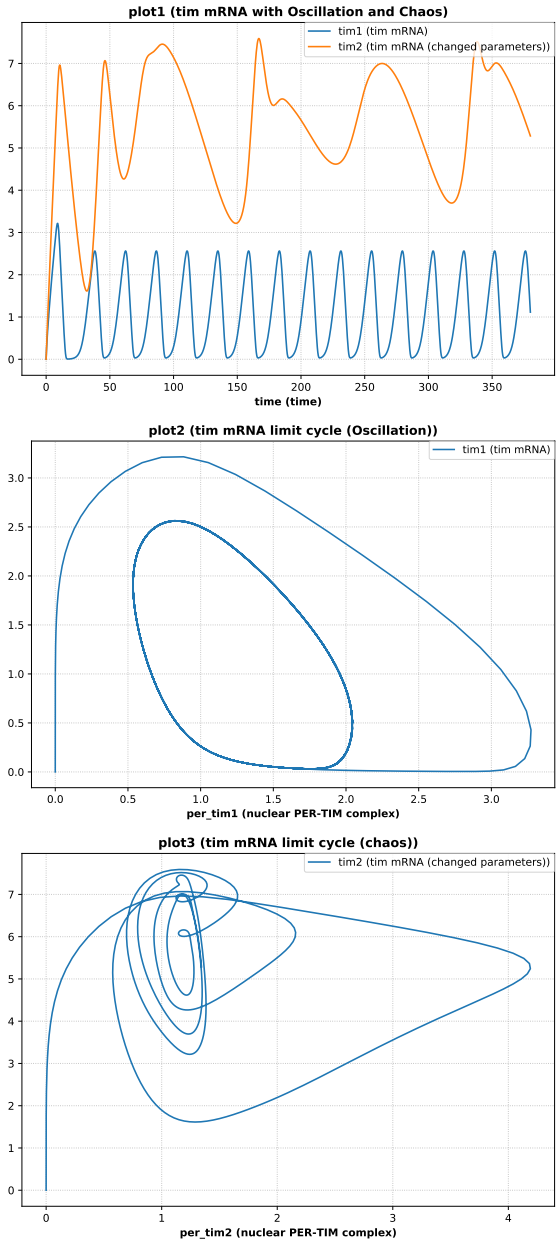

Figure A.18: *Simulation with tellurium [5].*

```
43          <ci> v1a </ci>
44        </math>
45      </dataGenerator>
46      <dataGenerator id="tim2" name="tim mRNA (changed parameters)">
47        <listOfVariables>
48          <variable id="v2" taskReference="task2" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
                sbml:species[@id='Mt']" />
49        </listOfVariables>
50        <math xmlns="http://www.w3.org/1998/Math/MathML">
51          <ci> v2 </ci>
52        </math>
53      </dataGenerator>
54      <dataGenerator id="per_tim2" name="nuclear PER-TIM complex">
55        <listOfVariables>
56          <variable id="v2a" taskReference="task2" target="/sbml:sbml/sbml:model/sbml:listOfSpecies/
                sbml:species[@id='Cn']" />
57        </listOfVariables>
58        <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 A.10: *LeLoup Model Simulation Description in SED-ML*

A.5.2 IkappaB signaling (`L1V3_ikkapab.omex`)

The following example provides a SED-ML description for the simulation of the IkappaB-NF-kappaB signaling module described in [12].

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 [15].

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 plotted against the simulation time as shown in Figure A.19 and A.20.

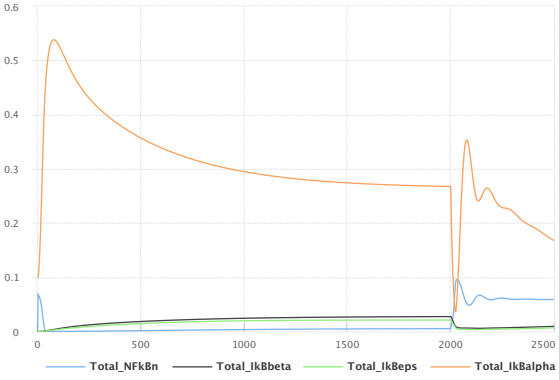

Figure A.19: *The simulation result gained from the simulation description given in Listing A.11. Simulation with SED-ML web tools [2].*

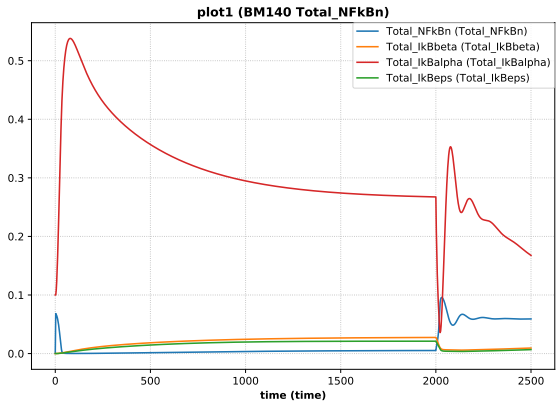

Figure A.20: *Simulation with tellurium [5].*

