Intelligent configuration of numerical solvers of environmental ODE/DAE models using machine learning techniques

Petra Claeys[a], Filip Claeys[a], Bernard De Baets[c] and Peter A. Vanrolleghem[a,b]
[a] *BIOMATH, Ghent University, Coupure Links 653, 9000 Ghent, Belgium: pclaeys @biomath.ugent.be*
[b] *modelEAU, Département de génie civil, Pavillon Pouliot, Université Laval, Québec G1K 7P4, Canada*
[c] *KERMIT, Ghent University, Coupure links 653, 9000 Gent, Belgium*

Abstract: The models considered in this work contain combinations of a large number of non-linear differential equations and algebraic equations, which have to be solved numerically. Because of this, running simulation experiments on a computer can be very time-consuming, i.e. it can last for days or even weeks. On top of that, some solvers are not appropriate for solving certain of these systems. For example, stiff solvers are only appropriate for stiff systems while they are not the best choice for non-stiff systems. Choosing the appropriate solver and solver settings for a certain initialized model can significantly shorten the computation time. In order to make this choice a scientist must combine the knowledge he/she has of the initialized model with the knowledge of certain solvers and their settings. Unfortunately, most environmental scientists do not have the opportunity to build up experience as modeller/mathematician and most of the time they rely on the default solver settings.
The goal of intelligent configuration is to automate the selection of numerical solvers and their settings in order to get the "best result" for the solution of a certain initialized model. The "best result" is influenced by the time it takes to compute the model, as well as by the goal of the simulation experiment, e.g. the required accuracy of the model results. Our intelligent configuration system is developed in two steps: first it gathers and interprets information that enables us to select a certain solver (e.g. input data, parameter values, and properties of the differential equations); in a second step it generalizes this information from one model, so that it can be used to select solvers for other models that resemble the training model. Both steps are developed using machine learning techniques.

Keywords: Numerical techniques; Simulation; Executable Models; Intelligent solver configuration.

1. INTRODUCTION

The models that we consider are systems of initial value Ordinary Differential Equations (ODE) or initial value Differential Algebraic Equations (DAE). In this contribution an “initialized model” comprises the model structure, the values of the parameters, the values of the initial conditions and the dynamic input data. A “simulation experiment specification” stands for an initialized model together with a certain solver and its settings.

A wide range of numerical software libraries is available for solving these systems, e.g. SUNDIALS (Hindmarsh et al. [2005]) and ODEPACK (Hindmarsh [1983]). These libraries implement several numerical methods (also known as solvers), for example: CVODE, IDA, LSODE. .
An environmental modeller who wants to select an appropriate solver for an initialized model must have a good insight into the structure of the model and a thorough mathematical background on numerical methods. Moreover, knowledge of the effect of the values of certain model parameters, initial conditions and dynamic input data on the behaviour of the solution trajectory is also important. Once a solver is chosen the configuration of that solver is again not straightforward and can only be done by a scientist with expert insight in numerical mathematics and the initialized model.

In the past, modellers also needed extensive programming skills to be able to implement their models in a conventional programming language, and the programming language they used was often inadequate for describing a model. Luckily, several powerful high-level modelling languages such as MSL (Vanhooren et al. [2003]) and Modelica (Modelica Association [2005]) exist nowadays. Furthermore, user-friendly modelling and simulation platforms were developed that help

engineers to create models through a graphical user-interface such as WEST® (Tornado kernel) (Vanhooren et al. [2003]) and The Open Modelica Project (PELAB [2005]).
The Tornado kernel (Claeys et al. [2006b]) is a portable and versatile C++ kernel for modelling and virtual experimentation. It runs on Linux and Windows platforms and allows for model building and running full-fledged virtual experiments. Tornado supports compound models. A compound model comprises one top model and several sub-models. A compound model is represented as a hierarchical tree which upper layer is the top model and which lower layers are sub-models. Sub-models can be atomic models or compound models. Fig. 1 shows a 2-layered compound model with atomic sub-models.

