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

Frank T. Bergmann¹, Jonathan Cooper²*, Nicolas Le Novère³,⁴, David Nickerson⁵ and
Dagmar Waltemath⁶

¹Department Modeling of Biological Processes, University of Heidelberg, Heidelberg, Germany
²Department of Computer Science, University of Oxford, UK
³The Babraham Institute, Cambridge, UK
⁴EMBL European Bioinformatics Institute, Cambridge, UK
⁵Auckland Bioengineering Institute, University of Auckland, Auckland, New Zealand
⁶Department of Systems Biology and Bioinformatics, University of Rostock, Rostock, Germany

Summary

The number, size and complexity of computational models of biological systems are growing at an ever increasing pace. It is imperative to build on existing studies by reusing and adapting existing models and parts thereof. The description of the structure of models is not sufficient to enable the reproduction of simulation results. One also needs to describe the procedures the models are subjected to, as recommended by the Minimum Information About a Simulation Experiment (MIASE) guidelines.

This document presents Level 1 Version 2 of the Simulation Experiment Description Markup Language (SED-ML), a computer-readable format for encoding simulation and analysis experiments to apply to computational models. SED-ML files are encoded in the Extensible Markup Language (XML) and can be used in conjunction with any XML-based model encoding format, such as CellML or SBML. A SED-ML file includes details of which models to use, how to modify them prior to executing a simulation, which simulation and analysis procedures to apply, which results to extract and how to present them. Level 1 Version 2 extends the format by allowing the encoding of repeated and chained procedures.

*To whom correspondence should be addressed. Email: sed-ml-editors@lists.sourceforge.net
Simulation Experiment Description Markup
Language (SED-ML) :
Level 1 Version 2

December 2, 2013

Editors
Frank T. Bergmann
Jonathan Cooper
Nicolas Le Novère
David Nickerson
Dagmar Waltemath

University of Washington, Seattle, USA
University of Oxford, UK
European Bioinformatics Institute, UK
Auckland Bioengineering Institute, New Zealand
University of Rostock, Germany

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

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

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

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

1.1 Motivation: A sample experiment

1.1.1 A simple time-course simulation

1.1.2 Applying pre-processing

1.1.3 Applying post-processing

2 SED-ML technical specification

2.1 Conventions used in this document

2.1.1 UML Classes

2.1.2 UML Relationships

2.1.3 XML Schema language elements

2.1.4 Type extensions

2.2 Concepts used in SED-ML

2.2.1 MathML subset

2.2.2 URI Scheme

2.2.3 XPath usage

2.2.4 KiSAO

2.2.5 SED-ML resources

2.3 General attributes and classes

2.3.1 id

2.3.2 name

2.3.3 SEDBase

2.3.4 SED-ML top level element

2.3.5 Reference relations

2.3.6 Variable

2.3.7 Parameter

2.3.8 ListOf* containers

2.4 SED-ML Components

2.4.1 Model

2.4.1.1 language

2.4.1.2 source

2.4.2 Change

2.4.2.1 NewXML

2.4.2.2 AddXML

2.4.2.3 ChangeXML

2.4.2.4 RemoveXML

2.4.2.5 ChangeAttribute

2.4.2.6 ComputeChange

2.4.3 Simulation

2.4.3.1 Algorithm

2.4.3.2 AlgorithmParameter

Copyright 2015 The Author(s). Published by Journal of Integrative Bioinformatics. This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License (http://creativecommons.org/licenses/by-nc-nd/3.0/).


http://journal.imbio.de

doi:10.2390/biecoll-jib-2015-262
1. Introduction

The number of available computational models of biological systems is growing at an ever increasing pace. At the same time, their size and complexity are also increasing. The need to build on existing studies by reusing models therefore becomes more imperative. It is now generally accepted that one needs to be able to exchange the biochemical and mathematical structure of models. The efforts to standardise the representation of computational models in various areas of biology, such as the Systems Biology Markup Language (SBML, [Hucka et al., 2003]), CellML [Cuellar et al., 2003] or NeuroML [Goddard et al., 2001], resulted in such an increase of the exchange and re-use of models. However, the description of the structure of models is not sufficient to enable the reproduction of simulation results. One also needs to describe the procedures the models are subjected to, as described by the Minimum Information About a Simulation Experiment (MIASE) [Waltemath et al., 2011a].

This document presents Level 1 Version 2 of the Simulation Experiment Description Markup Language (SED-ML), a computer-readable format for encoding simulation experiments. SED-ML files are encoded in the Extensible Markup Language (XML) [Bray et al., 2006]. The SED-ML format is defined by an XML Schema [Fallside et al., 2001]. Level 1 Version 2 is the successor of Level 1 Version 1, which is described in [Waltemath et al., 2011b].
1.1 Motivation: A sample experiment

The *repressilator* is a rather small, though famous, model that is capable of displaying rich and variable behaviors. We will use this model to demonstrate how a simulation experiment can be described simply and effectively. The simulation example is taken from Waltemath et al. [2011a].

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

1.1.1 A simple time-course simulation

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

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

Following those steps and performing the simulation in the simulation tool COPASI [Hoops et al., 2006] led to the result shown in Figure 1.1.

1.1.2 Applying pre-processing

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

1. Import the model as above.
2. Change the value of the parameter tps\_repr from “0.0005” to “1.3e-05”.

3. Change the value of the parameter tps\_active from “0.5” to “0.013”.

4. Select a deterministic method.

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

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

Figure 1.2 shows the result of the simulation.

### 1.1.3 Applying post-processing

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

(Please note that the description steps 1 - 4 remain as given in section 1.1.1 above.)

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

6. Compute the highest value for each of the repressor proteins, max(lacI(t)), max(tetR(t)), max(cI(t)).

7. Normalize the data for each of the repressor proteins by dividing each time point by the maximum value, i.e. lacI(t)/max(lacI(t)), tetR(t)/max(tetR(t)), and cI(t)/max(cI(t)).

8. Plot the normalized lacI protein as a function of the normalized cI, the normalized cI as a function of the normalized tetR protein, and the normalized tetR protein against the normalized lacI protein in a 2D plot.

Figure 1.3 on the following page illustrates the result of the simulation after post-processing of the output data.
Figure 1.3: Time-course simulation of the repressilator model, imported from BioModels Database and simulated in COPASI, showing the normalized temporal evolution of repressor proteins lacI, tetR and cl in phase-plane. (taken from Waltemath et al. [2011a])
2. SED-ML technical specification

This document represents the technical specification of SED-ML. We also provide an XML Schema [W3C, 2004] and a UML class diagram representation of that XML Schema (Appendix A). UML class diagrams are a subset of the Unified Markup Language notation (UML, [OMG, 2009]). Sample experiment descriptions are given as XML snippets that comply with the XML Schema.

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

2.1.1 UML Classes

A SED-ML UML class (Figure 2.1) consists of a class name (ClassName) and a number of attributes (attribute) each of a specific data type (type). The SED-ML UML specification does not make use of UML operations.

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

2.1.2 UML Relationships

2.1.2.1 UML Relation Types

Links between classes specify the connection of objects with each other (Figure 2.2). The different relation types used in the SED-ML specification include aggregation, composite aggregation, and generalisation. The label on the line is called symbol (label) and describes the relation of the objects of both classes.

The association (Figure 2.3) indicates the existence of a connection between the objects of the participating classes. Often associations are directed to show how the label should be read (in which direction). Associations can be uni-directional (one arrowhead), or bidirectional (zero or two arrowheads).

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

The composite aggregation (Figure 2.4 on the next page, bottom) indicates that the objects of the participating classes are connected in a way that one class (Whole) consists of several parts (Part). In
contrast to the aggregation, the subelements (Part) are dependent on the parent class (Whole). An example is that a university (Whole) consists of a number of departments (Part) which have a so-called “lifetime responsibility” with the university, e.g. if the university vanishes, the departments will vanish with it [Bell, 2003].

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

2.1.3 XML Schema language elements

The main building blocks of an XML Schema specification are:
simple and complex types
• element specifications
• attribute specifications

XML Schema definitions create new types, declarations define new elements and attributes. The definition of new (simple and complex) types can be based on a number of already existing, predefined types (string, boolean, float). Simple types are restrictions or extensions of predefined types. Complex types describe how attributes can be assigned to elements and how elements can contain further elements. The current SED-ML XML Schema only makes use of complex type definitions. An example for a complex type definition is given in Listing 2.1:

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

Listing 2.1: Complex Type definition of the SED-ML computeChange element

It shows the declaration of an element called computeChange that is used in SED-ML to change mathematical expressions. The element is defined using an unnamed complex type which is built of further elements called listOfVariables, listOfParameters, and math. Additionally, the element computeChange has an attribute target declared. Please note that the definition of the elements inside the complex type are only referred to and will be found elsewhere in the schema.

The nesting of elements in the schema can be expressed using the xs:sequence (a sequence of elements), xs:choice (an alternative of elements to choose from), or xs:all (a set of elements that can occur in any order) concepts. The current SED-ML XML Schema only uses the sequence of elements.

2.1.3.1 Multiplicities

The standard multiplicity for each defined element is 1. Explicit multiplicity is to be defined using the minOccurs and maxOccurs attributes inside the complex type definition, as shown in Listing 2.2.

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

Listing 2.2: Multiplicity for complex types in XML Schema

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

2.1.4 Type extensions

XML Schema offers mechanisms to restrict and extend previously defined complex types. Extensions add element or attribute declarations to existing types, while restrictions restrict the types by adding further characteristics and requirements (facets) to a type. An example for a type extension is given in Listing 2.3.
The `sedML` element is an extension of the previously defined `SEDBase` type. It extends `SEDBase` by a sequence of five additional elements (`listOfSimulations`, `listOfModels`, `listOfTasks`, `listOfDataGenerators`, and `listOfOutputs`) and two new attributes `version` and `level`. 

