Explicit modelling of DEVS experiments in AToMPM

Timmy Nelen

Abstract

This paper proposes a graphical language to allow lesser-techniaclly skilled developers to make changes in simulations to get desired results. These constructs then get transformed into a simulation to be ran in a DEVS simulator.

Keywords: AtomPM, DEVS, SED-ML, EGL, Transformations

1. Introduction

When it comes to running experimental simulations, we have a wide array of options, frameworks and languages to work in. In this report we will discuss how we can expand the existing framework that allows us to model DEVS (textual simulation language) in AtomPM (a visual modelling language). AtomPM allows us to create our own Meta-Models, and making use of a pre-made DEVS Meta-Model we are able to create our own version of a language wherein we can create visual representations of experiments, in a format that allows us to export it to a DEVS-framework (in this case, PyPDEVS).

Section 2 introduces us to our research material, Section 3 will talk about the SED-ML paper, Section 4 will briefly talk about DEVS, section 5 will discuss possible additions we can do, and Section 6 will outline the approach we will take to make this all possible.

2. Research material

The reading part of this report is based off of the following 2 papers:

1. "Reproducible computational biology experiments with SED-ML - The Simulation Experiment Description Markup Language" (Waltemath et al.)
2. "Debugging Parallel DEVS" (Van Mierlo, Van Tendeloo et al.)

in combination with the PyPDEVS documentation.

The first paper discusses SED-ML, short for Simulation Experiment Description Mark-up Language. This is a computer-readable exchange format to report information needed to reproduce simulation experiments. This paper talks about how SED-ML encodes all the needed information into XML, making it computer-readable and easy to use. It also provides methods to modify experiments (via parameters or the actual model structure of the experiment).

The second paper is a technical report on PDEVS, mainly used for background information.

3. SED-ML

SED-ML is a technology created specifically for facilitating the reproduction of simulation experiments. Often researchers spend a lot of time running multiple, slightly different, experiments on a set of data, without an easy way to manipulate the experiment in question, apart from actually editing the code.

Using the SED-ML, we can circumvent this problem by providing the researcher with an XML-file (easily computer-readable and -editable) that describes the experiment. In here we can find several parameters and options for the experiment so that we can run the experiment in several different ways without having to change the actual code.

Also included are ways to include combinations of models, output generation and grouping of output.

A visual representation of SED-ML is shown below.

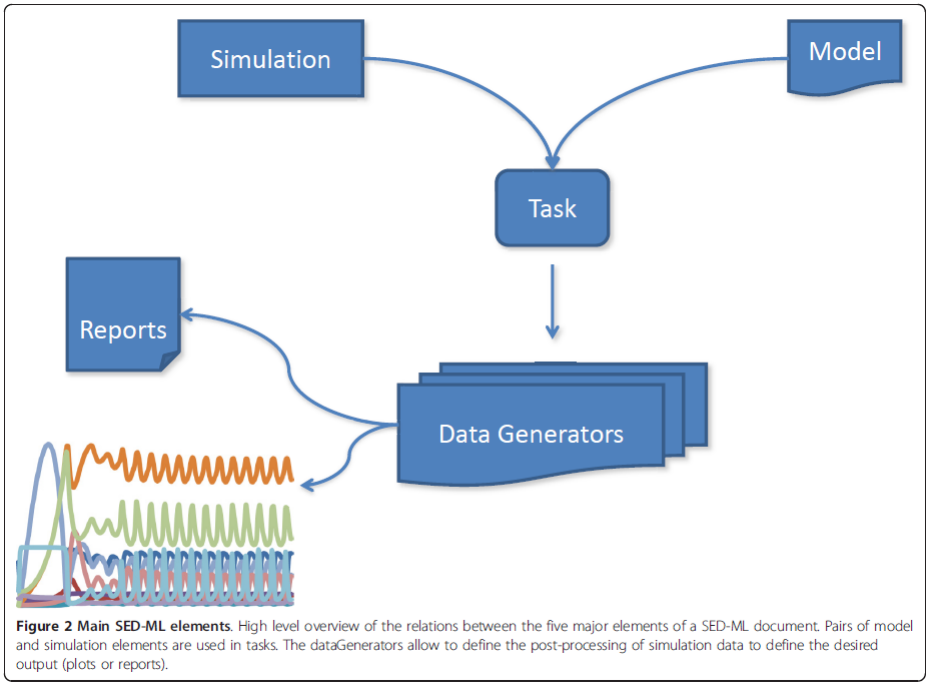
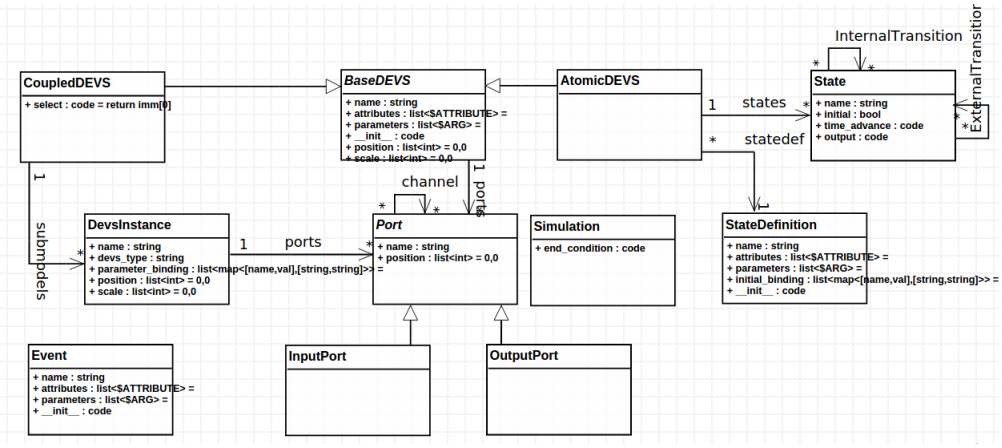


