Using Modelica Models for Complex Virtual Experimentation with the Tornado Kernel

Filip H.A. Claeys
Department of Applied Mathematics,
Biometrics and Process Control (BIOMATH)
Ghent University
Coupure Links 653
B-9000 Gent
Belgium
E-mail: fc@biomath.ugent.be

Peter Fritzson
Programming Environments
Laboratory (PELAB)
Linköping University
Campus Valla
SE-581 83 Linköping
Sweden
E-mail: petfr@ida.liu.se

Peter A. Vanrolleghem
model*EAU*
Département de génie civil
Université Laval
Pavillon Pouliot
Québec, G1K 7P4
QC, Canada
E-mail: peter@modelEAU.org

Abstract

Tornado is a software kernel for virtual experimentation on the basis of ODE/DAE models. Recently, a model compiler has been developed that converts flat Modelica code to executable models suitable for use with the Tornado kernel. As a result, a subset of Modelica models can now be used for tasks such as parameter estimation, scenario analysis, Monte Carlo simulation, sensitivity analysis and steady-state analysis. The inherent computational complexity of the virtual experiment types implemented by Tornado can be efficiently handled by the kernel's semi-automated distributed execution capabilities.

Keywords: Model compiler; Virtual experimentation; Tornado; Modelica

1 Introduction

Tornado [1] is an advanced kernel for modelling and virtual experimentation (*i.e.*, any evaluation of a model such as simulation, optimization, scenario analysis, ...) that was recently jointly developed by BIOMATH (Ghent University) and HEMMIS N.V. (Kortrijk, Belgium). Although the kernel is generic in nature, it is mostly adopted in the water quality domain. In water quality research, the biological and/or chemical quality of water in rivers, sewers and wastewater treatment plants (WWTP) is studied. Research in this domain is facilitated by a number of models that have received a formal or *de facto* standardization status. Most notable are River Water Quality Model No.1 (RWQM1) [2] and the Activated Sludge Model (ASM) series [3]. Water quality models typically consist of large sets of non-linear Ordinary Differential Equations (ODE) and/or Differential-Algebraic Equations (DAE). These equations are mostly well-behaved, although discontinuities occur regularly. The complexity of water quality models is therefore not in the nature of the equations, but in the sheer number. In WWTP, smaller models such as the well-known Benchmark Simulation Model (BSM) [4] consist of approximately 150 derived variables. Larger systems have up to 1,000 derived variables and over 10,000 (partly coupled) parameters. On a typical workstation, a simulation run usually lasts minutes to hours.

The modelling language that has thus far been used in the scope of Tornado is MSL (Model Specification Language) [5]. This language is similar to Modelica [6] in the sense that it is high-level, declarative and object-oriented. In fact, both MSL and Modelica were designed according to the ideas resulting from the 1993 ESPRIT Basic Research Working Group 8467 on "Simulation for the Future: new concepts, tools and applications" [7]. Although similar in nature, MSL lacks some of the readability and expressiveness of Modelica. Therefore, it was decided to work towards inclusion of support for Modelica-based modelling in the Tornado framework.

The most recent result of our efforts to bridge the gap between Modelica and Tornado is a model compiler that converts flat Modelica (*i.e.*, a Modelica model description that does not rely on inheritance nor decomposition) to executable models suitable for use with the Tornado kernel. At the moment, this compiler is a prototype that supports basic functionalities of the Modelica language. However, it does allow for a subset of Modelica models to be used in the context of Tornado. Since solutions already exist that generate flat Modelica from full Modelica (*e.g.*, `omc` - the OpenModelica Compiler), only the conversion from flat Modelica to

executable model code had to be implemented.
The sequel of this paper is structured as follows: Section 2 and Section 3 respectively provide a further introduction to Tornado and its complex virtual experimentation capabilities. Subsequently, Section 4 explains how Modelica models can be used in Tornado. Section 5 discusses two simple Modelica models for which virtual experiments were run with Tornado. Finally, Section 6 contains some conclusions and references to future work.

2 Tornado

The Tornado kernel for modelling and virtual experimentation attempts to offer a compromise between the computational efficiency of custom hard-coded (typically FORTRAN or C) model implementations and the flexibility of less computationally efficient generic tools such as MATLAB. In Tornado, hierarchical models are specified in high-level, declarative, object-oriented modelling languages such as MSL [5] and - since recently - also Modelica. From these high-level specifications, efficient executable code is generated by a model compiler. Using the dynamically-loadable executable models generated by the model compiler, Tornado allows for running a variety of so-called *virtual experiments*. Virtual experiments are the virtual-world counterpart of real-world experiments, similar to the way models relate to real-world systems. A highly simplified conceptual diagram of Tornado is shown in Figure 1.

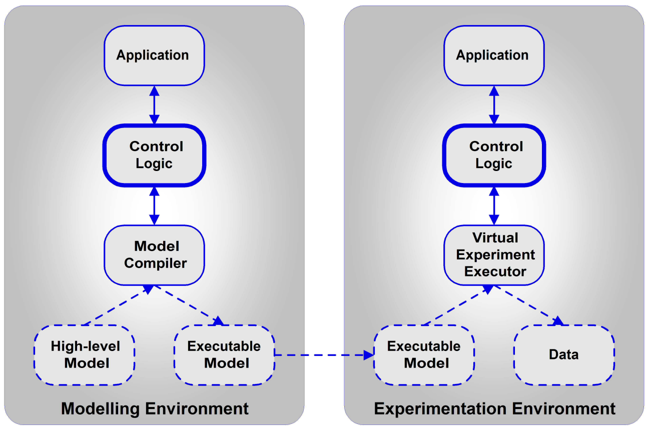

Figure 1: Tornado Conceptual Diagram

The Tornado kernel relies on a flexible input provider and output acceptor mechanism to deal with I/O for virtual experiments. Input can be provided by any combination of data files, internal data buffers and data generators. Output will be accepted by any combinations of data files, internal data buffers and plot handles (Note: since Tornado is merely a kernel, it does not have any data visualization interface of its own).
In order to allow for the kernel to be deployed in a diverse array of applications, it has been equipped with multiple interfaces. Next to its native C++ interface, Tornado currently also has a C, .NET and MATLAB MEX interface (*cf.* Figure 2). The kernel is portable across platforms and was designed according to the three-tier principle. Most persistent representations of information types are XML-based. The grammar of these representations is expressed in XSD (XML Schema Definition) format and mimics very closely the internal representation of the respective types of information. An interesting feature of Tornado is the fact that it allows for dynamic loading of numerical solvers for tasks such as integration, optimization and Latin Hypercube Sampling. In order to support this principle, a generalized framework has been set up [8].