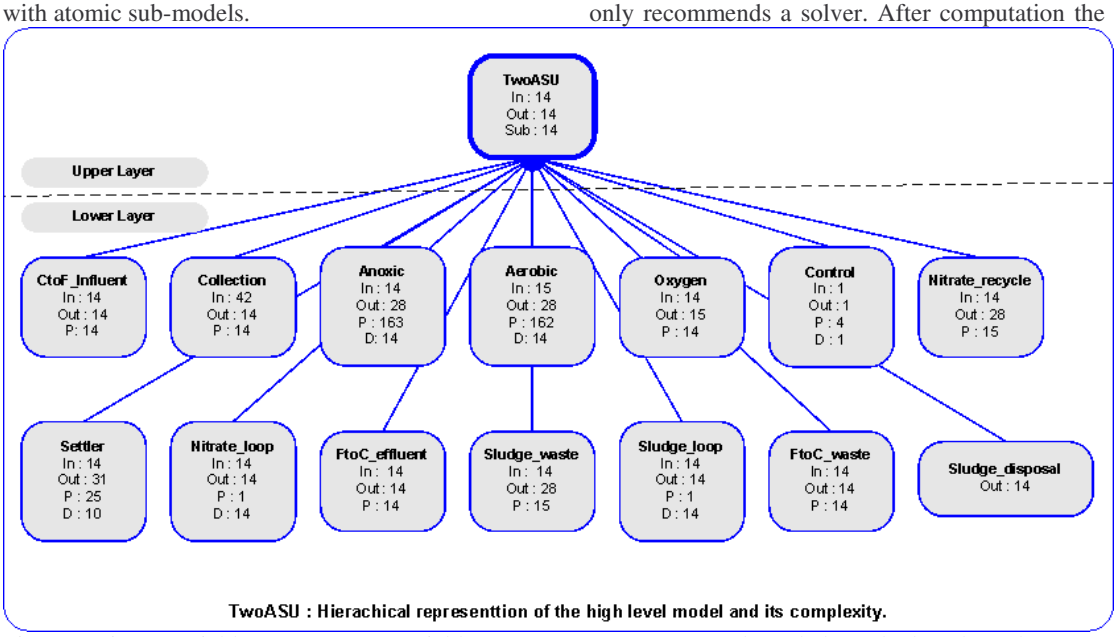

Fig. 1: In: input variables, Out: output variables, P: parameters, D: derived state variables.

The model building takes place in a modelling environment: a high-level model written in MSL or Modelica is compiled with a model compiler that produces an executable model (C) and a symbolic model (XML). The experimentation environment enables a user to run a virtual experiment. For the execution of a virtual experiment an executable model, a symbolic model, a virtual experiment specification file (XML) and dynamic input data are combined.
Typhoon (Claeys et al. [2006a]) is a extension for Tornado that can distribute the load of a complex virtual experiment over a number of (unused) work nodes. These work nodes can have different hardware and operating systems. The architecture of Typhoon is based on a master and slave principle. A complex virtual experiment is sent to a master, which decomposes the virtual experiment into atomic experiments and sends these as jobs to all slaves. The slaves execute the jobs and send the results back to the master. Tornado supports pluggable solvers, and the choice and settings of a solver are specified in the specification file of the virtual experiment. The system that helps users with the choice of the solver and that is presented in this paper, has not been integrated yet.

Some stand-alone systems that give advice on the intelligent configuration of a solver have already been developed, e.g. ODEEXPERT (Kamel and Ma, [2003]) and PHYTHIA (Weerawarana and Houstis [1996]). ODEEXPERT uses textual parsing to determine some properties of the ODEs and performs some automatic tests. The system only recommends a solver. After computation the user cannot acknowledge whether the solver was appropriate or not, preventing knowledge acquisition. PHYTHIA asks for information on the characteristics of the equations to make a decision on which algorithm to use. PHYTIA requests detailed information on the characteristics of the equations from the user.

