SED-ML Script Language

Editing / Creating SED-ML descriptions

Frank T. Bergmann

7/9/2011

This document describes a human readable alternative to writing SED-ML descriptions.

About this Document

The Simulation Experiment Description Markup Language (SED-ML) is a XML format describing simulation experiments, so that they can be easily exchanged independent of software tools that created them. This document describes the SED-ML Script, a Python based shorthand that makes it easy to create / edit SED-ML documents.

Table of Contents

Functions

The following describes the list of functions available in SED-ML Script.

AddTimeCourseSimulation(id, algorithm, startTime, initialTime, endTime, numPoints)

Parameters:

id	The id to give the new simulation experiment
algorithm	A KISAO identifier that describes the simulation algorithm used
startTime	A double representing the start time for the simulation
initialTime	A double representing the time when to start collecting data
endTime	The end time, or the time when to stop the simulation
numPoints	An integer value equal to the number of data points to collect between the initialTime and the endTime

Example:

For example, the following describes a simulation 'simulation1' carried out with the CVODE ode solver to simulate from time 0 to 500, collecting 1000 output points.

```
AddTimeCourseSimulation('simulation1', 'KISAO:0000019', 0, 0, 500, 1000)
```

AddModel(modelId, source, language)

Parameters:

modelId	The id to give the new model
source	The source where to find the model, this can be a reference to a local file, a url or a Miriam uri, or an id of another model
language	The language of the model (defaults to urn:sedml:language:sbml)

Example:

A model defined from a web address:

```
AddModel('model1', 'http://www.ebi.ac.uk/biomodels/models-
main/publ/BIOMD0000000012/BIOMD0000000012.xml', 'urn:sedml:language:sbml')
```

Or a model by Miriam Uri (using the default language):

```
AddModel('model1', 'urn:miriam:biomodels.db:BIOMD0000000139')
```

AddTask(taskId, simulationId, modelId)

Parameters:

modelId	The id to give the new task
simulationId	The simulation that should be applied in this task
modelId	The model to use in this task

Example:

The following applies 'simulation1' to 'model1':

```
AddTask('task1', 'simulation1', 'model1')
```

Changing the model

Often it is convenient to change the initial model, by applying parameter changes or the like.

AddChange(modelId, xpathTarget, newValue)

Parameters:

modelId	The id of the model to change
xpathTarget	An xpath expression to the attribute to change
newValue	The new value to assign to the variable

Example:

The following changes the value attribute of the SBML parameter with id 'ps_0' to 1.3e-5:

```
AddChange('model2',
'/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id="ps_0"]/@value
', '1.3e-5')
```

AddParameterChange(modelId, sbmlId, newValue)

Note:

This is a convenience method available only for SBML.

Parameters:

modelId	The id of the model for which to change a parameter value
sbmlId	The sbml id of the SBML component to change
newValue	The new value to set

Example:

The following changes the value of the SBML parameter with id 'ps_0' to 1.3e-5:

```
AddParameterChange('model2', 'ps_0', '1.3e-5')
```

AddXMLChange(modelId, xpathExpression, newXML)

Note:

This is a convenience method available only for SBML.

Parameters:

modelId	The id of the model to change
xpathExpression	The xpathExpression to the element on which to apply the change
newXML	The snippet of xml to apply

Example:

The following changes the xml for reaction J2:

```
AddXMLChange('model3',
'/sbml:sbml/sbml:model/sbml:listOfReactions/sbml:reaction[@id="J2"]',
'<reaction id="J2" reversible="false" ><listOfReactants><speciesReference
species="S1" stoichiometry="1"
/></listOfReactants><listOfProducts><speciesReference species="S2"
stoichiometry="1" /></listOfProducts><kineticLaw><math
xmlns="http://www.w3.org/1998/Math/MathML"><apply><times /><apply><minus
/><apply><times /><ci>J2_k1</ci><ci>S1</ci></apply><apply><times
/><ci>J2_k_1</ci><ci>S2</ci></apply></apply><apply><plus /><cn type="integer">
1 </cn><apply><times /><ci>J2_c</ci><apply><power /><ci> S2 </ci><ci> J2_q
</ci></apply></apply></apply></apply></math></kineticLaw></reaction><reaction
id="J3" reversible="false" xmlns="http://www.biomodels.net/sed-
ml"><listOfReactants><speciesReference species="S2" stoichiometry="1"
/></listOfReactants><listOfProducts><speciesReference species="X2"
stoichiometry="1" /></listOfProducts><kineticLaw><math
xmlns="http://www.w3.org/1998/Math/MathML"><apply><times
/><ci>J3_k2</ci><ci>S2</ci></apply></math></kineticLaw></reaction>')
```

AddComputeChange(modelId, xpathTarget, infix, variableList)

Parameters:

modelId	Of the model to change
xpathTarget	An xpath expression targeting the attribute to change
Infix	An infix expression containing the formula
variableList	A list of variables of different models that are used in the expression.