---

Listing 2.3: Definition of the `sedML` type through extension of `SEDBase` in SED-ML
2.2 Concepts used in SED-ML

2.2.1 MathML subset

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

2.2.1.1 MathML elements

In order to support for the SED-ML format easier to implement we restrict the MathML subset to the following elements:

- **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, arccsc, arccsc, arccot, arccoth, arcsinh, arccosh, arctanh, arcsech, arccsch, arccoth
- **constants**: true, false, notanumber, pi, infinity, exponentiale
- **MathML annotations**: semantics, annotation, annotation-xml

2.2.1.2 MathML Symbols

All the operations listed above only operate on scalar values. However, as one of SED-ML’s aims is to provide post processing on the results of simulation experiments, we need to enhance this basic set of operations by some aggregate functions. Therefore a defined set of MathML symbols that represent vector values are supported by SED-ML Level 1 Version 2. To simplify the use of SED-ML L1V2 the only symbols to be used are the identifiers of variables defined in the listOfVariables of DataGenerators. These variables represent the data collected from the simulation experiment with the associated task.

2.2.1.3 MathML functions

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

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

```
1  <apply>
2   <csymbol encoding="text" definitionURL="http://sed-ml.org/#min">
3     min
4   </csymbol>
5   <ci>variableId</ci>
6  </apply>
```

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

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

```
1  <apply>
2   <csymbol encoding="text" definitionURL="http://sed-ml.org/#max">
3     max
4   </csymbol>
5   <ci>variableId</ci>
6  </apply>
```

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

- **sum**: All values of the variable returned by the simulation experiment are summed (Listing 2.6).
• product: All values of the variable returned by the simulation experiment are multiplied (Listing 2.7).

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

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

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

2.2.2 URI Scheme

URIs are needed at different points in SED-ML Level 1 Version 2. Firstly, they are the preferred mechanism to refer to model encodings. Secondly, they are used to specify the language of the referenced model. Thirdly, they enable addressing implicit model variables. Finally, annotations of SED-ML elements should be provided with a standardised annotation scheme.

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

2.2.2.1 Model references

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

One model repository that is part of MIRIAM resources is the BioModels Database [Li et al., 2010]. Its data resource name in MIRIAM is urn:miriam:biomodels.db. To refer to a particular model, a standardised identifier scheme is defined in MIRIAM Resources1. The ID entry maps to a particular model in the model repository. That model is never deleted. A sample BioModels Database ID is BIOMD0000000048. Together with the data resource name it becomes unambiguously identifiable by the URN urn:miriam:biomodels.db:BIOMD0000000048 (in this case referring to the 1999 Kholodenko model on EGFR signaling).

SED-ML recommends to follow the above scheme for model references, if possible. SED-ML does not specify how to resolve the URNs. However, MIRIAM Resources offers web services to do so2. For the above example of the urn:miriam:biomodels.db:BIOMD0000000048 model, the resolved URL may look like:

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

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

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

1http://www.ebi.ac.uk/miriam/
2http://www.ebi.ac.uk/miriam/
2.2.2.2 Language references

To specify the language a model is encoded in, a set of pre-defined SED-ML URNs can be used. The structure of SED-ML language URNs is urn:sedml:language:name.version. SED-ML allows to specify a model representation format very generally as being XML, if no standardised representation format has been used to encode the model. On the other hand, one can be as specific as defining a model being in a particular version of a language, as “SBML Level 2, Version 2, Revision 1”.

The list of URNs is available from http://sed-ml.org/. Further information on the language attribute is provided in Section 2.4.1.1.

2.2.2.3 Implicit variables

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

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

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

2.2.2.4 Annotations

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

2.2.3 XPath usage

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

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

2. Within a Change definition, where XPath is used to identify the target XML to which a change should be applied.

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

\[
\text{/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='X']} \quad \text{-CORRECT}
\]

is preferable to

\[
\text{/sbml/model/listOfSpecies/species[@id='X']} \quad \text{-INCORRECT}
\]

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:
2.2.4 KiSAO

The Kinetic Simulation Algorithm Ontology (KiSAO, [Courtot et al., 2011]) is used in SED-ML to specify simulation algorithms and uniquely identify algorithm parameters. KiSAO is a community-driven approach of classifying and structuring simulation approaches by model characteristics and numerical characteristics. The ontology is available in OWL format from BioPortal at http://purl.bioontology.org/ontology/KiSAO.

SED-ML refers to terms from KiSAO as referencing a simulation algorithm, or its parameters, solely through a name is error prone and ambiguous. After all, typing mistakes or language differences complicate the identification of the correct algorithm. Additionally, many algorithms exist under multiple names or abbreviation.

The identification of a simulation algorithm through KISAO not only identifies the simulation algorithm to be used in the SED-ML simulation, it also enables software to find a related algorithm, 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 not exist within KISAO, please request one from the project homepage at http://www.biomodels.net/kisao/.

2.2.5 SED-ML resources

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

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

2.3.1 id

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

\[
\begin{align*}
\text{letter} &::= 'a'..'z','A'..'Z' \\
\text{digit} &::= '0'..'9' \\
\text{idChar} &::= \text{letter} | \text{digit} | '.' \\
\text{SId} &::= (\text{letter} | '.') \text{idChar}* \\
\end{align*}
\]

Figure 2.7: The definition of the type SId

For a detailed description see also the SBML specification on the “Type SId” [Hucka et al., 2010, p. 11]. All ids have a global scope, i.e. the id must be unambiguous throughout a whole SED-ML document. As such it identifies the constituent it is related to.

An example for a defined id is given in Listing 2.8.

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

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

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

2.3.2 name

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

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

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

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

2.3.3 SEDBase

SEDBase is the base class of SED-ML Level 1 Version 2. All other classes are derived from it. As such it provides means to attach additional information on all other classes (Figure 2.8 on the next page). That information can be specified by human readable Notes or custom Annotations.
Table 2.1 shows all attributes and sub-elements for the SEDBase element as defined by the SED-ML Level 1 Version 2 XML Schema.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid(^\circ)</td>
<td>page 18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes(^\circ)</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation(^\circ)</td>
<td>page 19</td>
</tr>
</tbody>
</table>

Table 2.1: Attributes and nested elements for SEDBase. \(xy^\circ\) denotes optional elements and attributes.

### 2.3.3.1 metaid

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

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

### 2.3.3.2 Notes

A note is considered a human-readable description of the element it is assigned to. It serves to display information to the user. Instances of the Notes class may contain any valid XHTML [Pemberton et al., 2002], ranging from short comments to whole HTML pages for display in a Web browser. The namespace URL for XHTML content inside the Notes class is http://www.w3.org/1999/xhtml. It may either be declared in the sedML XML element, or directly used in top level XHTML elements contained within the notes element. For further options of how to set the namespace and detailed examples, please refer to [Hucka et al., 2010, p. 14].

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlns:string</td>
<td>“<a href="http://www.w3.org/1999/xhtml%E2%80%9D">http://www.w3.org/1999/xhtml”</a></td>
<td>page 21</td>
</tr>
</tbody>
</table>

| sub-elements     | well-formed content permitted in XHTML |

Table 2.2: Attributes and nested elements for Notes. \(xy^\circ\) denotes optional elements and attributes.

doi:10.2390/biecoll-jib-2015-262
Listing 2.10 shows the use of the \texttt{notes} element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

\begin{verbatim}
<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>
\end{verbatim}

\textit{Listing 2.10: The \texttt{notes} element}

In this example, the namespace declaration is inside the \texttt{notes} element and the note is related to the \texttt{sedML} root element of the SED-ML file. A note may, however, occur inside any SED-ML XML element, except \texttt{note} itself and \texttt{annotation}.

\subsection{Annotation}

An \textit{annotation} is considered a computer-processible piece of information. Annotations may contain any valid XML content. For further guidelines on how to use annotations, we would like to encourage the reading of the corresponding section in the SBML specification [Hucka et al., 2010, pp. 14-16]. The style of annotations in SED-ML is briefly described in Section 2.2.2.4 on page 15.