In this paper an attempt is made to design a system that intelligently configures a numerical solver for an initialized model to be solved by the Tornado simulation kernel. Our system does take the acknowledgment of the user into account and automatically extracts detailed information on the initialized model without any user intervention. The acknowledgment of the user is useful when the solution trajectory predicts values (= mainly concentrations in the case of environmental models) that are physically impossible.
Our system makes decisions on the basis of experiences from the past. It takes advantage of the fact that complex virtual experiments can run

in a distributed environment. The detailed monitoring of a cluster of several work nodes can tell us the do's and don'ts for the selection of a solver for a certain initialized model. Data on simulations is made persistent, analyses of this data and the suitability of the solution helps the system to recommend a numerical method when a user wants to start a new simulation.

The design of our system can be divided into two parts. In the first part the automatic extraction of the features of the initialized model and the selection of the best solver for that model and closely related models is treated. The second part handles the generalization of the results from the first part. Thanks to this generalization our system is able to suggest a solver for an initialized model that differs significantly from all models that were already processed (= unseen models).

This contribution mainly describes the first part of the process of building the intelligent configuration system. Analysis, design and implementation of the second part are planned for the future.

2. ANALYSIS OF THE PROBLEM

Solving models as the ones evaluated here boils down to solving an Initial Value Problem (IVP). The model seen as an IVP has a unique solution for certain values of the parameters, certain initial conditions and dynamic input data. The computation time of a simulation experiment specification is influenced by the behaviour of the solution trajectory.

As an example, let us consider the Vanderpol model where the solution is influenced by only one parameter. The Vanderpol equation is a model of an electronic circuit that appeared in very early radios. This circuit pumps up small oscillations, but drags down large oscillations. This behaviour is known as a relaxation oscillation. The equations that describe this system are:

$$\begin{aligned} dy_1/dt &= y_2 \\ dy_2/dt &= \mu * (1 - y_1 * y_1) * y_2 - y_1 \end{aligned} \qquad (1)$$

where μ is a parameter that affects the system's non-linearity. For μ equal to zero, the system is actually just a linear oscillator. As μ grows the non-linearity of the system can no longer be ignored. When μ becomes large, the cycle becomes stiff.

When μ is equal to zero a non-stiff solver that uses Adams Moulton together with Functional Iteration is a good choice. When μ becomes larger a stiff solver that uses Backward Difference Formulas instead of Adams Moulton is a much better choice. Depending on the desired accuracy a Newton Raphson Iteration that uses the complete Jacobian matrix (more accurate) or a Jacobi-Newton Iteration that uses a diagonal Jacobian matrix (less accurate) can be used (Radhakrishnan et al [1993]).
Environmental models contain a considerable number of parameters, derived state variables, input and output variables. The values of all these variables can have a significant impact on the solution trajectory. Fig. 1 shows some aspects of the complexity of a typical (high-level) model used in our studies.
A detailed description of the features (that we considered) that influence the solution and thus the computation time is given in the next sections.

2.1 Model Structure

The features of the model structure that can influence the computation time are:

- The number of parameters.
- The number of algebraic state variables.
- The number of derived state variables.
- The type of the set of differential equations (only ODE and DAE are considered).
- If-statements that appear in the model can introduce discontinuous changes in values of variables.
- The number of operations: additions, subtractions, multiplications, divisions, exponentiations, built-in function evaluations.
- The set of differential equations themselves.

2.2 Experimental Specifications

The experimental specifications that can influence the computation time are:

- The accuracy desired by the user.
- The values of the parameters.
- The initial values of the derived state variables.
- The dynamic input data.
- The integration time interval.

2.3 Numerical Method

Even the numerical method itself can influence the behaviour of the solution trajectory. When the

chosen solver and its settings are appropriate, it will not influence the behaviour of the solution significantly, but when it is incorrect the influence becomes bigger.

