A Simple Nested Simulation for SED-ML

Frank T. Bergmann (fbergman@u.washington.edu)

About this document

This document describes several proposals that help to broaden the features currently included with the SED-ML [1]. Currently, SED-ML effectively describes the exchange of time course simulation experiments. Through suggestions made at the Super Hackathon in New Zealand[1] last year, this general uniform time course simulation was extended, by applying different ranges to simulation experiments (Figure 1).

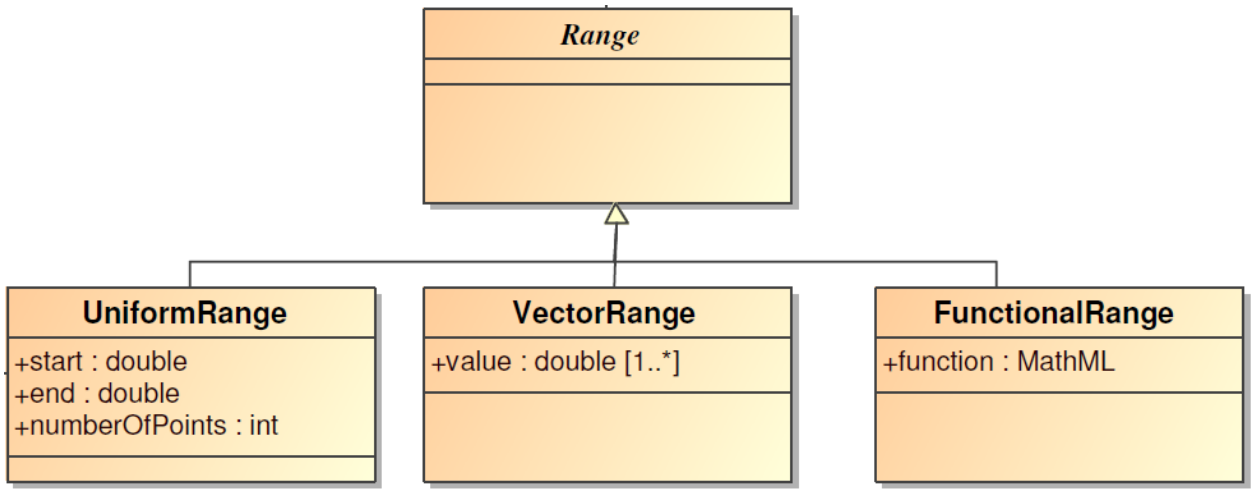

Figure 1: Extending Simulations Through Ranges (snippet from current proposed SED-ML object model[2])

However, by directly applying these ranges to the *TimeCourse* simulation element (and other future simulation types), it will be arguably harder for the community to implement this standard. Currently available simulation tools do not have this functionality. Moreover, a custom implementation will be necessary for each simulation experiment encoded this way. Here, an alternative will be presented that will allow for the same functionality as the current proposal and, perhaps even more important, make it easy for developers to implement. It will also allow for the community to implement novel simulation experiments.

[1] http://www.cellml.org/community/events/workshop/2009
[2] http://sed-ml.svn.sourceforge.net/viewvc/sed-ml/sed-ml/documents/sed-om/sedom-tmp.pdf

The Problem

So what makes the current proposal hard to implement? The problem is not so much the actual Range object. The problem lies in the fact, that the Range object is directly applied to a *Simulation* object, such as *TimeCourse*. In the following, the *TimeCourse* simulation example will be taken and the various ranges applied.

- **Uniform Range**: A *TimeCourse* object with a uniform range maps precisely to the agreed upon *UniformTimeCourse* object. So there is no problem to implement support for this. Moreover, this is precisely what is implemented in most available software today, so it has excellent chances of being reliably exchanged.
- **Vector Range**: One example here would be to get the *TimeCourse* at arbitrary time points, say 0, 0.5, 0.6 and 20. This is completely different from before. Support for this is not available in most simulators today.
- **Functional Range**: A functional range might not make much sense with the *TimeCourse* object. However, for sake of argument, say that someone would like to sample a log function in a certain range. (Note: The Functional Range, should probably have a dependency to Uniform Range, in order to specify where to evaluate it.) Again, support for this is not available in simulators today. However, if a Vector Range is implemented, then this functional range can be easily mapped to the Vector Range by utility libraries like libSedML[3].

Similarly, support for each of the three range types would have to be manually provided for other simulation types, such as the suggested steady state parameter scan. As soon as a *TimeCourse* parameter scan simulation task is added, this one will have to be implemented multiple times as well.

Proposal 1: A Simple, Nested Simulation Experiment

Instead of forcing the community to adopt all these individual simulation experiments, we should try to make it as easy as possible. Otherwise even partial adoption will take years. Moreover, we will have to constantly define new simulation experiments! Instead, let us try and develop a set of primitives, and nest these together, in order to form simulation experiments.

The Primitives

For the start just defining three primitives, will allow us to define nearly any time course or steady state based simulation experiment. Including parameter scans and pulse experiments:

- **OneStep**: This primitive carries out one integration step of the integrator defined through the KISAO identifier. We need two parameters, the current time, and a step.
- **SteadyState**: This primitives tries to bring the model to steady state. There probably won't be any parameters necessary here (unless we start defining tolerances)

[3] http://libsedml.sourceforge.net

- **SetValue**: This primitive targets a model variable through a XPath expression, and assigns it a new value through an arbitrary MathML expression, similar to the data generators this should allow for the referencing of parameter values and model variables.

Both, **OneStep** and **SteadyState**, would inherit from *Simulation* and thus, in their own right present valid simulation experiments, the only difference is, that they would only yield one output row.

The Nested Experiment

What brings the individual elements together is the definition of a nested experiment. A nested experiment will always refer to:

- A task object, that defines the model and simulation experiment that will be called repeatedly
- A range object, that defines how often the simulation experiment will be repeated.
- A **SetValue** object, that would determine a value to be changed after each run. We could make it convention, that if this is not defined, then the models time parameter is changed.

We could also add a couple of convenience flags, such as whether the model is reset (i.e.: the models initial conditions have to be reapplied) after each run.

I would also like to make it convention, that the first output point should be the models initial values.

Finally, since the Nested Experiment would inherit from *Simulation* itself, it would even be possible to define nested, nested experiments. Here I think of 2D parameter scans for example!

Conclusion

This simple, nested simulation experiment, would allow us to circumvent the problems introduced through the addition of the ranges construct. Moreover, it will provide a multitude of simulation experiments to be used in SED-ML, without the need of re-implementing them in software in each case. No longer has the specification to be constantly changed by adding new simulation experiments. By adding new primitives this proposal is also extendable.

Bibliography

1. Köhn, D. and N. Le Novère. *SED-ML - An XML Format for the Implementation of the MIASE Guidelines*. in *Proceedings of the 6th International Conference on Computational Methods in Systems Biology*. 2008. Rostock, Germany: Springer, ISBN: 978-3-540-88561-0.