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

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
attribute & description \\
\hline
\texttt{none} & \texttt{none in the SED-ML namespace} \\
\hline
\end{tabular}
\caption{Attributes and nested elements for \texttt{Annotation}. \texttt{xy} denotes optional elements and attributes.}
\end{table}

Listing 2.11 shows the use of the \texttt{annotation} element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

\begin{verbatim}
<sedML [...]>
  <model id="model1" metaid="_001" language="urn:sedml:language:cellml"
d6613d7e1051b3eff2bb1d3d419a445bb8c754ad/leloup_gonze_goldbeter_1999_a.cellml">
    <annotation>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
               xmlns:bqmodel="http://biomodels.net/model-qualifiers/"
               xmlns:xmlns=rdf:resource="urn:miriam:pubmed:10415827"/>
    </annotation>
  </model>
</sedML>
\end{verbatim}

\textit{Listing 2.11: The \texttt{annotation} element}

In that example, a SED-ML \texttt{model} element is annotated with a reference to the original publication. The \texttt{model} contains an \texttt{annotation} that uses the \texttt{biomodels.net} model-qualifier \texttt{isDescribedBy} to link to the external resource \texttt{urn:miriam:pubmed:10415827}. In natural language the annotation content could be interpreted as “The model is described by the published article available from \texttt{pubmed} under ID 10415827”. The example annotation follows the proposed URI Scheme suggested by the MIRIAM reference standard. The MIRIAM URN can be resolved to the PubMed (\texttt{http://pubmed.gov}) publication with ID 10415827, namely the article “Alternating oscillations and chaos in a model of two coupled biochemical oscillators driving successive phases of the cell cycle.” published by Romond et al. in 1999.
2.3.4 SED-ML top level element

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

![Diagram of SED-ML classes](image)

Figure 2.9: The sub-classes of SED-ML

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

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

The basic XML structure of a SED-ML file is shown in listing 2.12 on the following page.
Table 2.4: Attributes and nested elements for SED-ML. $\times^o$ denotes optional elements and attributes.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid$^o$</td>
<td>page 18</td>
<td></td>
</tr>
<tr>
<td>xmlns</td>
<td>page 21</td>
<td></td>
</tr>
<tr>
<td>level</td>
<td>page 21</td>
<td></td>
</tr>
<tr>
<td>version</td>
<td>page 21</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes$^o$</td>
<td>page 18</td>
<td></td>
</tr>
<tr>
<td>annotation$^o$</td>
<td>page 19</td>
<td></td>
</tr>
<tr>
<td>listOfModels$^o$</td>
<td>page 28</td>
<td></td>
</tr>
<tr>
<td>listOfSimulations$^o$</td>
<td>page 29</td>
<td></td>
</tr>
<tr>
<td>listOfTasks$^o$</td>
<td>page 30</td>
<td></td>
</tr>
<tr>
<td>listOfDataGenerators$^o$</td>
<td>page 30</td>
<td></td>
</tr>
<tr>
<td>listOfOutputs$^o$</td>
<td>page 31</td>
<td></td>
</tr>
</tbody>
</table>

Listing 2.12: The SED-ML root element

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

2.3.4.1 xmlns

The `xmlns` attribute declares the namespace for the SED-ML document. The pre-defined namespace for SED-ML documents is `http://sed-ml.org/sed-ml/level1/version2`.

In addition, SED-ML makes use of the MathML namespace `http://www.w3.org/1998/Math/MathML` to enable the encoding of mathematical expressions in MathML 2.0. SED-ML uses a subset of MathML as described in Section 2.2.1 on page 13.

SED-ML notes use the XHTML namespace `http://www.w3.org/1999/xhtml`. The Notes class is described in Section 2.3.3.2 on page 18.

Additional external namespaces might be used in annotations.

2.3.4.2 level

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

The level attribute is required and its value is a fixed decimal. For SED-ML Level 1 Version 2 the value is set to 1, as shown in the example in Listing 2.12.

2.3.4.3 version

The current SED-ML version is “version 2”. Minor revisions containing corrections and refinements of SED-ML elements, or new constructs which do not affect backwards compatibility, will lead to the definition of forthcoming versions.

The version attribute is required and its value is a fixed decimal. For SED-ML Level 1 Version 2 the...
value is set to 2, as shown in the example in Listing 2.12.

2.3.5 Reference relations

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

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

The definition of a Task object requires a reference to a particular Model object (modelReference, see Section 2.3.5.1 on page 22); furthermore, the Task object must be associated with a particular Simulation object (simulationReference, see Section 2.3.5.3 on page 23).

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

a. The reference association might occur between a Variable object and a Model object, e.g. if the variable is to define a Change. In that case the variable element contains a modelReference to refer to the particular model that contains the variable used to define the change (see Section 2.3.5.1 on page 22).

b. If the reference is used as an association between a Variable object and an AbstractTask object inside the dataGenerator class, then the variable element contains a taskReference to unambiguously refer to an observable in a given task (see Section 2.3.5.2 on page 23).

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

2.3.5.1 modelReference

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

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

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

The modelReference attribute of the variable element contains the id of a model that is defined in the document. Listing 2.13 shows the use of the modelReference element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.
In the example, a change is applied on model m0001. In the computeChange element a list of variables is defined. One of those variable is v1 which is defined in another model (cellML). The XPath expression given in the target attribute identifies the variable in the model which carries the ID cellML.

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

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

2.3.5.2 taskReference

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

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.

Task t1 references the model model1. Therefore we can conclude that the variable v1 defined in Listing 2.15 targets an element of the model with ID model1. The targeting process itself will be explained in section 2.3.6.1 on page 25.

2.3.5.3 simulationReference

The simulationReference is used to refer to a particular Simulation in a Task. Listing 2.14 shows the reference to a defined simulation for a sample SED-ML document. In the example, both tasks t1 and t2 use the simulation settings defined in simulation1 to run the experiment.

2.3.5.4 dataReference

The dataReference is used to refer to a particular DataGenerator instance from an Output instance. Listing 2.17 shows the reference to a defined data set for a sample SED-ML document.
Listing 2.17: Example for the use of data references in a curve definition

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

2.3.6 Variable

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

![Variable class diagram]

Figure 2.10: The Variable class

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

Table 2.5 shows all attributes and sub-elements for the Variable element as defined by the SED-ML

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid³</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name⁰</td>
<td>page 17</td>
</tr>
<tr>
<td>target⁰</td>
<td>page 25</td>
</tr>
<tr>
<td>symbol⁰</td>
<td>page 26</td>
</tr>
<tr>
<td>taskReference</td>
<td>page 23</td>
</tr>
<tr>
<td>modelReference</td>
<td>page 22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes⁴</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation⁰</td>
<td>page 19</td>
</tr>
</tbody>
</table>

Table 2.5: Attributes and nested elements for Variable. xy⁰ denotes optional elements and attributes.

A variable element must contain a taskReference if it occurs inside a listOfVariables inside a dataGenerator element. A variable element must contain a modelReference if it occurs inside a listOfVariables inside a computeChange element. A variable element appearing within a functionalRange or setValue element must contain a modelReference if and only if it references a model variable.

Listing 2.18 shows the use of the variable element in a SED-ML file as defined by the SED-ML Level 1

<sedML>
Listing 2.18: SED-ML variable definitions inside the computeChange element and inside the dataGenerator element

Listing 2.18 defines a variable v1 (line 7) to compute a change on a model constituent (referenced by the target attribute on computeChange in line 5). The value of v1 corresponds with the value of the targeted model constituent referenced by the target attribute in line 8. The second variable, v2 (line 21), is used inside a dataGenerator. As the variable is time as used in task1, the symbol attribute is used to refer to the SED-ML URI for time (line 21).

2.3.6.1 target

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

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

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

It should be noted that the identifier and names inside the SED-ML document do not have to match the identifiers and names that the model and its constituents carry in the model definition. In listing 2.19, the variable with ID v1 is defined. It is described as the TetR protein. The reference points to a species in the referenced SBML model. The particular species can be identified through its ID in the SBML model, namely PY. However, SED-ML also permits using identical identifiers and names as in the referenced models. The following Listing 2.20 is another valid example for the specification of a variable, but uses the same naming in the variable definition as in the original model (as opposed to Listing 2.19):

Listing 2.20: SED-ML variable definition using the original model identifier and name in SED-ML
The XPath expression used in the `target` attribute unambiguously leads to the particular place in the XML SBML model – the species is to be found in the `sbml` element, and there inside the `listOfSpecies` (Listing 2.21 on the preceding page). Note that while it is possible to write XPath expressions that select multiple nodes within a referenced model, when used within a `target` attribute a single element or attribute must be selected by the expression.

2.3.6.2 symbol

Symbols are predefined, implicit variables that can be called in a SED-ML file by referring to the defined URNs representing that variable’s concept. The notion of implicit variables is explained in Section 2.2.2.3 on page 15.

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

Listing 2.22: SED-ML symbol definition

2.3.7 Parameter

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

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

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

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

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

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

doi:10.2390/biecoll-jib-2015-262
Table 2.6: Attributes and nested elements for parameter. $xy^o$ denotes optional elements and attributes.

2.3.7.1 value

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

2.3.8 ListOf* containers

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

2.3.8.1 ListOfVariables: The variable definition container

SED-ML uses the variable concept to refer to existing entities inside a model. The container for all variables is ListOfVariables (Figure 2.12). It includes all variables that need to be defined to either describe a change in the model by means of mathematical equations (ComputeChange) or to set up a dataGenerator.

Figure 2.12: The SED-ML ListOfVariables container

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

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

Listing 2.24: SED-ML ListOfVariables element
2.3.8.2 listOfParameters: The parameter definition container

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

Listing 2.25 shows the use of the listOfParameters element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The element is optional and may contain zero to many parameters.

```xml
<listOfParameters>
  <parameter id="p1" value= "1" />
  <parameter id= "p2" name= "Kd_2" value= "0.23" />
</listOfParameters>
```

Listing 2.25: SED-ML listOfParameters element

2.3.8.3 listOfModels: The model description container

In order to specify a simulation experiment, the participating models have to be defined. SED-ML uses the listOfModels container for all necessary models (Figure 2.14).

Listing 2.26 shows the use of the listOfModels element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The element is optional and may contain zero to many models. However, if the Level 1 Version 2 document contains one or more Task elements, at least one Model element must be defined to which the Task element refers (c.f. Section 2.3.5.1 on page 22).

```xml
<listOfModels>
  <model id= "m0001" language= "urn:sedml:language:sbml"
    source= "urn:miriam:biomodels.db:BIOMD0000000012" />
  <model id= "m0002" language= "urn:sedml:language:cellml"
    d6613d7e1961b3eff2bb1d3d419e443bb8e794ad/leloup_gonze_goldbeter_1999_a.cellml" />
</listOfModels>
```

Listing 2.26: SED-ML listOfModels element
2.3.8.4 listOfChanges: The change definition container

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

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

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

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

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

2.3.8.5 listOfSimulations: The simulation description container

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

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

```xml
1  <listOfSimulations>
  2  <simulation id="s1"[..]>
  3  [UNIFORM TIMECOURSE DEFINITION]
  4  </simulation>
  5  <simulation id="s2"[..]>
  6  [UNIFORM TIMECOURSE DEFINITION]
  7  </simulation>
  8  </listOfSimulations>
```

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

The listOfSimulations is optional and may contain zero to many simulations. However, if the Level 1 Version 2 document contains one or more Task elements, at least one Simulation element must be defined to which the Task element refers — see section 2.3.5.3 on page 23.