A solution curve can diverge from the real solution curve if the chosen step size is too large for the numerical method used to calculate the solution. Every combination of the IVP and a numerical method has a region of stability and the step size that assures convergence is bounded by this region (Asher and Petzolf [1998]).

Stiffness is another property of the behaviour of the solution curves. Scientists often describe a system as stiff if the rate of change of the derived state variables differs widely amongst the derived state variables. Mathematically stiffness depends, in addition to the differential equation, on the desired accuracy, the length of the interval of integration and the region of stability of the numerical method (Asher and Petzolf [1998]).

3 LEARNING BY EXPERIENCE

The purpose of our system is to learn from experience. Reinforcement learning is a machine learning technique that allows this. It optimizes a sequential decision process. An agent repeatedly perceives a certain state of the environment, executes an action, receives a reward because of that action and puts the environment in another state (Sutton and Barto [1998]).

At this moment we let a numerical method be chosen only at the beginning of the simulation, thus this choice cannot change during simulation. Hence, our problem is not a sequential decision process, but we keep the concepts of reinforcement learning and let our system learn from the environment through the analysis of a reward. The reward will augment depending on the suitability of the solution of the virtual experiment calculated with a certain numerical method.

When we use the concepts of reinforcement learning an initialized model can be seen as a state, a simulation experiment specification together with its solution curve is the next state and the action is the choice of the numerical method and its settings. The reward depends on the quality of the solution.
The quality of the solution is determined by the following features:

- The simulation time: The number of time intervals needed to compute the solution were taken.
- The success of the computation: The computation will not succeed when the solver returns an error. For example, if the solution does not converge, most solvers stop their computations and return an error message.
- Acknowledgement by the user: The result can be completely wrong even though the computation was successful. Only a user can determine this.

In our experiments the reward is a weighted sum of these features, where the acknowledgement of the user is very important in case it was negative.
Several simulations on the same model with different solvers and settings enable the construction of a table (see Table1) with one entry (the reward) for each state-action pair. Using this table it is very easy to find the solver that produces the largest reward for a certain initialized model.

	S_1	S_2	S_3	S_4
IM_1	8	2	0	4
IM_2	8	8	7	1
IM_3	2	5	3	0
IM_4	0	0	1	8
IM_5	0	5	8	7

Table 1: An example of a reward table, IM_x stands for a certain initialized model, S_1 for a certain solver and its settings.

We can apply the concept of exploring and exploiting, which is also used in reinforcement learning. During exploration various of numerical methods are tested while the rewards are measured. During exploitation the system uses what it has learned to test the effect of small variations in the settings of the best numerical method on the reward.

4. DATA COLLECTION FROM TORNADO

The information that we want to extract automatically from the simulation experiment specification are the features of the model structure (see paragraph 2.1) and the experimental specifications (see paragraph 2.2). This is conducted as follows.

The model compiler transforms the high-level model (see Fig. 1) into an executable model and a symbolic model. During compilation some optimization can occur and information that is present in the high-level model can be absent from the executable and symbolic model. As only the information in the executable and symbolic model is used to compute the model, we will extract the desired information from the model at that stage.

Tornado's C++ API allows us to extract almost all necessary information (after compilation) from the simulation experiment specification. Only the differential equations themselves are not readily available. To acquire these equations some changes in the model compiler are needed alternatively we can retrieve them by parsing the executable model (C).