Figure 2 Main SED-ML elements. High level overview of the relations between the five major elements of a SED-ML document. Pairs of model and simulation elements are used in tasks. The dataGenerators allow to define the post-processing of simulation data to define the desired output (plots or reports).

The way this paper is related to the project is that we are also trying to find a way to describe experiments (here in a visual way, as opposed to textual XML), and give the researcher ways to modify the experiment to his likings. For this we will have to provide visual elements to modify the parameters of the experiment and provide additional functionality, which will get processed and included in the resulting DEVS code.

4. DEVS

A good introduction to DEVS can be found in (Van Mierlo, Van Tendeloo, Mustafiz, Barroca , 2014, p. 4-6). DEVS is implemented in AtomPM as follows:



As we can see, a basic DEVS system can be modelled using this Meta-Model. Our goal is to expand this visual notation.

5. Additions

Most of the basic functionality is already implemented in the basic DEVS framework in AtomPM but there are many other functions which we are able to implement, but this paper mainly focusses on customizing parameters in the experiment.

5.1. Custom parameters

PyPDEVS supplies an easy-to-use interface to customize settings to your desire, for example:

setDrawModel	Disable model drawing if we aren't interested in its visual representation
setLogging	Set a destination file for the logs to be written to (VCD and XML format)
setTerminationTime	Set a maximum time for the simulation to run
setTerminationCondition	Set a custom function as a termination condition
setVerbose	Allow verbose output in a file

These are some of the possiblities that are offered within PyPDEVS which can be implemented visually to allow us more control over the actual experiment. The logging functionality is mainly used for analysis afterwards.

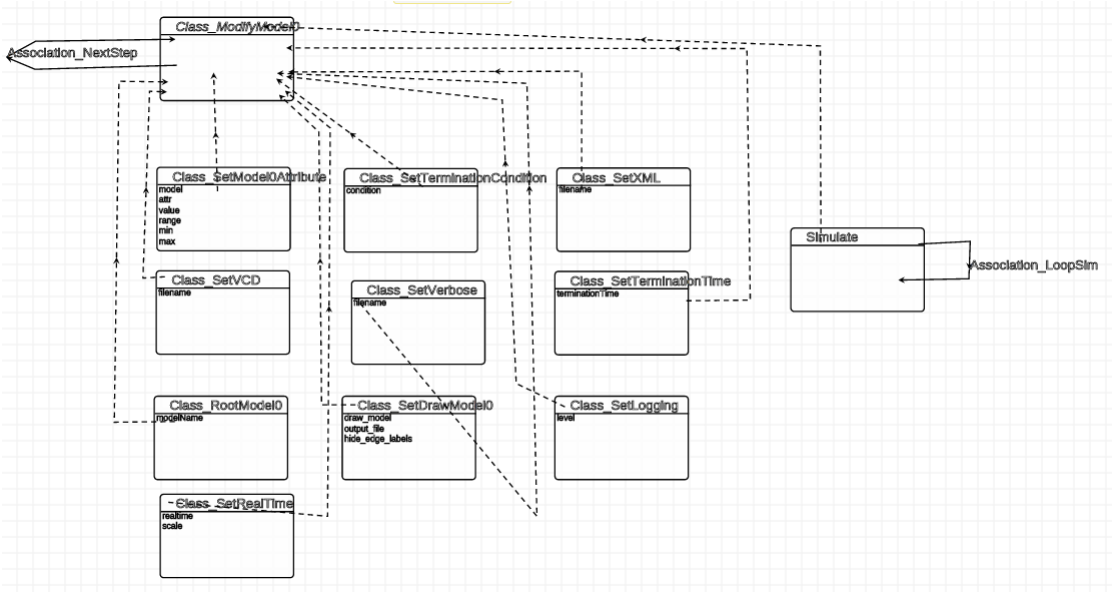
Using a custom logging system we can set up our experiment to log meaningful data to a file, in a GNUPlot-readable format. This allows us to create plots of the system. We can extend this as far as we want to get as much detail as we want in our plots. When it comes to optimizing experiments we can opt for limited logging, as the intermediate values aren't of interest, but for manual analysis we may want very detailed plots. To do this in practice we will need to create a cost-function to determine the performance of our experiment at set intervals.