2.3.8.6 listOfAlgorithmParameters: The container for algorithm parameters

The listOfAlgorithmParameters contains the the settings for the simulation algorithm used in a simulation experiment. It may list several instances of the AlgorithmParameter class.

Listing 2.29 on the next page shows the use of the listOfAlgorithmParameters element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema. The listOfAlgorithmParameters is optional and may contain zero to many parameters.
2.3.8.7 listOfTasks: The task specification container

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

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

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

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

The `listOfTasks` is optional and may contain zero to many tasks, each of which is an instance of a subclass of `AbstractTask`. However, if the Level 1 Version 2 document contains a `DataGenerator` element with at least one `Variable` element, at least one `task` must be defined to which variable(s) in the `DataGenerator` element refer — see Section 2.3.5.2 on page 23.

2.3.8.8 listOfDataGenerators: The post-processing container

In SED-ML, all variable- and parameter values that shall be used in the Output class need to be defined as a `dataGenerator` beforehand. The container for those data generators is the `listOfDataGenerators` (Figure 2.18).

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

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

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

The `listOfDataGenerators` is optional and in general may contain zero to many `DataGenerator` elements. However, if the Level 1 Version 2 document contains an `Output` element, at least one `DataGenerator` must
be defined to which the Output element refers - see section 2.3.5.4 on page 23.

2.3.8.9 listOfOutputs: The output specification container

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

![Figure 2.19: The SED-ML listOfOutputs container](image)

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

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

```
1 <listOfOutputs>
2  <report id= "report1">
3   [REPORT DEFINITION FOLLOWING]
4  </report>
5  <plot2D id= "plot1">
6   [2D PLOT DEFINITION FOLLOWING]
7  </plot2D>
8 </listOfOutputs>
```

Listing 2.32: The listOfOutput element
2.4 SED-ML Components

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

2.4.1 Model

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

![Figure 2.20: The SED-ML Model class](image)

Each instance of the Model class has an unambiguous and mandatory id. An additional, optional name may be given to the model. The language may be specified, defining the format the model is encoded in, if such a format exists. Example formats are SBML or CellML.

The Model class refers to the particular model of interest through the source attribute. The restrictions on the model reference are

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

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
<td></td>
</tr>
<tr>
<td>language</td>
<td>page 33</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>page 33</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
<td></td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
<td></td>
</tr>
<tr>
<td>change</td>
<td>page 34</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.7: Attributes and nested elements for model. xy° denotes optional elements and attributes.

Listing 2.33 on the next page shows the use of the model element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.
The above `listOfModels` contains three models: The first model `m0001` is the Repressilator model taken from BioModels Database. The original model is available from `urn:miriam:biomodels.db:BIOMD0000000012`. For the SED-ML simulation, the model might undergo preprocessing, described in the `change` element (lines 5-7). Based on the description of the first model `m0001`, the second model is built. It refers to the model `m0001` in the `source` attribute, that is the modified version of the Repressilator model. `m0002` might then have even further changes applied to it on top of the changes defined in the pre-processing of `m0001`. The third model in the code example above (lines 13-15) is a different model in CellML representation. `m0003` is the model available from the given URL in the `source` attribute. Again, it might have additional pre-processing applied to it before used in the simulation.

### 2.4.1.1 language

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

To help software decide whether or not it supports a SED-ML description file, the information on the model encoding for each referenced model can be provided through the `language` attribute, as the description of a language name and version through an unrestricted `String` is error-prone. A prerequisite for a language to be fully supported by SED-ML is that a formalised language definition, e.g. an XML Schema, is provided online. SED-ML also defines a set of standard URIs to refer to particular language definitions. The list of URNs for languages so far associated with SED-ML is available from the SED-ML web site on [http://sed-ml.org/](http://sed-ml.org/) (Section 2.2.2.2 on page 15). To specify language and version, following the idea of MIRIAM URNs, the SED-ML URN scheme `urn:sedml:language:language name` is used. A model’s language being “SBML Level 2 Version 2” can be referred to, for example, through the URN `urn:sedml:language:sbml.level-2.version-2`.

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

However, the use of the `language` attribute is strongly encouraged for two reasons. Firstly, it helps a user decide whether or not he is able to run the simulation, that is to parse the model referenced in the SED-ML file. Secondly, the language attribute is also needed to decide how to handle the implicit variables in the `Variable` class, as the interpretation of implicit variables depends on the language of the representation format. The concept of implicit variables has been introduced in Section 2.2.2.3 on page 15.

### 2.4.1.2 source

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

The URI should preferably point to a public, consistent location that provides the model description file and follows the proposed URI Scheme. References to curated, open model bases are recommended, such as BioModels Database.
as the BioModels Database. However, any resource registered with MIRIAM resources\(^3\) can easily be referenced. Even without a MIRIAM URN, SED-ML can be used (Section 2.2.2.1 on page 14).

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

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

Listing 2.34: The SED-ML source element, using the URI scheme

The example defines one model m1. urn:miriam:biomodels.db:BIOMD0000000012 defines the source of the model code. The MIRIAM URN can be resolved into the SBML model stored in BioModels Database under ID BIOMD0000000012 using the MIRIAM web service. The resulting URL is http://www.ebi.ac.uk/biomodels-main/BIOMD0000000012.

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

```
<model id="m1" name="repressilator" language="urn:sedml:language:cellml"
  source="http://models.cellml.org/exposure/bba4e39f2c7baa8f51fddf4563e7bde1/aguda_b_1999.cellml">
  <listOfChanges />
</model>
```

Listing 2.35: The SED-ML source element, using a URL

In the example one model is defined. The language of the model is CellML. As the CellML model repository currently does not provide a MIRIAM URI for model reference, the URL pointing to the model code is used to refer to the model. The URL is given in the source attribute.

### 2.4.2 Change

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

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

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

---

\(^3\)http://www.ebi.ac.uk/miriam/main/
Table 2.8 shows all attributes and sub-elements for the `change` element as defined by the SED-ML Level 1 Version 2 XML Schema.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
<th>page or section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>metaid</code></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td><code>id</code></td>
<td></td>
<td>17</td>
</tr>
<tr>
<td><code>name</code></td>
<td></td>
<td>17</td>
</tr>
<tr>
<td><code>target</code></td>
<td></td>
<td>25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
<th>page or section</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>notes</code></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td><code>annotation</code></td>
<td></td>
<td>19</td>
</tr>
</tbody>
</table>

Table 2.8: Attributes and nested elements for `change`. `xy` denotes optional elements and attributes.

Each Change has a `target` attribute that holds a valid XPath expression pointing to the XML element or XML attribute that is to undergo the defined changes. Except for the cases of `ChangeXML` and `RemoveXML`, this XPath expression must always select a single element or attribute within the relevant
model.

2.4.2.1 NewXML

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anyXML</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

2. If it is used as a sub-element of the changeXML element, then the XML it contains is to replace the XML element addressed by the XPath.

Examples are given in the relevant change class definitions.

2.4.2.2 AddXML

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
<tr>
<td>target</td>
<td>page 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
<tr>
<td>newXML</td>
<td>page 36</td>
</tr>
</tbody>
</table>

Table 2.10: Attributes and nested elements for addXML. xy o denotes optional elements and attributes.
An example for a change that adds an additional parameter to a model is given in Listing 2.36.

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

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

The code of the model is changed so that a parameter with ID `V_mT` is added to its list of parameters. The `newXML` element adds an additional XML element to the original model. The element’s name is `parameter` and it is added to the existing parent element `listOfParameters` that is addressed by the XPath expression in the `target` attribute.

### 2.4.2.3 ChangeXML

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

![Figure 2.23: The ChangeXML class](image)

The XPath expression is specified in the required `target` attribute (Section 2.3.6.1 on page 25). The replacement XML content is specified in the `NewXML` class.

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 17</td>
</tr>
<tr>
<td>target</td>
<td>page 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 19</td>
</tr>
<tr>
<td>newXML</td>
<td>page 36</td>
</tr>
</tbody>
</table>

Table 2.11: Attributes and nested elements for changeXML. <sup>*</sup> denotes optional elements and attributes.

An example for a change that adds an additional parameter to a model is given in Listing 2.37.
Listing 2.37: The changeXML element

The code of the model is changed in the way that its parameter with ID $V_{mT}$ is substituted by two other parameters $V_{mT\_1}$ and $V_{mT\_2}$. The target attribute defines that the parameter with ID $V_{mT}$ is to be changed. The newXML element then specifies the XML that is to be exchanged for that parameter.

2.4.2.4 RemoveXML

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

The XPath is specified in the required target attribute.

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
<tr>
<td>target</td>
<td>page 25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
</tbody>
</table>

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

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

Listing 2.38:

```xml
1 <model [..]>
2 <listOfChanges>
3 <changeXML target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='V_mT']" >
4  <newXML>
5   <parameter metaid="metaid_0000010" id="V_mT_1" value="0.7" />
6   <parameter metaid="metaid_0000050" id="V_mT_2" value="4.6" />
7  </newXML>
8 </changeXML>
9 </listOfChanges>
10 </model>
```
Listing 2.38: The removeXML element

The code of the model is changed by deleting the reaction with ID \( V_{mT} \) from the model’s list of reactions.