```
<?xml version="1.0" encoding="utf-8"?>
<sedML 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" />
            <curve  id="c2" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_IkBbeta" />
        <curve  id="c3" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_IkBeps" />
        <curve id="c4" logX="false" logY="false" xDataReference="time"
        yDataReference="Total_IkBalpha" />
      </listOfCurves>
    </plot2D>
  </listOfOutputs>
</sedML>
```

Listing A.11: *IkappaB-NF-kappaB signaling Model Simulation Description in SED-ML*

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:attribute name="format" type="xs:anyURI" use="optional" />
                    <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>
```

```
692
693     <xs:element name="listOfSurfaces">
694         <xs:complexType>
695             <xs:complexContent>
696                 <xs:extension base="SEDBase">
697                     <xs:sequence>
698                         <xs:element ref="surface" maxOccurs="unbounded" />
699                     </xs:sequence>
700                 </xs:extension>
701             </xs:complexContent>
702         </xs:complexType>
703     </xs:element>
704
705     <xs:element name="listOfDataSets">
706         <xs:complexType>
707             <xs:complexContent>
708                 <xs:extension base="SEDBase">
709                     <xs:sequence>
710                         <xs:element ref="dataSet" maxOccurs="unbounded" />
711                     </xs:sequence>
712                 </xs:extension>
713             </xs:complexContent>
714         </xs:complexType>
715     </xs:element>
716
717     <xs:element name="listOfChanges">
718         <xs:complexType>
719             <xs:complexContent>
720                 <xs:extension base="SEDBase">
721                     <xs:choice minOccurs="0" maxOccurs="unbounded">
722                         <xs:element ref="changeAttribute" />
723                         <xs:element ref="changeXML" />
724                         <xs:element ref="addXML" />
725                         <xs:element ref="removeXML" />
726                         <xs:element ref="computeChange" />
727                     </xs:choice>
728                 </xs:extension>
729             </xs:complexContent>
730         </xs:complexType>
731     </xs:element>
732
733     <xs:element name="listOfRanges">
734         <xs:complexType>
735             <xs:complexContent>
736                 <xs:extension base="SEDBase">
737                     <xs:choice maxOccurs="unbounded">
738                         <xs:element ref="uniformRange" />
739                         <xs:element ref="vectorRange" />
740                         <xs:element ref="functionalRange" />
741                     </xs:choice>
742                 </xs:extension>
743             </xs:complexContent>
744         </xs:complexType>
745     </xs:element>
746
747     <xs:complexType name="repeatedTaskListOfChanges">
748         <xs:complexContent>
749             <xs:extension base="SEDBase">
750                 <xs:sequence>
751                     <xs:element ref="setValue" minOccurs="0" maxOccurs="unbounded" />
752                 </xs:sequence>
753             </xs:extension>
754         </xs:complexContent>
755     </xs:complexType>
756
757     <xs:element name="listOfSubTasks">
758         <xs:complexType>
759             <xs:complexContent>
760                 <xs:extension base="SEDBase">
761                     <xs:sequence>
762                         <xs:element ref="subTask" maxOccurs="unbounded" />
763                     </xs:sequence>
764                 </xs:extension>
765             </xs:complexContent>
766         </xs:complexType>
767     </xs:element>
768 </xs:schema>
```

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

Bibliography

[1] 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.

[2] F. T. Bergmann, D. Nickerson, D. Waltemath, and M. Scharm. SED-ML web tools: generate, modify and export standard-compliant simulation studies. *Bioinformatics*, 33(8):1253–1254, 2017.

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

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

[5] K. Choi, J. K. Medley, C. Cannistra, M. König, L. Smith, K. Stocking, and H. M. Sauro. Tellurium: A python based modeling and reproducibility platform for systems biology. *bioRxiv*, 2016.

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

[10] A. Garny and P. J. Hunter. OpenCOR: a modular and interoperable approach to computational biology. *Frontiers in physiology*, 6, 2015.

[11] 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.

[12] A. Hoffmann, A. Levchenko, M. L. Scott, and D. Baltimore. The IκB-NF-κB signaling module: temporal control and selective gene activation. *Science*, 298(5596):1241–1245, 2002.

[13] 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.

[14] 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.

[15] 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.

[16] J.-C. Leloup, D. Gonze, and A. Goldbeter. Limit cycle models for circadian rhythms based on transcriptional regulation in drosophila and neurospora. *Journal of Biological Rhythms*, 14(6):433–448, 1999.

[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] 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.

[19] 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.

[20] D. Waltemath, R. Adams, F. T. Bergmann, M. Hucka, F. Kolpakov, A. K. Miller, I. I. Moraru, D. Nickerson, S. Sahle, J. L. Snoep, and N. Le Novère. Reproducible computational biology experiments with SED-ML: the simulation experiment description markup language. *BMC systems biology*, 5(1):198, 2011.