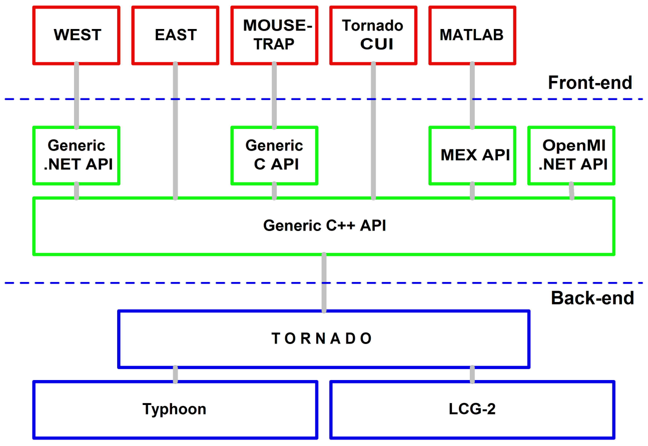

Figure 2: Tornado-based Interfaces and Applications

Several applications (graphical and other) can be built on top of Tornado. Examples include the next generation of the WEST® [5] commercial modelling and simulation tool for WWTP's, its research-oriented counterpart named EAST and DHI's MOUSE-TRAP (*cf.*, http://www.dhigroup.com/Software/Urban.aspx). However, the most direct way of using the kernel is through the Tornado CUI (Command-line User Interface) suite, which is a comprehensive set of tools that is included with the kernel distribution. Full-fledged graphical applications such as WEST® are conceived to be used by all types of users (expert, intermediate, novice). The Tornado CUI suite however focuses on experts only. Table 1 gives an overview of the most commonly used command-line tools. The results discussed further in this paper were obtained through the Tornado CUI suite.

Table 1: Tornado CUI Suite

Program	Description
`tbuild`	Compiles and links executable model code to a dynamically-loadable binary object (.dll / .so)
`tcreate`	Creates an empty XML description of a virtual experiment
`texec`	Executes virtual experiments described in XML
`tinitial`	Dumps all model quantity values after initialization
`tmsl`	Compiles a high-level MSL model to executable model code
`tobj`	Computes aggregation functions and other criteria from simulation trajectories
`tproject`	Manages sets of related experiments and connection graphs
`tsort`	Sorts a Tornado-generated data file
`t2msl`	Converts a connection graph to MSL code

3 Complex Virtual Experimentation

Tornado consists of strictly separated modelling and virtual experimentation environments. Virtual experiments can either be *atomic* or *compound*. The latter are hierarchically structured whereas the first cannot be further decomposed. Atomic experiment types that are available in Tornado are dynamic simulation and steady-state analysis. The most straightforward types of compound experiments are optimization, scenario analysis, Monte Carlo analysis (*e.g.* using Latin Hypercube Sampling) and sensitivity analysis. More convoluted types of compound experiments are also available, such as combinations of scenario / Monte Carlo analysis and optimization. Thanks to the object-oriented nature of Tornado, new virtual experiment types can easily be added. Several types of virtual experiments are based on the computation of objective values. As far as possible, the same set of objective types is available for each objective-based experiment type, thereby promoting orthogonality.

Given the hierarchical nature of compound virtual experiments, computational complexity can be substantial. Tornado therefore allows for coarse-grained gridification of certain types of compound virtual experiments. Supported distributed execution environments include BIOMATH's Typhoon cluster software [9] and CERN's LCG-2 grid middleware (*cf.*, http://public.eu-egee.org). Tornado generates generic job descriptions for dynamic execution. Typhoon is capable of directly interpreting these generic job descriptions, whereas for LCG-2, an additional conversion step has to be applied.

Using Tornado's powerful complex virtual experimentation capabilities, large risk/cost/benefit analyses for integrated water systems were carried out, including Latin Hypercube Sampling from multi-dimensional parameter spaces and the automated execution of 1,000's of simulations [10], each requiring an average of 0.5h of computation time.

4 Using Modelica Models in Tornado

In Tornado, executable models consist of two distinct parts. The first part is represented in C and is made up of the actual model equations, in addition to a number of flat arrays containing data containers for parameters and variables. The second part is a XML representation of meta-information, *i.e.*, information regarding names, descriptions, units, constraints, ... of parameters, variables and models. The relationship between these hierarchically structured meta-information items and the respective elements of the flat C arrays is also expressed in XML. The availability of meta-information in executable models allows for the latter to be self-describing, which is a requirement given the strict separation between modelling and experimentation in Tornado.

In the Tornado framework, model compilers are to generate executable models in the format that was described above. The MSL model compiler that is part of the Tornado suite generates these executable models directly from MSL input. In the case of Modelica however, the approach is two-phased. During the first phase, the OpenModelica Compiler is used to generate flat Modelica (.mof) from full Modelica input (.mo). During the second phase, a new Tornado CUI tool called *mof2t* is used to convert flat Modelica to the Tornado executable model format. This approach was mainly inspired by practical considerations (lack of resources for the re-implementation of the non-trivial full-to-flat Modelica conversion). At the moment *mof2t* only supports a subset of flat Modelica.

For the development of the *mof2t* compiler, the same technologies and libraries were used as for the remainder of the Tornado framework, *i.e.*, C++, flex/bison, and Elcel Technologies OpenTop (*cf.*, http://www.elcel.com). The *mof2t* compiler nicely completes the Tornado CUI suite, which in all consists of approximately 20 tools. The relationship between *mof2t* and the most important other CUI tools is depicted in Figure 3.