2.4.2.5 ChangeAttribute

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

![Figure 2.25: The ChangeAttribute class](image)

The ChangeXML class covers the possibilities provided by the ChangeAttribute class. That is, everything that can be expressed by a ChangeAttribute construct can also be expressed by a ChangeXML. However, for the common case of changing an attribute value ChangeAttribute is easier to use, and so it is recommended to use the ChangeAttribute for any changes of an XML attribute’s value, and to use the more general ChangeXML for other cases.

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid(^o)</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name(^o)</td>
<td>page 17</td>
</tr>
<tr>
<td>target</td>
<td>page 25</td>
</tr>
<tr>
<td>newValue</td>
<td>page 39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes(^o)</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation(^o)</td>
<td>page 19</td>
</tr>
</tbody>
</table>

Table 2.13: Attributes and nested elements for ChangeAttribute. \(*\text{\(^o\)}\) denotes optional elements and attributes.

2.4.2.5.1 newValue

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

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

```xml
<model id= "model1" name= "Circadian Chaos" language= "urn:sedml:language:sbml"

source= "urn:miriam:biomodels.db:BIOMD0000000021">

<listOfChanges>

<changeAttribute target= "/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='V_{mT}']/@value"

newValue="0.28"/>

<changeAttribute target= "/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='V_{dT}']/@value"

newValue="4.8"/>

</listOfChanges>
</model>
```

doi:10.2390/biecoll-jib-2015-262
2.4.2.6 ComputeChange

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

The computed new value is described by a mathematical expression using a subset of MathML (see section 2.2.1 on page 13). The computation can use the value of variables from any model defined in the simulation experiment. Those variables need to be defined, and can then be addressed by their ID. A variable used in a ComputeChange must carry a modelReference attribute (page 22) but no taskReference attribute (page 23). To carry out the calculation it may be necessary to introduce additional parameters, that are not defined in any of the models used by the experiment. This is done through the parameter class, and such parameters are thereafter referred to by their ID. Finally, the change itself is specified using an instance of the Math class.

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

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

2.4.2.6.1 Math

The Math element encodes mathematical functions. If used as an element of the ComputeChange class, it computes the change of the element or attribute addressed by the target attribute. Level 1 Version 2 supports the subset of MathML 2.0 shown in section 2.2.1.

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

```xml
1 <model [...]>
2  <computeChange target="/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='sensor']">
3    <listOfVariables>
4      <variable modelReference="model1" id="R" name="regulator"
5        target="/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='regulator']" />
6    </listOfVariables>
7  </computeChange>
8 </model>
```

doi:10.2390/biecoll-jib-2015-262
Table 2.14: Attributes and nested elements for computeChange. xy denotes optional elements and attributes.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
<td></td>
</tr>
<tr>
<td>target</td>
<td>page 25</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
<td></td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
<td></td>
</tr>
<tr>
<td>listOfVariables</td>
<td>page 27</td>
<td></td>
</tr>
<tr>
<td>listOfParameters</td>
<td>page 28</td>
<td></td>
</tr>
<tr>
<td>math</td>
<td>page 40</td>
<td></td>
</tr>
</tbody>
</table>

Listing 2.40: The computeChange element

The example in Listing 2.40 computes a change of the variable sensor of the model model2. To do so, it uses the value of the variable regulator coming from model model1. In addition, the calculation used two additional parameters, the cooperativity n and the sensitivity K. The mathematical expression in the mathML then computes the new initial value of sensor using the equation:

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

2.4.3 Simulation

A simulation is the execution of some defined algorithm(s). Simulations are described differently depending on the type of simulation experiment to be performed (Figure 2.27 on the next page).
Simulation is an abstract class and serves as the container for the different types of simulation experiments. SED-ML Level 1 Version 2 offers the predefined simulation classes UniformTimeCourse, OneStep and SteadyState. Further simulation classes are planned for future versions of SED-ML, including simulation classes for bifurcation analysis. Simulation algorithms used for the execution of a simulation setup are defined in the Algorithm class.

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaId</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
<tr>
<td>algorithm</td>
<td>page 42</td>
</tr>
</tbody>
</table>

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

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

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

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

Two timecourses with uniform range are defined.

2.4.3.1 Algorithm

SED-ML makes use of the KiSAO ontology (Section 2.2.4 on page 16) to refer to a term in the controlled vocabulary identifying the particular simulation algorithm to be used in the simulation.

Each instance of the Simulation class must contain one reference to a simulation algorithm (Figure 2.28 on the following page).
Each instance of the Algorithm class must contain a KiSAO reference to a simulation algorithm. The reference should define the simulation algorithm to be used in the simulation as precisely as possible, and should be defined in the correct syntax, as defined by the regular expression \texttt{KISAO:0-9}\{7\}.

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid*</td>
<td>page 18</td>
</tr>
<tr>
<td>kisaoID</td>
<td>page 16</td>
</tr>
<tr>
<td>sub-elements</td>
<td></td>
</tr>
<tr>
<td>notes*</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation*</td>
<td>page 19</td>
</tr>
<tr>
<td>algorithmParameter*</td>
<td>page 43</td>
</tr>
</tbody>
</table>

Table 2.16: Attributes and nested elements for algorithm. \textit{xy}\* denotes optional elements and attributes.

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

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

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

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

### 2.4.3.2 AlgorithmParameter

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

```
<algorithm kisaoID="KISAO:0000032" >
  <listOfAlgorithmParameters>
    <algorithmParameter kisaoID="KISAO:0000031" value= "23" />
  </listOfAlgorithmParameters>
</algorithm>
```

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
<tr>
<td>initialTime</td>
<td>page 44</td>
</tr>
<tr>
<td>outputStartTime</td>
<td>page 44</td>
</tr>
<tr>
<td>outputEndTime</td>
<td>page 45</td>
</tr>
<tr>
<td>numberOfPoints</td>
<td>page 45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
<tr>
<td>algorithm</td>
<td>page 42</td>
</tr>
</tbody>
</table>

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

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

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

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

2.4.3.3.1 initialTime

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

2.4.3.3.2 outputStartTime

Sometimes a researcher is not interested in simulation results at the start of the simulation (i.e. the initial time). To accommodate this in SED-ML the uniformTimeCourse class uses the attribute outputStartTime of type double. To be valid the outputStartTime cannot be before initialTime. For an example, see Listing 2.44.
2.4.3.3 outputEndTime

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

2.4.3.4 numberOfPoints

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

Just because the output points lie on the regular grid described above, this does not mean that the simulation algorithm has to work with the same step size. Usually the step size the simulator chooses will be adaptive and much smaller than the required output step size. On the other hand a stochastic simulator might not have any new events occurring between two grid points. Nevertheless the simulator has to produce data on this regular grid. For an example, see Listing 2.44.

2.4.3.4 OneStep

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaId</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
<tr>
<td>step</td>
<td>page 46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
<tr>
<td>algorithm</td>
<td>page 42</td>
</tr>
</tbody>
</table>

Table 2.18: Attributes and nested elements for oneStep. $xy^o$ denotes optional elements and attributes.
Listing 2.45 shows the use of the oneStep element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

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

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

2.4.3.4.1 step

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

2.4.3.5 SteadyState

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

![SteadyState class diagram]

Figure 2.31: The SteadyState class

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
<tr>
<td>algorithm</td>
<td>page 42</td>
</tr>
</tbody>
</table>

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

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

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

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid&lt;sup&gt;0&lt;/sup&gt;</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name&lt;sup&gt;0&lt;/sup&gt;</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes&lt;sup&gt;0&lt;/sup&gt;</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation&lt;sup&gt;0&lt;/sup&gt;</td>
<td>page 19</td>
</tr>
</tbody>
</table>

Table 2.20: Attributes and nested elements for abstractTask. <sup>x</sup>y<sup>0</sup> denotes optional elements and attributes.

2.4.5 Task

A task in SED-ML links a model to a certain simulation description via their respective identifiers (Figure 2.33 on the next page), using the modelReference and the simulationReference. The task class receives the id and name attributes from the AbstractTask.
In SED-ML Level 1 Version 2 it is only possible to link one simulation description to one model at a time. However, one can define as many tasks as needed within one experiment description. Please note that the tasks may be executed in any order, as determined by the implementation.

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
<tr>
<td>modelReference</td>
<td>page 22</td>
</tr>
<tr>
<td>simulationReference</td>
<td>page 23</td>
</tr>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
</tbody>
</table>

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

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

Listing 2.47: The task element

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

### 2.4.6 Repeated Task

The repeatedTask class provides a generic looping construct, allowing complex tasks to be represented by composing separate steps. It performs a specified task (or sequence of tasks) multiple times (where the exact number is specified through a range construct), while allowing specific quantities in the model to be altered at each iteration (as defined in the listOfChanges).
The RepeatedTask inherits from AbstractTask. Additionally it has two required attributes range and resetModel as well as child elements listOfRanges, listOfChanges and listOfSubTasks. Of these only listOfChanges is optional.

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

![Diagram of RepeatedTask class]

Figure 2.34: The SED-ML RepeatedTask class

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid*</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name*</td>
<td>page 17</td>
</tr>
<tr>
<td>range</td>
<td>page 50</td>
</tr>
<tr>
<td>resetModel</td>
<td>page 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes*</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation*</td>
<td>page 19</td>
</tr>
<tr>
<td>range</td>
<td>page 50</td>
</tr>
<tr>
<td>change*</td>
<td>page 53</td>
</tr>
<tr>
<td>subTask*</td>
<td>page 53</td>
</tr>
</tbody>
</table>

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

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

```xml
1 <task id="task1" modelReference="model1" simulationReference="simulation1" />
2 <repeatedTask id="task3" resetModel="false" range="current"
3 xmlns:s= 'http://www.sbml.org/sbml/level3/version1/core'
4 >
```
In the example, task1 is repeated three times, each time with a different value for a model parameter w.