6. Approach

The approach we will take for this project is in 4 distinct steps:

6.1. Abstract modelling

First of all, we need an abstract syntax to represent our elements in. This is what this syntax looks like:

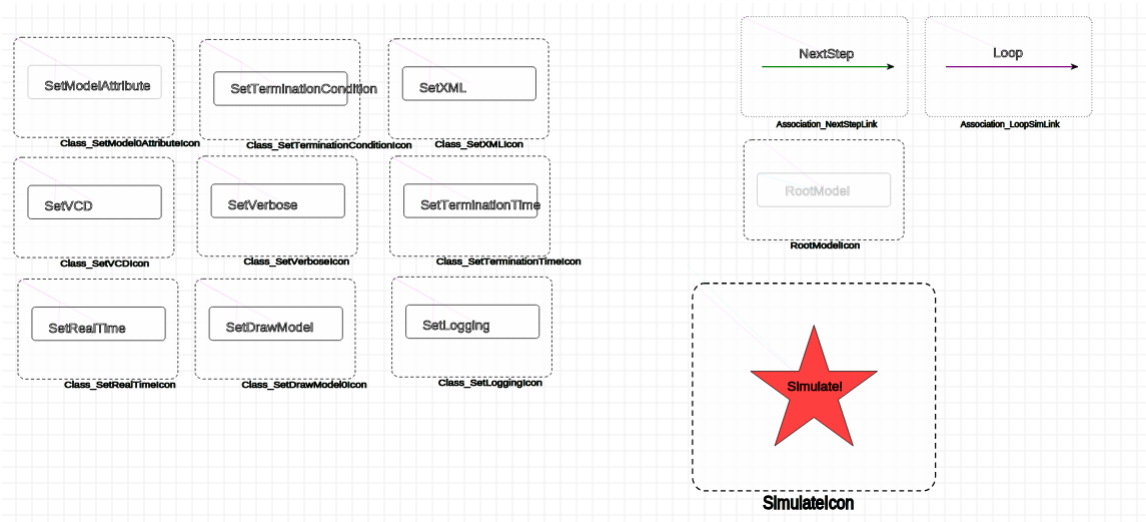


Every building block is a subtype of a very general block for easy computation and linking.

6.2. Visual modelling

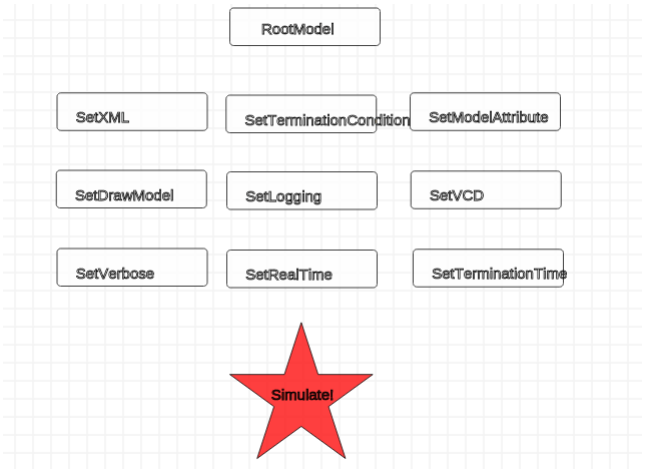
We aim to create a separate visual language to be used in combination with the existing DEVS framework in AtomPM. With this we mean that we will be adding several new elements that can all be configured separately. As this will be a separate model, it will not interfere with the existing DEVS standard and can be treated as an add-on.

The implementation of this looks as follows



For this project I've kept the basis nice and simple. There are several blocks that depict actions can you undertake on the data, a few connectors (which aren't required), a "root" object to supply the models name, and an element to start the actual simulation.

An example of a simple simulation looks like this:



As you can see, connections aren't strictly needed. As these blocks are only there to change some parameters, the order is not important, and everything else gets handled in the back-end.

6.3. Transformation to a usable format

To convert this visual interface into a format that PyPDEVS can use (e.g., Python code), the MetaDepth-exporter was used. This transforms the visual elements into plain text, which we can then convert into Python code using the EGL language, giving us pure Python instructions. The EGL code scans around for all available blocks, converts them one by one into Python instructions and puts the appropriate imports and static commands in the file so we have a complete program.

7. Bibliography

Van Mierlo, Van Tendeloo, Musafiz, Barroca 2014. Debugging Parallel DEVS

Waltemath et al. 2014. Reproducible computational biology experiments with SED-ML - The Simulation Experiment Description Markup Language