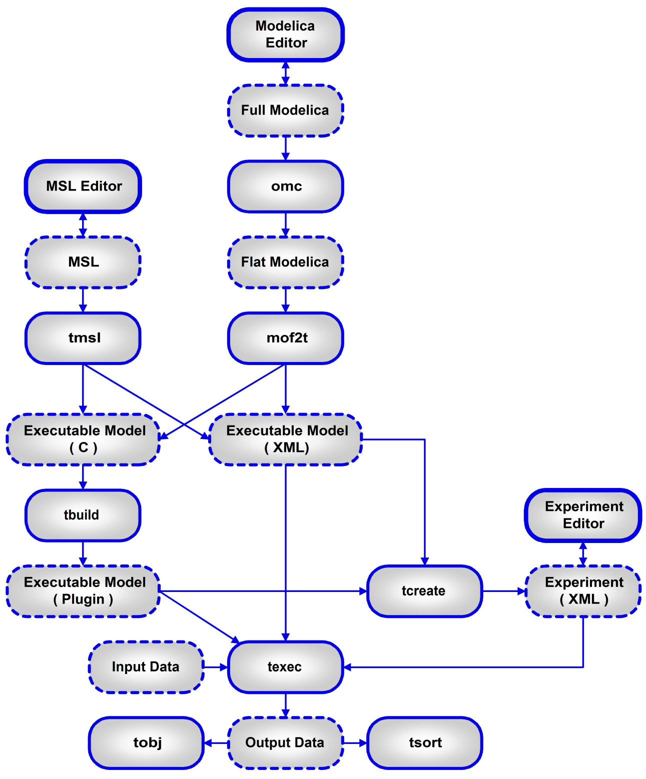

Figure 3: Relationship between the Main Tornado CUI Tools

5 Examples

In this section, two simple cases are presented that illustrate the use of Modelica models in Tornado. The first case is based on the ubiquitous *Van der Pol* system, which is frequently found as an example in modelling and simulation textbooks when stiff systems are discussed. The second case is based on the *ARGESIM - C1* simulator comparison. In both cases, results were obtained using the Tornado CUI suite. Evidently, when using Tornado through one of the GUI applications that it supports, most of the technical details shown below are hidden from the user.

5.1 Van der Pol

5.1.1 Model

The *Van der Pol* system can be described in Modelica as follows:

```
VanDerPol.mof:
---
fclass VanDerPol
Real x(start = 1.0);
Real y(start = 1.0);
parameter Real mu = 1;
equation
  der(x) = y;
  der(y) = -x + mu * (1.0 - x * x) * y;
end VanDerPol;
---
```

Given the fact that this model does not rely on inheritance nor decomposition, there is no difference between its full and flattened version. In order to generate executable code for Tornado and convert this code into a dynamically-loadable object, the `mof2t` and `tbuild` CUI tools are to be used:

```
> mof2t VanDerPol.mof

Flat Modelica to Tornado Convertor (Build: Jun 23 2006)

I   Loading license spec: Tornado.lic|.Tornado
I   Checking MAC address...
I   Starting executable model code generation...
I   Executable model code generation ended
I   Total execution time: 0 seconds

> tbuild -p win32-msvc7.1 VanDerPol

Tornado Model Builder (Build: Jun 23 2006)

I   Loading license spec: Tornado.lic|.Tornado
I   Checking MAC address...
I   Starting build...
I   Build ended
I   Total execution time: 0 seconds
```

The executable C code and XML meta-information that is generated by `mof2t` is as follows:

```
VanDerPol.c:
---
#include <math.h>
#include <stdlib.h>

#include "Tornado/EE/Common/DLL.h"
#include "Tornado/EE/MSLE/MSLE.h"

#define _mu_ pModel->Params[0]
#define _time_ pModel->IndepVars[0]
#define _x_ pModel->DerVars[0]
#define _D_x_ pModel->Derivatives[0]
#define _y_ pModel->DerVars[1]
#define _D_y_ pModel->Derivatives[1]

void ComputeInitial(struct TModel* pModel) {}

void ComputeState(struct TModel* pModel)
{
  _D_x_ = _y_;
  _D_y_ = -_x_ + _mu_ * (1 - _x_ * _x_) * _y_;
}

void ComputeOutput(struct TModel* pModel) {}

void ComputeFinal(struct TModel* pModel) {}

void* GetID() { return (void*)L"Tornado.MSLE.Model.VanDerPol"; }

void* Create()
{
  struct TModel* pModel;

  pModel = (struct TModel*)malloc(sizeof(struct TModel));

  pModel->Type = L"ODE";

  pModel->NoParams = 1;
  pModel->NoIndepVars = 1;
  pModel->NoInputVars = 0;
  pModel->NoOutputVars = 0;
  pModel->NoAlgVars = 0;
  pModel->NoDerVars = 2;
  pModel->NoDerivatives = 2;
  pModel->NoPrevious = 0;
  pModel->NoResidues = 0;
  pModel->NoSolveSets = 0;
  pModel->NoEvents = 0;
  pModel->Params =
    (double*)malloc(sizeof(double) * pModel->NoParams);
  pModel->IndepVars =
    (double*)malloc(sizeof(double) * pModel->NoIndepVars);
  pModel->InputVars =
    (double*)malloc(sizeof(double) * pModel->NoInputVars);
  pModel->OutputVars =
    (double*)malloc(sizeof(double) * pModel->NoOutputVars);
  pModel->AlgVars =
    (double*)malloc(sizeof(double) * pModel->NoAlgVars);
  pModel->DerVars =
    (double*)malloc(sizeof(double) * pModel->NoDerVars);
  pModel->Derivatives =
    (double*)malloc(sizeof(double) * pModel->NoDerivatives);
  pModel->Previous =
    (double*)malloc(sizeof(double) * pModel->NoPrevious);
  pModel->Residues =
    (double*)malloc(sizeof(double) * pModel->NoResidues);
  pModel->SolveSets =
    (TSolveSetP)malloc(sizeof(struct TSolveSet) *
                       pModel->NoSolveSets);
  pModel->Events =
    (TEventP)malloc(sizeof(struct TEvent) * pModel->NoEvents);

  pModel->ComputeInitial = ComputeInitial;
  pModel->ComputeState = ComputeState;
  pModel->ComputeOutput = ComputeOutput;
  pModel->ComputeFinal = ComputeFinal;

  return (void*)pModel;
}
---
```

```
VanDerPol.SymbModel.xml:
---
<Tornado>
  <Model>
    <Exec FileName="VanDerPol"/>
    <Symb>
      <Model Name="">
        <Params>
          <Param Name="mu" DefaultValue="1"/>
        </Params>
        <IndepVars>
          <IndepVar Name="time" DefaultValue="0"/>
        </IndepVars>
        <InputVars>
        </InputVars>
        <OutputVars>
        </OutputVars>
        <AlgVars>
        </AlgVars>
        <DerVars>
          <DerVar Name="x" DefaultValue="1"/>
          <DerVar Name="y" DefaultValue="1"/>
        </DerVars>
        <Models>
        </Models>
      </Model>
    </Symb>
    <Links>
      <Link Name=".mu" ValueType="Params" ValueIndex="0"/>
      <Link Name=".time" ValueType="IndepVars" ValueIndex="0"/>
      <Link Name=".x" ValueType="DerVars" ValueIndex="0"
            DerivativeType="Derivatives" DerivativeIndex="0"/>
      <Link Name=".y" ValueType="DerVars" ValueIndex="1"
            DerivativeType="Derivatives" DerivativeIndex="1"/>
    </Links>
  </Model>
</Tornado>
---
```

The exact semantics of these representations are beyond the scope of this paper and will therefore not be further discussed. Important to note however is that the format of the generated C code has been kept as simple as possible in order to be able to compile the code with as many C compilers as possible. The compilers that have been shown to work so far are Borland C++ 5.5, MS Visual C++ 6.0, 7.1 & 8.0, LCC, INTEL C++ 9.0 and g++.