### 2.4.6.1 The range attribute

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

### 2.4.6.2 The resetModel attribute

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

### 2.4.6.3 The listOfRanges

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

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

#### 2.4.6.3.1 Range

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

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

For example:

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

Listing 2.49: The UniformRange element

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

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

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

Listing 2.50: The UniformRange element with a logarithmic range.

2.4.6.3.3 VectorRange

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

```xml
<value>1.0</value>
<value>4.0</value>
<value>10.0</value>
```

Figure 2.36: The SED-ML VectorRangeClass class
A **FunctionalRange** constructs a range through calculations that determine the next value based on the value(s) of other range(s) or model variables. In this it is quite similar to the **ComputeChange** element, and shares some of the same child elements. It consists of an optional attribute **range**, two optional elements **listOfVariables** and **listOfParameters**, and a required element **math**.

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

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

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

For example:

```xml
<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>
```

---

**Figure 2.37:** The SED-ML **FunctionalRange** class

**Listing 2.51:** The **VectorRange** element
Listing 2.52: An example of a functionalRange where a parameter \( w \) of model model2 is multiplied by \( index \) each time it is called.

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

```xml
<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></and></piece>
      <otherwise><cn>8</cn></otherwise>
    </piecewise>
  </math>
</functionalRange>
```

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

2.4.6.4 The listOfChanges

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

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

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

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

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

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

Listing 2.54: A setValue element setting \( w \) to the values of the range with id current.

2.4.6.5 The listOfSubTasks

The listOfSubTasks contains one or more subTask elements that specify what simulations are to be performed by the RepeatedTask. All 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 subTask itself has one required attribute task that references the id of another task defined in the
listOfTasks. The order in which to run multiple subTasks should be specified using order attributes
on the subTask elements; if these are omitted the ordering is given by the order of the subTask elements.
In order to establish that one subTask should be carried out before another its order attribute has to
have a lower number (c.f. Listing 2.55).

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

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

### 2.4.7 DataGenerator

The DataGenerator class prepares the raw simulation results for later output (Figure 2.38). 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 2.38: The SED-ML DataGenerator class](image)

Each instance of the DataGenerator class is identifiable within the experiment by its unambiguous id. It
can be further characterised by an optional name. The related Math class contains a mathML expression
for the calculation of the data generator. Mathematical functions available for the specification of data
generators are given in Section 2.2.1 on page 13. Variable and Parameter instances can be used to encode
the mathematical expression.

Table 2.23 shows all attributes and sub-elements for the dataGenerator element as defined by the SED-

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid²</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>math</td>
<td>page 40</td>
</tr>
<tr>
<td>notes³</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation⁴</td>
<td>page 19</td>
</tr>
<tr>
<td>variable⁵</td>
<td>page 24</td>
</tr>
<tr>
<td>parameter⁶</td>
<td>page 26</td>
</tr>
</tbody>
</table>

Table 2.23: Attributes and nested elements for dataGenerator. xy° denotes optional elements and attributes.

Listing 2.56 shows the use of the dataGenerator element in a SED-ML file as defined by the SED-ML

```
1 <listOfDataGenerators>
2  <dataGenerator id= "d1" name= "time">
3   <listOfVariables>
4    <variable id= "time" taskReference= "task1" symbol= "urn: sedml: symbol: time" />
5  </listOfVariables>
6 </dataGenerator>
7 </listOfDataGenerators>
```

Listing 2.56: The use of the dataGenerator element in a SED-ML file as defined by the SED-ML
The `listOfDataGenerator` contains two `dataGenerator` elements. The first one, `d1`, refers to the task definition `t1` (which itself refers to a particular model), and from the corresponding model it reuses the symbol `time`. The second one, `d2`, references a particular species defined in the same model (and referred to via the `taskReference="t1"`). The model species with ID `PX` is reused for the data generator `d2` without further post-processing.

### 2.4.8 Output

The `Output` class describes how the results of a simulation should be presented to the user (Figure 2.39).

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

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

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

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

#### 2.4.8.1 Plot2D

A 2 dimensional plot (Figure 2.40 on the next page) contains a number of `curve` definitions.
Table 2.24: Attributes and nested elements for output. $xy^o$ denotes optional elements and attributes.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid$^p$</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name$^o$</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes$^o$</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation$^o$</td>
<td>page 19</td>
</tr>
<tr>
<td>plot2D$^o$</td>
<td>page 55</td>
</tr>
<tr>
<td>plot3D$^o$</td>
<td>page 57</td>
</tr>
<tr>
<td>report$^o$</td>
<td>page 57</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid$^p$</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name$^o$</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes$^o$</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation$^o$</td>
<td>page 19</td>
</tr>
<tr>
<td>curve</td>
<td>page 58</td>
</tr>
</tbody>
</table>

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

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

```
1 <plot2D>
2  <listOfCurves>
3    <curve>
4      [CURVE DEFINITION]
5    </curve>
6  [FURTHER CURVE DEFINITIONS]
7  </listOfCurves>
8 </plot2D>
```

Listing 2.57: The plot2D element with the nested listOfCurves element

The listing shows the definition of a 2 dimensional plot containing one curve element inside the listOfCurves. The curve definition follows in Section 2.4.9.1 on page 58.
2.4.8.2 Plot3D

A 3 dimensional plot (Figure 2.41) contains a number of surface definitions.

Figure 2.41: The SED-ML Plot3D class

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation&lt;sup&gt;*&lt;/sup&gt;</td>
<td>page 19</td>
</tr>
<tr>
<td>surface</td>
<td>page 60</td>
</tr>
</tbody>
</table>

Table 2.26: Attributes and nested elements for plot3D. xy<sup>*</sup> denotes optional elements and attributes.

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

```xml
1 <plot3D>
2   <listOfSurfaces>
3     <surface>
4       [SURFACE DEFINITION]
5     </surface>
6   </listOfSurfaces>
7 </plot3D>
```

Listing 2.58: The plot3D element with the nested listOfSurfaces element

The example defines a surface for the 3 dimensional plot. The surface definition follows in Section 2.4.9.2 on page 60.

2.4.8.3 The Report class

The Report class defines a data table consisting of several single instances of the DataSet class (Figure 2.42 on the next page). Its output returns the simulation result in actual numbers. The particular columns of the report table are defined by creating an instance of the DataSet class for each column.
Table 2.27 shows all attributes and sub-elements for the \textit{report} element as defined by the SED-ML Level 1 Version 2 XML Schema.

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid(^o)</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name(^o)</td>
<td>page 17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes(^o)</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation(^o)</td>
<td>page 19</td>
</tr>
<tr>
<td>dataSet</td>
<td>page 61</td>
</tr>
</tbody>
</table>

Table 2.27: Attributes and nested elements for \textit{report}. \textit{xy}\(^o\) denotes optional elements and attributes.

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

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

Listing 2.59: The \textit{report} element with the nested \textit{listOfDataSets} element

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

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

2.4.9 Output components

2.4.9.1 Curve

One or more instances of the \textit{Curve} class define a 2D plot. A \textit{curve} needs a data generator reference to refer to the data that will be plotted on the x-axis, using the \textit{xDataReference}. A second data generator reference is needed to refer to the data that will be plotted on the y-axis, using the \textit{yDataReference}.

Table 2.28 on the following page shows all attributes and sub-elements for the \textit{curve} element as defined...
Listing 2.60 shows the use of the *curve* element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.

Listing 2.60: The SED-ML *curve* element, defining the output curve showing the result of simulation for the referenced *dataGenerators*

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

### 2.4.9.1.1 logX

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

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

### 2.4.9.1.2 logY

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

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

### 2.4.9.1.3 xDataReference

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

An example for the definition of a curve is given in Listing 2.60. *xDataReference* is also used in the definition of the x-axis of a *Surface* in a *3D Plot*.

### 2.4.9.1.4 yDataReference

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

An example for the definition of a curve is given in Listing 2.60. *yDataReference* is also used in the
2.4.9.2 Surface

A surface is a three-dimensional figure representing a simulation result (Figure 2.44).

Creating an instance of the Surface class requires the definition of three different axes, that is which data to plot on which axis and in which way. The aforementioned xDataReference and yDataReference attributes define the according data generators for both the x- and y-axis of a surface. In addition, the zDataReference attribute defines the output for the z-axis. All axes might be logarithmic or not. This can be specified through the logX, logY, and the logZ attributes in the according dataReference elements.

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaid</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name</td>
<td>page 17</td>
</tr>
<tr>
<td>logX</td>
<td>page 59</td>
</tr>
<tr>
<td>xDataReference</td>
<td>page 59</td>
</tr>
<tr>
<td>logY</td>
<td>page 59</td>
</tr>
<tr>
<td>yDataReference</td>
<td>page 59</td>
</tr>
<tr>
<td>logZ</td>
<td>page 61</td>
</tr>
<tr>
<td>zDataReference</td>
<td>page 61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation</td>
<td>page 19</td>
</tr>
</tbody>
</table>

Table 2.29: Attributes and nested elements for surface. xy° denotes optional elements and attributes.

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

logZ is a required attribute of the Surface class and defines whether or not the data output on the z-axis is logarithmic. The data type of logZ is boolean. To make the output on the z-axis of a surface plot logarithmic, logZ must be set to “true”, as shown in the sample Listing 2.61.

2.4.9.2.2 zDataReference

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

An example using the zDataReference is given in Listing 2.61 on page 60.