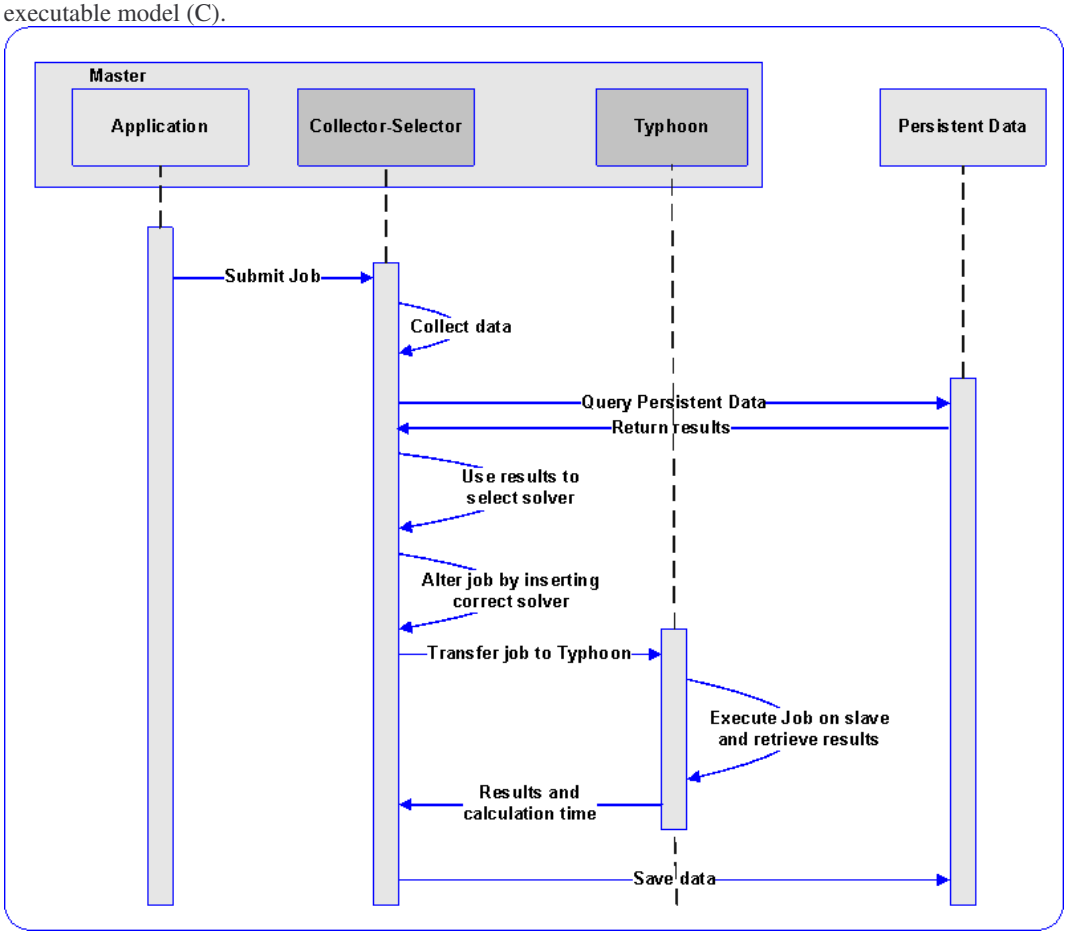

Fig 2: Sequence diagram of the intelligent selection of a solver.

To ensure that the collection of the information does not interfere with the computation time, the data is extracted before the start of the simulation and after the end of the simulation. A Collector-Selector works in unison with Typhoon and takes care of the collection and saving of the data and of the intelligent selection of the solver. It queries the data for a certain initialized model or for a closely related initialized model and automatically builds up a reward table. This table enables the Collector-Selector to select the best solver for a certain initialized model (see Fig. 2). A closely related initialized model is a model with the same model structure as the tested model but with slightly different input variables and parameter values.

5. TEST SIMULATIONS

Before launching the system some test simulations are run. Test simulations allow testing the system and providing initial experience on solver selection. We compute initialized models for which we know the appropriate solver and compare the solver suggested by the system with the appropriate one. Furthermore, data of test simulations for which we do not know the appropriate solver beforehand provides our system with some initial experience that it can use when the actual launching occurs.

Test simulations with perturbed values of specific parameters are designed. As running test simulations is very time-consuming we cannot test the importance of all the factors that influence the computation time. For the design of useful test simulations the inclusion of expert knowledge is inevitable. The selection of the parameters to perturb is done on the basis of domain expert knowledge on the model.