5.1.2 Simulation

In order to simulate the generated model code, a simulation experiment spec is to be provided to the experiment executor. Specs must conform to the respective XML schemas that have been defined in the scope of Tornado. When using the Tornado CUI suite, empty specs can be generated by the `tcreate` program and must then be further completed manually. Below is a simulation experiment spec for the *Van der Pol* model that was generated by invoking `tcreate -t Simul VanDerPol` and further completed through manual editing:

```
VanDerPol.Simul.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="Simul">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Wed Jun 28 14:58:02 2006"/>
      <Prop Name="Desc" Value="Van der Pol simulation"/>
      <Prop Name="FileName"
            Value="VanDerPol.Simul.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <Simul>
      <Model Name="VanDerPol" CheckBounds="false">
        <Quantities>
          <Quantity Name=".mu" Value="100"/>
          <Quantity Name=".x" Value="2"/>
        </Quantities>
      </Model>
      <Inputs Enabled="false">
      </Inputs>
      <Outputs Enabled="true">
        <Output Name="*Calc*">
          <CalcVars Enabled="false">
          </CalcVars>
        </Output>
        <Output Name="*Plot*">
          <Plot Enabled="false">
            <Props>
              <Prop Name="CommInt" Value="0"/>
              <Prop Name="Info" Value=""/>
              <Prop Name="Interpolated" Value="false"/>
              <Prop Name="StartTime" Value="-INF"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
            <Quantities>
            </Quantities>
          </Plot>
        </Output>
        <Output Name="File">
          <File Name="VanDerPol.Simul.out.txt" Enabled="true">
            <Props>
              <Prop Name="CommInt" Value="0"/>
              <Prop Name="CommIntType" Value="Linear"/>
              <Prop Name="DecSep" Value="."/>
              <Prop Name="Interpolated" Value="false"/>
              <Prop Name="Precision" Value="8"/>
              <Prop Name="StartTime" Value="-INF"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
            <Quantities>
              <Quantity Name=".x"/>
              <Quantity Name=".y"/>
            </Quantities>
          </File>
        </Output>
      </Outputs>
      <Time>
        <Start Value="0"/>
        <Stop Value="300"/>
      </Time>
      <Solve>
        <Integ Method="CVODE">
          <Props>
            <Prop Name="AbsoluteTolerance" Value="1e-006"/>
            <Prop Name="CVBandLowerBandwidth" Value="0"/>
            <Prop Name="CVBandUpperBandwidth" Value="0"/>
            <Prop Name="CVSPGMRGSType" Value="ModifiedGS"/>
            <Prop Name="IterationMethod" Value="Newton"/>
            <Prop Name="LinearMultistepMethod" Value="BDF"/>
            <Prop Name="LinearSolver" Value="Diag"/>
            <Prop Name="MaxNoSteps" Value="0"/>
            <Prop Name="RelativeTolerance" Value="1e-005"/>
          </Props>
        </Integ>
      </Solve>
    </Simul>
  </Exp>
</Tornado>
---
```

For a simulation experiment, XML specs basically allow for specifying initial values (hereby overruling initializations that were specified through the model's meta-information), defining input providers / output acceptors, specifying the simulation start / stop time and configuring integrator solver settings. In this case, initial values were given for `.mu` and `.x`, input was disabled and one output file acceptor was defined. The simulation will be run from 0 to 300 and the CVODE stiff system solver (*cf.*, http://www.llnl.gov/CASC/sundials) will be used as an integrator. Important to note is that the settings of the CVODE integrator are given through a flexible attribute-value pair mechanism instead of through tags that are part of the XML grammar. This is required to support dynamic loading of solver plugins. The fragment below shows the output of the experiment executor, when applied to the *VanDerPol* simulation spec. One will notice that before execution starts, a number of solver plugins are dynamically loaded (in this case only a subset of the 35 solver plugins that are provided with Tornado are loaded):

```
> texec VanDerPol.Simul.Exp.xml

Tornado Experiment Executor (Build: Jun 13 2006)

I   Loading license spec: Tornado.lic|.Tornado
I   Checking MAC address...
I   Loading main spec: Tornado.Main.xml|.Tornado
I   Loading plugin: Tornado.Solve.Integ.CVODE
I   Loading plugin: Tornado.Solve.Integ.Euler
I   Loading plugin: Tornado.Solve.Integ.RK4
I   Loading plugin: Tornado.Solve.Integ.RK4ASC
I   Loading plugin: Tornado.Solve.Optim.GA
I   Loading plugin: Tornado.Solve.Optim.Praxis
I   Loading plugin: Tornado.Solve.Optim.SA
I   Loading plugin: Tornado.Solve.Optim.Simplex
I   Loading plugin: Tornado.Solve.Root.Broyden
I   Loading plugin: Tornado.Solve.Root.Hybrid
I   Loading plugin: Tornado.Solve.Scen.Cross
I   Loading plugin: Tornado.Solve.Scen.Fixed
I   Loading plugin: Tornado.Solve.Scen.Grid
I   Loading plugin: Tornado.Solve.Scen.Plain
I   Loading plugin: Tornado.Solve.Scen.Random
I   Loading plugin: Tornado.Solve.Sens.Plain
I   Loading plugin: Tornado.Solve.CI.Nelder
I   Loading plugin: Tornado.Solve.CI.Richardson
I   Loading plugin: Tornado.Solve.MC.IHS
I   Loading plugin: Tornado.Solve.MC.CVT
I   Loading plugin: Tornado.Solve.MC.LHS
I   Loading plugin: Tornado.Solve.MC.PR
I   Main information:
I     Author = PCFC1\fc
I     Date = Thu Oct 13 16:08:06 2005
I     Desc = Main spec
I     EnableHashOutputHeaders = true
I     EnableWESTInputHeaders = true
I     EnableWESTOutputHeaders = false
I     FileName =
I     KernelAuthor = Filip Claeys, Dirk De Pauw
I     KernelDesc = Advanced Kernel for Modelling and Virtual Ex...
I     KernelVersion = 0.22
I     LimitMRE = 0.7
I     LimitSRE = 0.0235
I     Precision = 8
I   New job: VanDerPol.Simul.Exp.xml
I   Starting thread...
I   Loading experiment spec: VanDerPol.Simul.Exp.xml|.Tornado
I   Loading simulation experiment spec: VanDerPol.Simul.Exp.xml...
I   Loading symbolic model spec: VanDerPol.SymbModel.xml|.Tornado
I   Loading executable model: Tornado.MSLE.Model.VanDerPol
I   Executable model information:
I     Type = ODE
I     #Params = 1
I     #IndepVars = 1
I     #InputVars = 0
I     #OutputVars = 0
I     #AlgVars = 0
I     #DerVars = 2
I     #Derivatives = 2
I     #Previous = 0
I     #Residues = 0
I     #SolveSets = 0
I     #Events = 0
I   Building model symbol table...
I   Checking model linkage...
I   Creating simulator...
I   Setting integration solver: Tornado.Solve.Integ.CVODE
I   Experiment information:
I     Type = Simul
I     Embedded = true
I     Author = PCFC1\fc
I     Date = Wed Jun 28 14:58:02 2006
I     Desc = Van der Pol simulation experiment
I     FileName = VanDerPol.Simul.Exp.xml|.Tornado
I     UnitSystem =
I   Initializing model...
I   Opening simulation output file: VanDerPol.Simul.out.txt
I   Simulation from 0 to 300
I   Starting simulation...
I   Simulation ended
I   Closing simulation output file: VanDerPol.Simul.out.txt
I   Executable model statistics:
I     #ComputeInitials: 1
I     #ComputeStates: 3240
I     #ComputeOutputs: 1686
I     #ComputeFinals: 1
I   Total execution time: 0 seconds
I   Thread ended
I   Unloading plugins
```