2.4.9.3 DataSet

The DataSet class holds definitions of data to be used in the Report class (Figure 2.45).

![Figure 2.45: The SED-ML DataSet class](image)

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

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

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaId⁰</td>
<td>page 18</td>
</tr>
<tr>
<td>id</td>
<td>page 17</td>
</tr>
<tr>
<td>name⁰</td>
<td>page 17</td>
</tr>
<tr>
<td>dataReference</td>
<td>page 61</td>
</tr>
<tr>
<td>label</td>
<td>page 61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sub-elements</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notes⁰</td>
<td>page 18</td>
</tr>
<tr>
<td>annotation⁰</td>
<td>page 19</td>
</tr>
</tbody>
</table>

Table 2.30: Attributes and nested elements for dataSet. xy⁰ denotes optional elements and attributes.

2.4.9.3.1 label

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

2.4.9.3.2 dataReference

The dataReference attribute contains the ID of a dataGenerator element and as such represents a link to it. The data produced by that particular data generator fills the according data set in the report. Listing 2.62 on the next page shows the use of the dataSet element in a SED-ML file as defined by the SED-ML Level 1 Version 2 XML Schema.
Listing 2.62: The SED-ML `<dataset>` element, defining a data set containing the result of the referenced task.
3. Acknowledgements

The SED-ML specification has been developed with the input of many people. 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
- Andrew Miller

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

Figure A.1 shows the complete UML diagram of the SED-ML. It gives the full picture of all implemented classes (see the XML Schema definition on page 65).

Figure A.1: The SED-ML UML class diagram
B. XML Schema

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

```xml
    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]|([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 name="name" use="optional" type="xs:string"/>
    </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:attribute name="metaid" type="xs:ID" use="optional"/>
        </xs:sequence>
    </xs:complexType>

    <xs:element name="sedML">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="listOfSimulations" minOccurs="0" />
                        <xs:element ref="listOfModels" minOccurs="0" />
                        <xs:element ref="listOfTasks" minOccurs="0" />
                        <xs:element ref="listOfDataGenerators" minOccurs="0" />
                        <xs:element ref="listOfOutputs" minOccurs="0" />
                    </xs:sequence>
                    <xs:attribute name="level" type="xs:decimal" use="required" fixed="1" />
                    <xs:attribute name="version" type="xs:decimal" use="required" fixed="2" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>

    <!-- notes and annotations -->
    <xs:element name="notes">
        <xs:complexType>
            <xs:complexContent>
                <xs:extension base="SEDBase">
                    <xs:sequence>
                        <xs:element ref="content" minOccurs="0" />
                        <xs:element ref="annotation" minOccurs="0" />
                    </xs:sequence>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
    </xs:element>
</xs:schema>
```
<xs:attribute name="target" use="required" type="xs:token" />
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>

<xs:element name="addXML">
<xs:complexType>
<xs:complexContent>
<xs:extension base="SEDBase">
<xs:sequence>
<xs:element ref="newXML" />
</xs:sequence>
<xs:attribute name="target" use="required" type="xs:token" />
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>

<xs:element name="removeXML">
<xs:complexType>
<xs:complexContent>
<xs:extension base="SEDBase">
<xs:attribute name="target" use="required" type="xs:token" />
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:element>

<xs:element name="computeChange" 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>

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

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

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

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

Copyright 2015 The Author(s). Published by Journal of Integrative Bioinformatics. This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License (http://creativecommons.org/licenses/by-nc-nd/3.0/).
Listing B.1: The SED-ML XML Schema definition


C. Examples

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

The current examples make use of models encoded in SBML and CellML. It is important to remember that SED-ML is not restricted to those formats, but can be used with models encoded in many formats, so long as they are serialized in XML. A list of formats known to have been used, at least tentatively, with SED-ML is available on http://sed-ml.org/.
C.1 Le Loup Model (SBML)

The following example provides a SED-ML description for the simulation of the model based on the publication by Leoup, Gonze and Goldbeter “Limit Cycle Models for Circadian Rhythms Based on Transcriptional Regulation in Drosophila and Neurospora” (PubMed ID: 10643740).

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

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

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

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

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

![Figure C.1: The simulation result gained from the simulation description given in Listing C.1](image-url)
<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">
    <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>
  <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='V_dT']/@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 ']">
        <math xmlns= "http://www.w3.org/1998/Math/MathML">
          <ci> v1 </ci>
        </math>
      </variable>
    </listOfVariables>
  </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 ']">
        <math xmlns= "http://www.w3.org/1998/Math/MathML">
          <ci> v1a </ci>
        </math>
      </variable>
    </listOfVariables>
  </dataGenerator>
  <dataGenerator id= "tim2" name= "tim mRNA (changed parameters)">
    <listOfVariables>
      <variable id= "v2" taskReference= "task2" target= "/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='Mt ']">
        <math xmlns= "http://www.w3.org/1998/Math/MathML">
          <ci> v2 </ci>
        </math>
      </variable>
    </listOfVariables>
  </dataGenerator>
  <dataGenerator id= "per_tim2" name= "nuclear PER -TIM complex">
    <listOfVariables>
      <variable id= "v2a" taskReference= "task2" target= "/sbml:sbml/sbml:model/sbml:listOfSpecies/sbml:species[@id='Cn ']">
        <math xmlns= "http://www.w3.org/1998/Math/MathML">
          <ci> v2a </ci>
        </math>
      </variable>
    </listOfVariables>
  </dataGenerator>
</listOfDataGenerators>

<listOfOutputs>
  <plot2D id= "plot1" name= "tim mRNA with Oscillation and Chaos">
    <listOfCurves>
      <curve id= "c1" logX= "false" logY= "false" xDataReference= "time" yDataReference= "tim1" />
      <curve id= "c2" logX= "false" logY= "false" xDataReference= "time" yDataReference= "tim2" />
    </listOfCurves>
  </plot2D>
  <plot2D id= "plot2" name= "tim mRNA limit cycle (Oscillation)">
    <listOfCurves>
      <curve id= "c3" logX= "false" logY= "false" xDataReference= "per_tim1" yDataReference= "tim1" />
    </listOfCurves>
  </plot2D>
  <plot2D id= "plot3" name= "tim mRNA limit cycle (chaos)">
    <listOfCurves>
      <curve id= "c4" logX= "false" logY= "false" xDataReference= "per_tim2" yDataReference= "tim2" />
    </listOfCurves>
  </plot2D>
</listOfOutputs>
</sedML>

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

doi:10.2390/biecoll-jib-2015-262
C.2 Le Loup Model (CellML)

The following example provides a SED-ML description for the simulation of the model based on the publication by Leloup, Gonze and Goldbeter “Limit Cycle Models for Circadian Rhythms Based on Transcriptional Regulation in Drosophila and Neurospora” (PubMed ID: 10643740). Whereas the previous example used SBML to encode the simulation experiment, here the model is taken from the CellML Model Repository [Lloyd et al., 2008].

The original model used in the simulation experiment is referred to using a URL (http://models.cellml.org/workspace/leloup_gonze_goldbeter_1999/@@rawfile/7606a47e222bc3b3d9117bbaa8d2e7246d67eedd/leloup_gonze_goldbeter_1999_a.cellml, ll. 14).

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

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

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

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

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

```
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:math="http://www.w3.org/1998/Math/MathML"
    xsi:schemaLocation="http://sed-ml.org/sed-ml/level1/version2.xsd"
    xmlns:math="http://www.w3.org/1998/Math/MathML"
    xmlns="http://sed-ml.org/sed-ml/level1/version2"
    xsi:schemaLocation="http://sed-ml.org/sed-ml/level1/version2" level="1"
    version="1">
  <notes><p xmlns="http://www.w3.org/1999/xhtml">
  Comparing Limit Cycles and strange attractors for oscillation in Drosophila</p>
  <listOfSimulations>
  <uniformTimeCourse id="simulation1"
  <notes><p xmlns="http://www.w3.org/1999/xhtml">Comparing Limit Cycles and strange attractors for oscillation in Drosophila</p>
```

Copyright 2015 The Author(s). Published by Journal of Integrative Bioinformatics. This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License (http://creativecommons.org/licenses/by-nc-nd/3.0/).

doi:10.2390/biecoll-jib-2015-262
initialTime="0" outputStartTime="0" outputEndTime="380"
<algorithm kisaoID= "KISAO:0000019" />
</uniformTimeCourse>
</listOfSimulations>
</listOfModels>
<listOfTasks>
<task id= "task1" name= "Limit Cycle" modelReference= "model1" simulationReference= "simulation1" />
<task id= "task2" name= "Strange attractors" modelReference= "model2" simulationReference= "simulation1" />
</listOfTasks>
<listOfDataGenerators>
<dataGenerator id= "time" name= "time" >
<listOfVariables>
<variable id= "t" taskReference= "task1" target= "/cellml:model/cellml:component[@name='environment']/cellml:variable[@name='time']" />
</listOfVariables>
</dataGenerator>
<dataGenerator id= "tim1" name= "tim mRNA" >
<listOfVariables>
<variable id= "v0" taskReference= "task1" target= "/cellml:model/cellml:component[@name='MT']/cellml:variable[@name='MT']" />
</listOfVariables>
</dataGenerator>
<dataGenerator id= "per_tim" name= "nuclear PER -TIM complex" >
<listOfVariables>
<variable id= "v1" taskReference= "task1" target= "/cellml:model/cellml:component[@name='CN']/cellml:variable[@name='CN']" />
</listOfVariables>
</dataGenerator>
<dataGenerator id= "tim2" name= "tim mRNA (changed parameters)" >
<listOfVariables>
<variable id= "v2" taskReference= "task2" target= "/cellml:model/cellml:component[@name='MT']/cellml:variable[@name='MT']" />
</listOfVariables>
</dataGenerator>
<dataGenerator id= "per_tim2" name= "nuclear PER -TIM complex" >
<listOfVariables>
<variable id= "v3" taskReference= "task2" target= "/cellml:model/cellml:component[@name='CN']/cellml:variable[@name='CN']" />
</listOfVariables>
</dataGenerator>
</listOfDataGenerators>
</listOfOutputs>
<plot2D id= "plot1" name= "tim mRNA with Oscillation and Chaos" >
<listOfCurves>
<curve id= "c1" logX= "false" logY= "false" xDataReference= "time" yDataReference= "tim1" />
<curve id= "c2" logX= "false" logY= "false" xDataReference= "time" yDataReference= "tim2" />
</listOfCurves>
</plot2D>
<plot2D id= "plot2" name= "tim mRNA limit cycle (Oscillation)" >
<listOfCurves>
<curve id= "c3" logX= "false" logY= "false" xDataReference= "per_tim" yDataReference= "tim1" />
</listOfCurves>
</plot2D>
<plot2D id= "plot3" name= "tim mRNA limit cycle (chaos)" >
<listOfCurves>
<cur...
Listing C.2: LeLoup Model Simulation Description in SED-ML
C.3 The IkappaB-NF-kappaB signaling module (SBML)