Example:

The following changes the value of the parameter J3_k2 of model2 to half of the value of J3_k2 of model1.

```
AddComputeChange('model2',
’/sbml:sbml/sbml:model/sbml:listOfParameters/sbml:parameter[@id="J3_k2"]/@valu
e',
'J3_k2 / 2',
[['J3_k2', 'model1']])
```

Defining the Output

The next methods define reports, 2D or 3D plots.

AddColumn(dataId, variableList)

Parameters:

dataId	The id to give the new data generator
variableList	The list of variables to use for this data generator. The list can consist of one or more variable references and an infix expression.

Example:

The simplest case would be a data generator ‘ds0’ defined by simple variable reference (as ‘var0’ representing ‘X’ from ‘task1’:

```
AddColumn('ds0', [['var0', 'task1', 'X']])
```

By adding an infix expression to the list, it is possible to normalize the value. Here the value is simply halved:

```
AddColumn('ds0', [['var0', 'task1', 'X'], 'var0/2'])
```

To add multiple variables:

```
AddColumn('ds0', [['var0', 'task1', 'X'], ['var1', 'task1', 'Y'], 'var0 +
var1'])
```

AddPlot(plotId, plotName, listOfCurves)

Parameters:

plotId	The id to give the new plot
plotName	The plot title
listOfCurves	A list of pairs of values defining the traces to draw

Example:

The following creates a plot with three traces, time vs. X, time vs. Y and time vs. Z:

```
AddPlot('plot1', 'Timecourse 1', [['time', 'X'], ['time', 'Y'], ['time',
'Z']]);
```

AddPlot3D(plotId, plotName, listOfSurfaces)

Parameters:

plotId	The id to give the new 3d plot
plotName	The plot title
listOfSurfaces	A list of triples of values defining the surfaces to draw.

Example:

The following defines a surface trace with components 'X', 'Y' and 'Z':

```
AddPlot3D('plot2', 'Surface Demo', [['X', 'Y', 'Z']]);
```

AddReport(reportId, reportName, listOfColumns)

Parameters:

reportId	The id for the report
reportName	The report title
listOfColumns	A list of data generator ids, that represent the columns of the report

Example:

The following defines a report titled 'Report 1' with two columns: 'time' and 'Node0':

```
AddReport('report1', 'Report 1', ['time', 'Node0']);
```

Additional Functions

The functions below are not part of the SED-ML standard, but are useful nonetheless.

SetTolerances(simId, aTol, relTol, maxSteps)

Parameters:

simId	The simulation for which to set the tolerances
aTol	Absolute tolerance to set for the integrator
relTol	Relative tolerance to set for the integrator
maxSteps	Maximum number of steps to use for the simulator

Example:

To set the tolerances of 'sim1' to 1e-6, 1e-3 with 1000 steps one would use:

```
SetTolerances('sim1', 1e-6, 1e-3, 1000)
```

SetInitialStep(simId, initialStep)

Parameters:

simId	The simulation for which to specify the initial step
initalStep	The initial step the integrator should take

Example:

```
SetInitialStep('sim1', 1e-10)
```

SetMaxStep(simId, maxStep)

Parameters:

simId	The simulation for which to specify the maximum step
maxStep	The maximum step size the integrator is allowed to take.

Example:

```
SetMaxStep('sim1', 1e-3)
```

Example Documents

Lorenz Attractor

```
AddTimeCourseSimulation('sim1', 'KISAO:0000019', 0, 0, 100, 1000)


AddModel('model1', 'c:/sbml models/lorenz.xml')


AddTask('task1', 'sim1', 'model1')


AddColumn('time', [['vTime', 'task1', 'time']])
AddColumn('X', [['vX', 'task1', 'X']])
AddColumn('Y', [['vY', 'task1', 'Y']])
AddColumn('Z', [['vZ', 'task1', 'Z']])


AddPlot('plot1', 'X, Y, Z', [['time', 'X'], ['time', 'Y'], ['time', 'Z']]);
AddPlot('plot2', 'X', [['time', 'X']]);
AddPlot('plot3', 'Y', [['time', 'Y']]);
AddPlot('plot4', 'Z', [['time', 'Z']]);
AddPlot('plot5', 'X vs Y', [['X', 'Y']]);
AddPlot('plot6', 'X vs Z', [['X', 'Z']]);
AddPlot('plot7', 'Y vs Z', [['Y', 'Z']]);


AddPlot3D('plot8', '3D Plot', [['X', 'Y', 'Z']]);
AddReport('report', 'Report', ['time', 'X', 'Y', 'Z']);
```

When executed this results in 7 plots, 1 3d plot and a report.