The Tornado CUI suite does not contain any data visualization mechanism, however one can easily use tools such as MS Excel, MATLAB or GNUPlot to display the simulated trajectories. Figure 4 shows the result of invoking the following commands in GNUPlot:

```
set xlabel "t"
set ylabel ".x"
plot 'VanDerPol.Simul.out.txt' using 1:2 with lines
```

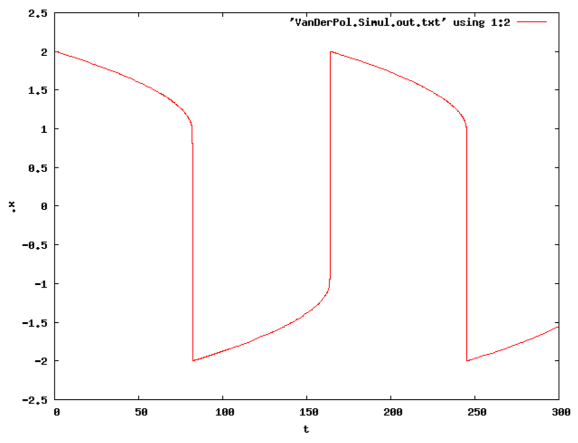

Figure 4: Van Der Pol for .mu = 100

5.1.3 Parameter variation

More interesting it becomes if we wish to run the same simulation for different initial values. For instance, suppose we wish to run the simulation for values of *.mu* that are logarithmically spaced between 1 and 100 with a spacing of 2. Suppose also that for each simulation, we want to determine the maximum value and standard deviation of the trajectory of *.y*, in addition to the value of *.y* at *t = 50*. The scenario analysis experiment shown below provides a solution to this problem:

```
VanDerPol.Scen.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="Scen">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Wed Jun 28 15:37:57 2006"/>
      <Prop Name="Desc" Value="VanDerPol scenario analysis"/>
      <Prop Name="FileName"
             Value="VanDerPol.Scen.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <Scen>
      <Obj>
        <Exp Version="1.0" Type="Simul"
            FileName="VanDerPol.Simul.Exp.xml|.Tornado"/>
        <Props>
          <Prop Name="CommInt" Value="0"/>
          <Prop Name="DecSep" Value="."/>
          <Prop Name="EnableNoComputeStates" Value="false"/>
          <Prop Name="EnableRetrieval" Value="false"/>
          <Prop Name="EnableStorage" Value="true"/>
          <Prop Name="Interpolated" Value="false"/>
          <Prop Name="OutputFileName"
               Value="VanDerPol.Scen.Simul.out.txt.{}"/>
          <Prop Name="Precision" Value="8"/>
          <Prop Name="ThousandSep" Value=","/>
          <Prop Name="TyphoonBaseName" Value="Typhoon"/>
        </Props>
        <Quantities>
          <Quantity Name=".y">
            <Props>
              <Prop Name="Criterion" Value="AbsSquared"/>
              <Prop Name="EnableAvg" Value="false"/>
              <Prop Name="EnableDiffMax" Value="false"/>
              <Prop Name="EnableDiffSum" Value="false"/>
              <Prop Name="EnableEndValue" Value="false"/>
              <Prop Name="EnableInt" Value="false"/>
              <Prop Name="EnableMax" Value="true"/>
              <Prop Name="EnableMin" Value="false"/>
              <Prop Name="EnableStdDev" Value="true"/>
              <Prop Name="EnableTIC" Value="false"/>
              <Prop Name="EnableValueOnTime" Value="true"/>
              <Prop Name="Time" Value="50"/>
              <Prop Name="Weighted" Value="false"/>
            </Props>
          </Quantity>
        </Quantities>
      </Obj>
      <Log Name="VanDerPol.Scen.log.txt" Enabled="true">
        <Props>
        </Props>
      </Log>
      <Inputs Enabled="false">
      </Inputs>
      <Outputs Enabled="true">
        <Output Name="*File*">
          <File Name="VanDerPol.Scen.out.txt" Enabled="true">
            <Props>
              <Prop Name="DecSep" Value="."/>
              <Prop Name="Precision" Value="8"/>
            </Props>
          </File>
        </Output>
        <Output Name="*Plot*">
          <Plot Enabled="false">
            <Props>
              <Prop Name="Info" Value=""/>
            </Props>
          </Plot>
        </Output>
      </Outputs>
      <Vars>
        <Var Name=".mu">
          <Props>
            <Prop Name="DistributionMethod" Value="Logarithmic"/>
            <Prop Name="LowerBound" Value="1"/>
            <Prop Name="NoValues" Value="0"/>
            <Prop Name="RefValue" Value="1"/>
            <Prop Name="Spacing" Value="2"/>
            <Prop Name="StdDev" Value="0"/>
            <Prop Name="UpperBound" Value="100"/>
            <Prop Name="UpperBoundPolicy"
                  Value="IncludeUpperBound"/>
            <Prop Name="Values" Value=""/>
          </Props>
        </Var>
      </Vars>
      <Solve>
        <Scen Method="Grid">
          <Props>
            <Prop Name="EnableRef" Value="false"/>
            <Prop Name="Generate" Value="true"/>
            <Prop Name="UseTyphoon" Value="false"/>
          </Props>
        </Scen>
      </Solve>
    </Scen>
  </Exp>
</Tornado>
```