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

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

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

As output this simulation description collects four parameters: `TotalNFkBn`, `TotalIkBbeta`, `TotalIkBeps` and `TotalIkBalpha`. These variables are to be plotted against the simulation time and displayed in four separate plots, as shown in Figure C.3.

![Figure C.3: The simulation result gained from the simulation description given in Listing C.3](image-url)

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

```xml
<?xml version="1.0" encoding="utf-8"?>
<sedML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://sed-ml.org/sed-ml/level1/version2.xml" level="1"
  version="1">
  <listOfSimulations>
    <uniformTimeCourse id="simulation1" initialTime="0" outputStartTime="0" outputEndTime="2500"
    numberOfPoints="1000">
      <algorithm kisaoID="KISAO:0000019"/>
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
    <model id="model1" language="urn:sedml:language:sbml" source="urn:miriam:biomodels.db:BIOMD0000000140"/>
  </listOfModels>
  <listOfTasks>
    <task id="task1" modelReference="model1" simulationReferences="simulation1"/>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id="time" name="time">
      <listOfSimulations>
        <uniformTimeCourse id="simulation1" initialTime="0" outputStartTime="0" outputEndTime="2500"
        numberOfPoints="1000"/>
      </listOfSimulations>
    </dataGenerator>
  </listOfDataGenerators>
</sedML>
```
Listing C.3: IkappaB-NF-kappaB signaling Model Simulation Description in SED-ML
C.4 Examples for Simulation Experiments involving repeatedTasks

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

C.4.1 One dimensional steady state parameter scan

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

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

Figure C.4: The simulation result gained from the simulation description given in Listing C.4
Listing C.4: SED-ML document implementing the one dimensional steady state parameter scan

C.4.2 Perturbing a Simulation

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

Note: In the example below we use a functionalRange, although the same result could also be achieved using the setValue element directly.
Figure C.5: The simulation result gained from the simulation description given in Listing C.5
C.4.3 Repeated Stochastic Simulation

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

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

Listing C.5: SED-ML document implementing the perturbation experiment
Figure C.6: The simulation result gained from the simulation description given in Listing C.6
Listing C.6: SED-ML document implementing repeated stochastic runs

C.4.4 One dimensional time course parameter scan

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

Here one repeatedTask runs repeated uniform time course simulations (performing a deterministic simulation run) after each run the parameter value is changed.
Figure C.7: The simulation result gained from the simulation description given in Listing C.7

```xml
<?xml version="1.0" encoding="utf-8"?>
<! -- Written by libSedML v1.1.4992.38982 see http://libsedml.sf.net -->
  <listOfSimulations>
    <uniformTimeCourse id= "timecourse1" initialTime= "0" outputStartTime= "0" outputEndTime= "20" numberOfPoints= "1000">
      <algorithm kisaoID= "KISAO:0000019" />
    </uniformTimeCourse>
  </listOfSimulations>
  <listOfModels>
  </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 ']">
          "current"
        </setValue>
      </listOfChanges>
    </repeatedTask>
  </listOfTasks>
  <listOfDataGenerators>
    <dataGenerator id= "time1" name= "time">
      <listOfVariables>
        <variable id= "time" name= "time" taskReference= "task1" target= "time" />
      </listOfVariables>
      <math xmlns= "http://www.w3.org/1998/Math/MathML">
        "time"
      </math>
    </dataGenerator>
    <dataGenerator id= "J0_v0_1" name= "J0_v0">
      <variable id= "J0_v0" name= "J0_v0" taskReference= "task1" target= "/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J0_v0 ']">
        "current"
      </variable>
      <math xmlns= "http://www.w3.org/1998/Math/MathML">
        "J0_v0"
      </math>
    </dataGenerator>
    <dataGenerator id= "S1_1" name= "S1">
      <math xmlns= "http://www.w3.org/1998/Math/MathML">
        "S1"
      </math>
    </dataGenerator>
    <dataGenerator id= "S2_1" name= "S2">
      <math xmlns= "http://www.w3.org/1998/Math/MathML">
        "S2"
      </math>
    </dataGenerator>
  </listOfDataGenerators>
</sedML>
```
C.4.5 Two dimensional steady state parameter scan

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

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

![Steady State Scan (Boris 2D)](image)

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

**Listing C.7:** SED-ML document implementing the one dimensional time course parameter scan
<listOfModels>
  <model id= "model1" language= "urn:sedml:language:sbml" source= "E:\Users\fbergmann\Documents\sbml models\borisejb.xml" />
</listOfModels>

<listOfTasks>
  <task id= "task0" modelReference= "model1" simulationReference= "steady1" />
  <repeatedTask id= "task1" resetModel= "false" range= "current" >
    <listOfRanges>
      <vectorRange id= "current" >
        <value>1</value>
        <value>5</value>
        <value>10</value>
        <value>50</value>
        <value>60</value>
        <value>70</value>
        <value>80</value>
        <value>90</value>
        <value>100</value>
      </vectorRange>
    </listOfRanges>
    <listOfChanges>
      <setValue target= "/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J1_KK2 ']
        range= "current" modelReference= "model1" >
        <math xmlns= "http://www.w3.org/1998/Math/MathML" >
          <ci> current </ci>
        </math>
      </setValue>
    </listOfChanges>
    <listOfSubTasks>
      <subTask order= "1" task= "task2" />
    </listOfSubTasks>
  </repeatedTask>
  <repeatedTask id= "task2" resetModel= "false" range= "current1" >
    <listOfRanges>
      <uniformRange id= "current1" start= "1" end= "40" numberOfPoints= "100" type= "linear" >
      </uniformRange>
    </listOfRanges>
    <listOfChanges>
      <setValue target= "/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id='J4_KK5 ']
        range= "current1" modelReference= "model1" >
        <math xmlns= "http://www.w3.org/1998/Math/MathML" >
          <ci> current1 </ci>
        </math>
      </setValue>
    </listOfChanges>
    <listOfSubTasks>
      <subTask order= "1" task= "task0" />
    </listOfSubTasks>
  </repeatedTask>
</listOfTasks>

<listOfDataGenerators>
  <dataGenerator id= "J4_KK5_1" name= "J4_KK5" >
    <listOfVariables>
      <variable id= "J4_KK5" name= "J4_KK5" taskReference= "task1" target= "/sbml:sbml:sbml:model/sbml:listOfParameters/sbml:parameter[@id='J4_KK5 ']
    </listOfVariables>
    <math xmlns= "http://www.w3.org/1998/Math/MathML" >
      <ci> J4_KK5 </ci>
    </math>
  </dataGenerator>
  <dataGenerator id= "J1_KK2_1" name= "J1_KK2" >
    <listOfVariables>
      <variable id= "J1_KK2" name= "J1_KK2" taskReference= "task1" target= "/sbml:sbml:sbml:model/sbml:listOfParameters/sbml:parameter[@id='J1_KK2 ']
    </listOfVariables>
    <math xmlns= "http://www.w3.org/1998/Math/MathML" >
      <ci> J1_KK2 </ci>
    </math>
  </dataGenerator>
  <dataGenerator id= "MKK_1" name= "MKK" >
    <listOfVariables>
      <variable id= "MKK" name= "MKK" taskReference= "task1" target= "/sbml:sbml:sbml:model/sbml:listOfSpecies/sbml:species[@id='MKK ']
    </listOfVariables>
    <math xmlns= "http://www.w3.org/1998/Math/MathML" >
      <ci> MKK </ci>
    </math>
  </dataGenerator>
  <dataGenerator id= "MKK_P_1" name= "MKK_P" >
    <listOfVariables>
      <variable id= "MKK_P" name= "MKK_P" taskReference= "task1" target= "/sbml:sbml:sbml:model/sbml:listOfSpecies/sbml:species[@id='MKK_P ']
    </listOfVariables>
    <math xmlns= "http://www.w3.org/1998/Math/MathML" >
      <ci> MKK_P </ci>
    </math>
  </dataGenerator>
</listOfDataGenerators>

<listOfOutputs>
  <plot2D id= "plot1" name= "Steady State Scan (Boris 2D)" />
</listOfOutputs>
Listing C.8: SED-ML document implementing the one dimensional steady state parameter scan
D. The COMBINE archive

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

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

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


doi:10.2390/biecoll-jib-2015-262