Test simulations are set up as Monte Carlo simulations. Their results can show us the effect that the perturbations have on the computation time for a certain initialized model and solver. An optimization experiment is carried out with regard

to the computation time. The objective of the optimization is to maximize the effect on computation time. These Monte Carlo simulations show us that for a certain model structure some parameters are more influential on the computation time than others. Using these findings we can already deduce some rules for the selection of solvers for a certain model structure.

6. CONCLUSIONS AND FUTURE WORK

Tornado is a flexible and powerful kernel for modelling and virtual experimentation, as well as for the retrieval of the information needed for the intelligent configuration of solvers. Typhoon is a extension to Tornado that enables the distribution of simulations over several work nodes and helps us with the accumulation of information on a plethora of simulations.

The system that we designed takes the effects of the initialized model and the experimental specifications on the solution trajectory into account. Our system accumulates knowledge by learning from past experiences. As time evolves, the strategy for selecting the best solver will converge to the optimal strategy.

Another challenge is lurking, namely: how will we select the best solver for an unseen initialized model with a different model structure than any experiment in our database. In other words what machine learning techniques can we use for the generalization of our results? Can we deduce a paradigm for the selection of solvers?

ACKNOWLEDGEMENTS

Peter Vanrolleghem is Canada Research Chair in Water Quality Modelling

REFERENCES

Ascher U.M & Petzold L.R., *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, Philadelphia, 1998.

Claeys F, Chtepen M., Benedetti L., Dhoedt B. & Vanrolleghem P.A, Distributed virtual experiments in water quality management, *Watermatex2004, 2004Water Science and Technology*, 53(1), pp. 297-305, 2006a.

Claeys F., De Pauw D.J.W., Benedetti L. & Vanrolleghem, P. A., Tornado: A versatile and efficient modelling & virtual experimentation kernel for water quality systems, in: *Proceedings of the iEMSs 2006 conference*, Burlington VE, USA, 2006b.

Hindmarsh A.C., ODEPACK, A systematized collection of ODE solvers, in *Scientific Computing*, Stepleman R.S et al. (eds.), North-Holland, Amsterdam, 1983 (vol. 1 of *IMACS Transactions on Scientific Computation*), pp. 55-64. Also available as LLNL Report UCRL-88007, August 1982.

Hindmarsh A.C, Brown P.N., Grant K.E., Lee S.L., Serban R., Shumaker D.E. & Woodward C.S., SUNDIALS: Suite of non-linear and differential/algebraic equation solvers, *ACM Transactions on Mathematical Software (TOMS)*, 31(3), pp. 363-396, 2005.

Kamel M.S., Enright W.H. & Ma K.S., ODEXPERT: An expert system to select numerical solvers for initial value ODE systems, *ACM Transactions on Mathematical Software*, 19(1), pp. 44-62, 2003.

Modelica Association, *Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling, Modelica Language Specification*, Version 2.2, February 2, 2005.

Radhakrishnan K. & Hindmarsh A.C, Description and use of lsode the livermore solver for ordinary differential equations, Available from http://gltrs.grc.nasa.gov/reports/1993/RP-1327.pdf, 1993.

Saltelli A., Chan K. & Scott E.M, *Sensitivity Analysis*, John Wiley & Sons publishers New York, Wiley Series in Probability and Statistics, 2000.

Sutton R.S. & Barto A.G., *Reinforcement Learning*, MIT Press, Cambridge, 1998.

The Open Modelica Project. Available from http://www.ida.liu.se/~pelab/modelica/OpenModelica.html, PELAB, IDA, Linköping University, Sweden, 2005.

Vanhooren H., Meirlaen J., Amerlinck Y., Claeys F., Vangheluwe H. & Vanrolleghem P.A. WEST: Modelling biological wastewater treatment, *Journal of Hydroinformatics*, 5(1), pp. 27-50, 2003

Weerawarana S., Elias N.H. & Rice J.R., PHYTIA: A knowledge-based system to select scientific algorithms, *ACM Transactions on Mathematical Software*, 22(4), pp. 447-468, 1996.