As one can see, this scenario analysis spec refers to a simulation spec that resides in an external file (VanDerPol.Simul.Exp.xml). Directly embedding the XML content of this file into the scenario analysis experiment is however also possible. One will also notice that other types of objectives and aggregation functions (next to the *Min*, *StdDev*, *ValueOnTime* functions that are needed for our application) such as *Avg* (average) and *Int* (integral) are also possible. Table 2 shows the contents of the *VanDerPol.Scen.out.txt* file that is generated during the execution of the scenario analysis.

Table 2: Results of the VarDerPol.Scen.Exp.xml Experiment

RunNo	.mu	Max(.y)	StdDev(.y)	ValueOnTime(.y)
1	1	2.6865596	1.4272097	-1.5137494
2	2	3.8300373	1.4645402	-0.034409599
3	4	6.3463996	1.5316644	0.58337344
4	8	11.553678	1.593544	-0.092195286
5	16	22.097001	1.6317544	0.058550465
6	32	43.305241	1.6404037	0.052380761
7	64	85.828672	1.5529595	-0.0439915
8	100	133.68028	1.4986709	-0.010200556

5.2 ARGESIM - C1

ARGE Simulation News (*cf.*, http://www.argesim.org) is a non-profit working group providing the infrastructure and adminstration for dissemination of information on modelling and simulation in Europe. ARGESIM is located at Vienna University of Technology, Dept. Simulation and publishes Simulation News Europe (SNE), which features a series on comparisons of simulation software. Based on simple, easily comprehensible models special features of modelling and experimentation within simulation languages, also with respect to an application area, are compared. Features are, for instance: modelling technique, event-handling, numerical integration, steady-state calculation, distribution fitting, parameter sweep, output analysis, animation, complex logic strategies, submodels, macros, statistical features *etc*. Approximately 20 comparisons have thusfar been defined, the first was published in November 1990, the last in December 2005.

5.2.1 Model

As a second example of the use of Modelica models in Tornado, the *C1 ARGESIM* comparison will be used. The model that is at the basis of this comparison can be represented in Modelica as follows:

```
C1.mof:
---
fclass C1
  parameter Real kr = 1;
  parameter Real kf = 0.1;
  parameter Real lf = 1000;
  parameter Real dr = 0.1;
  parameter Real dm = 1;
  parameter Real p = 0;
  Real f(start = 9.975);
  Real m(start = 1.674);
  Real r(start = 84.99);
equation
  der(r) = -dr * r + kr * m * f;
  der(m) = dr * r - dm * m + kf * f * f - kr * m * f;
  der(f) = dr * r + 2 * dm * m - kr * m * f -
           2 * kf * f * f - lf * f + p;
end C1;
---
```

The comparison requires the following tasks to be performed:

- *Simulation of the stiff system over [0,10].*
- *Parameter variation of lf from 1.0e2 to 1.0e4 and a plot of all f(t; lf), logarithmic steps preferred.*
- *Calculation of steady states during constant bombardment (p(t) = pc = 1.0E4) and without bombardment (p(t) = 0).*

5.2.2 Simulation

As in the first example, a dynamically-loadable executable model for Tornado can be generated using `mof2t` and `tbuild`. Afterwards, an empty simulation experiment can be generated with `tcreate` and then be completed through manual editing:

```
C1.Simul.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="Simul">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Fri Jun 30 12:28:12 2006"/>
      <Prop Name="Desc" Value=""/>
      <Prop Name="FileName"
            Value="C1.CVODE.Simul.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <Simul>
      <Model Name="C1" CheckBounds="false">
        <Quantities>
        </Quantities>
      </Model>
      <Inputs Enabled="false">
      </Inputs>
      <Outputs Enabled="true">
        <Output Name="*Calc*">
          <CalcVars Enabled="false">
          </CalcVars>
        </Output>
        <Output Name="*Plot*">
          <Plot Enabled="false">
            <Props>
              <Prop Name="CommInt" Value="0"/>
              <Prop Name="Info" Value=""/>
              <Prop Name="Interpolated" Value="false"/>
              <Prop Name="StartTime" Value="-INF"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
            <Quantities>
            </Quantities>
          </Plot>
        </Output>
        <Output Name="File">
          <File Name="C1.Simul.out.txt" Enabled="true">
            <Props>
              <Prop Name="CommInt" Value="1.2"/>
              <Prop Name="CommIntType" Value="Logarithmic"/>
              <Prop Name="DecSep" Value="."/>
              <Prop Name="Interpolated" Value="true"/>
              <Prop Name="Precision" Value="8"/>
              <Prop Name="StartTime" Value="1e-007"/>
              <Prop Name="StopTime" Value="+INF"/>
              <Prop Name="UseDisplayUnits" Value="true"/>
            </Props>
            <Quantities>
              <Quantity Name=".f"/>
              <Quantity Name=".m"/>
              <Quantity Name=".r"/>
            </Quantities>
          </File>
        </Output>
      </Outputs>
      <Time>
        <Start Value="0"/>
        <Stop Value="10"/>
      </Time>
      <Solve>
        <Integ Method="CVODE">
          <Props>
            <Prop Name="AbsoluteTolerance" Value="1e-006"/>
            <Prop Name="CVBandLowerBandwidth" Value="0"/>
            <Prop Name="CVBandUpperBandwidth" Value="0"/>
            <Prop Name="CVSPGMRGSType" Value="ModifiedGS"/>
            <Prop Name="IterationMethod" Value="Functional"/>
            <Prop Name="LinearMultistepMethod" Value="Adams"/>
            <Prop Name="LinearSolver" Value="Dense"/>
            <Prop Name="MaxNoSteps" Value="0"/>
            <Prop Name="RelativeTolerance" Value="1e-006"/>
          </Props>
        </Integ>
        <Root Method="Broyden">
          <Props>
            <Prop Name="MaxNoSteps" Value="0"/>
            <Prop Name="MaxStepSize" Value="1"/>
          </Props>
        </Root>
      </Solve>
    </Simul>
  </Exp>
</Tornado>
---
```

Important to notice in this simulation experiment is that for the output file acceptor, the communication interval type (*CommIntType*) was set to logarithmic. In this case, logarithmic spacing of output timepoints is required in order to be able to accurately represent the dynamics of the simulated trajectories during the initial phase of the simulation (without generating huge amounts of irrelevant data). After running the simulation with `texec`, one could for instance use GNUPlot to display the results (see Figure 5) onto logarithmic axes using the following commands:

```
set logscale xy
set xlabel "t"
set ylabel ".f"
plot 'C1.Simul.out.txt' using 1:2 with lines
```

5.2.3 Parameter variation

The parameter variation that is requested by the comparison can easily be implemented in Tornado us-

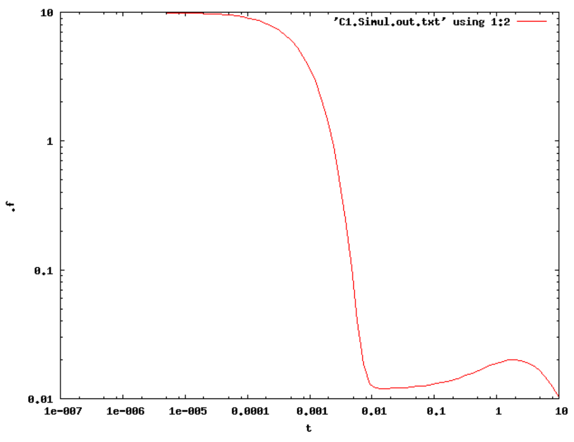

Figure 5: Simulation results for the ARGESIM C1 model

ing the scenario analysis experiment type. However, in contrast to the *Van der Pol* example, no post-processing functions (such as *Min*, *StdDev*, ...) are needed in this case. Important however is that the variation of *.lf* is to be set to *Logarithmic*, as requested. Figure 6 shows the results of a 10-shot scenario analysis experiment defined in this way.

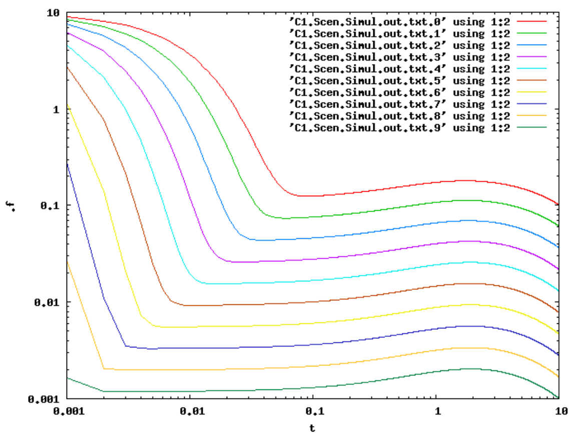

Figure 6: Scenario analysis results for the ARGESIM C1 model

5.2.4 Calculation of steady states

For the calculation of steady states, the steady-state (SS) experiment type can be used. In Tornado, the steady-state of a system is directly computed through the application of a root finding solver to the system equations, where the derivatives (*i.e.*, the left hand sides) of state equations are used as residues (that are to be brought to zero).

The following describes a steady-state experiment for the *ARGESIM C1* model where *p* = *1e4*:

```
C1.p=1e4.SS.Exp.xml:
---
<Tornado>
  <Exp Version="1.0" Type="SS">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Fri Jun 30 15:39:27 2006"/>
      <Prop Name="Desc" Value=""/>
      <Prop Name="FileName" Value="C1.p=1e4.SS.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <SS>
      <Model Name="C1" CheckBounds="false">
        <Quantities>
          <Quantity Name=".p" Value="1e4"/>
        </Quantities>
      </Model>
      <Solve>
        <Root Method="Hybrid">
          <Props>
            <Prop Name="Tolerance" Value="1e-008"/>
          </Props>
        </Root>
      </Solve>
    </SS>
  </Exp>
</Tornado>
---
```

Execution of this experiment with `texec` will instantly yield the correct steady state values for *.f*, *.m* and *.r* :

```
> texec "C1.p=1e4.SS.Exp.xml"

Tornado Experiment Executor (Build: Jun 13 2006, 10:32:40)

I   Loading license spec: Tornado.lic|.Tornado
I   Checking MAC address...
I   Loading main spec: Tornado.Main.xml|.Tornado
I   Loading plugin: Tornado.Solve.Integ.CVODE
I   Loading plugin: Tornado.Solve.Integ.Euler
I   Loading plugin: Tornado.Solve.Integ.RK4
I   Loading plugin: Tornado.Solve.Integ.RK4ASC
I   Loading plugin: Tornado.Solve.Optim.GA
I   Loading plugin: Tornado.Solve.Optim.Praxis
I   Loading plugin: Tornado.Solve.Optim.SA
I   Loading plugin: Tornado.Solve.Optim.Simplex
I   Loading plugin: Tornado.Solve.Root.Broyden
I   Loading plugin: Tornado.Solve.Root.Hybrid
I   Loading plugin: Tornado.Solve.Scen.Cross
I   Loading plugin: Tornado.Solve.Scen.Fixed
I   Loading plugin: Tornado.Solve.Scen.Grid
I   Loading plugin: Tornado.Solve.Scen.Plain
I   Loading plugin: Tornado.Solve.Scen.Random
I   Loading plugin: Tornado.Solve.Sens.Plain
I   Loading plugin: Tornado.Solve.CI.Nelder
I   Loading plugin: Tornado.Solve.CI.Richardson
I   Loading plugin: Tornado.Solve.MC.IHS
I   Loading plugin: Tornado.Solve.MC.CVT
I   Loading plugin: Tornado.Solve.MC.LHS
I   Loading plugin: Tornado.Solve.MC.PR
I   Main information:
I     Author = PCFC1\fc
I     Date = Thu Oct 13 16:08:06 2005
I     Desc = Main spec
I     EnableHashOutputHeaders = true
I     EnableWESTInputHeaders = true
I     EnableWESTOutputHeaders = false
I     FileName =
I     KernelAuthor = Filip Claeys, Dirk De Pauw
I     KernelDesc = Advanced Kernel for Modelling and Virtual...
I     KernelVersion = 0.22
I     LimitMRE = 0.7
I     LimitSRE = 0.0235
I     Precision = 8
I   New job: C1.p=1e4.SS.Exp.xml
I   Starting thread...
I   Loading experiment spec: C1.p=1e4.SS.Exp.xml|.Tornado
I   Loading steady-state analysis experiment spec: C1.p=1e4...
I   Loading symbolic model spec: C1.SymbModel.xml|.Tornado
I   Loading executable model: Tornado.MSLE.Model.C1
I   Executable model information:
I     Type = ODE
I     #Params = 6
I     #IndepVars = 1
I     #InputVars = 0
I     #OutputVars = 0
I     #AlgVars = 0
I     #DerVars = 3
I     #Derivatives = 3
```

```
I     #Previous = 0
I     #Residues = 0
I     #SolveSets = 0
I     #Events = 0
I   Building model symbol table...
I   Checking model linkage...
I   Creating steady-state analyser...
I   Setting root solver: Tornado.Solve.Root.Hybrid
I   Experiment information:
I     Type = SS
I     Embedded = true
I     Author = PCFC1\fc
I     Date = Fri Jun 30 15:39:27 2006
I     Desc =
I     FileName = C1.p=1e4.SS.Exp.xml|.Tornado
I     UnitSystem =
I   Initializing model...
I   Initializing model...
I   Starting steady-state analysis...
I   Steady-state analysis ended
I   Executable model statistics:
I     #ComputeInitials: 8
I     #ComputeStates: 8
I     #ComputeOutputs: 0
I     #ComputeFinals: 0
I   Final variable values:
I     .f = 10
I     .m = 10
I     .r = 1000
I   Total execution time: 0 seconds
I   Thread ended
I   Unloading plugins
```

For *p = 0*, one can proceed in a similar way. However, in this case the process is more sensitive to the initial value of the state variables. The experiment below therefore shows that for *.f*, a differing initial value had to be chosen to ensure convergence of the algorithm.

```
C1.p=0.SS.Exp.xml|.Tornado:
---
<Tornado>
  <Exp Version="1.0" Type="SS">
    <Props>
      <Prop Name="Author" Value="PCFC1\fc"/>
      <Prop Name="Date" Value="Fri Jun 30 15:39:20 2006"/>
      <Prop Name="Desc" Value=""/>
      <Prop Name="FileName" Value="C1.p=0.SS.Exp.xml|.Tornado"/>
      <Prop Name="UnitSystem" Value=""/>
    </Props>
    <SS>
      <Model Name="C1" CheckBounds="false">
        <Quantities>
          <Quantity Name=".f" Value="0.1"/>
        </Quantities>
      </Model>
      <Solve>
        <Root Method="Hybrid">
          <Props>
            <Prop Name="Tolerance" Value="1e-008"/>
          </Props>
        </Root>
      </Solve>
    </SS>
  </Exp>
</Tornado>
---
```

```
...
I   Fri Jun 30 15:59:44 2006   Final variable values:
I   Fri Jun 30 15:59:44 2006     .f = 0
I   Fri Jun 30 15:59:44 2006     .m = 0
I   Fri Jun 30 15:59:44 2006     .r = -2.47032822920623e-323
I   Fri Jun 30 15:59:44 2006   Total execution time: 0 seconds
...
```

6 Conclusions and Future Work

Through the development of the *mof2t* compiler, Tornado's powerful complex virtual experimentation capabilities have become available for a subset of Modelica models. To facilitate maintenance and further integration, *mof2t* was implemented using the same technologies as the remainder of the Tornado framework. In the forthcoming months, the *mof2t* will be further stabilized and enhanced.

Acknowledgement

Peter A. Vanrolleghem is Canadian Research Chair in Water Quality Modelling.

References

[1] F. Claeys, D. De Pauw, L. Benedetti, I. Nopens, and P.A. Vanrolleghem. Tornado: A versatile efficient modelling & virtual experimentation kernel for water quality systems. In *Proceedings of the iEMSs 2006 Conference*, Burlington, VT, 2006.

[2] P. Reichert, Borchardt D., Henze M., Rauch W., Shanahan P., Somlyódy L., and P.A. Vanrolleghem. *River Water Quality Model No.1.* Scientific and Technical Report No.12. IWA Publishing, London, UK, 2001.

[3] M. Henze, W. Gujer, T. Mino, and M. van Loosdrecht. *Activated Sludge Models ASM1, ASM2, ASM2d, and ASM3.* Scientific and Technical Report No.9. IWA Publishing, London, UK, 2000.

[4] J.B. Copp, editor. *The COST simulation benchmark.* European Commission, 2002.

[5] H. Vanhooren, J. Meirlaen, Y. Amerlinck, F. Claeys, H. Vangheluwe, and P.A. Vanrolleghem. WEST: modelling biological wastewater treatment. *Journal of Hydroinformatics*, 5(1):27–50, 2003.

[6] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Press, February 2004.

[7] H. Vangheluwe, F. Claeys, S. Kops, F. Coen, and G.C. Vansteenkiste. A modelling simulation environment for wastewater treatment plant design. In *Proceedings of the 1996 European Simulation Symposium*, Genoa, Italy, October 24-26 1996.

[8] F.H.A Claeys, P.A. Vanrolleghem, and P. Fritzson. A generalized framework for abstraction and dynamic loading of numerical solvers. In *Proceedings of the 2006 European Modeling and Simulation Symposium*, Barcelona, Spain, 2006.

[9] F. Claeys, M. Chtepen, L. Benedetti, B. Dhoedt, and P.A. Vanrolleghem. Distributed virtual experiments in water quality management. *Water Science and Technology*, 53(1):297–305, 2006.

[10] L. Benedetti, D. Bixio, F. Claeys, and P.A. Vanrolleghem. A model-based methodology for benefit/cost/risk analysis of wastewater systems. In *Proceedings of the iEMSs 2006 Conference*, Burlington, VT, 2006.