18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

Hierarchical modeling of complex systems: A hybrid approach combining the best of flattening and component-based modeling

Claeys, F.H.A.[1] **and P.A. Vanrolleghem**[2]

[1] *MOSTforWATER NV, Sint-Sebastiaanslaan 3a, B-8500 Kortrijk, Belgium*
[2] *model*EAU, *Université Laval, Québec, QC, G1K 7P4, Canada*
Email: fc@mostforwater.com

Abstract: In software tools for modeling and simulation of complex systems described through ODEs (*e.g.* those occurring in the scope of water quality research), hierarchical composition is often adopted as a mechanism to handle complexity. Currently, two competing techniques are used with regard to the generation of executable code from hierarchical models: component-based modeling and flattening.

In component-based modeling, hierarchical models are constructed on the basis of self-contained binary components representing atomic sub-models. The manner in which these binary components are implemented is usually of no importance, as long as each component implements a specific well-defined inter-component interfacing protocol. A prime example of a tool that supports component-based modeling is MATLAB’s Simulink toolbox.

The more recent flattening paradigm is typically adopted in the scope of high-level – often object-oriented – modeling languages such as Modelica and MSL. Non-executable, high-level model descriptions are processed by a model compiler in order to produce efficient, low-level, binary executable model code. During this process, the original composition and inheritance hierarchies are resolved in order to produce one large flat set of equations that offers potential for code optimization (*i.e.* automatic reduction of complexity). Examples of tools that adopt flattening are Dynasim’s Dymola, the OSMC’s OpenModelica and MOSTforWATER’s Tornado.

Flattening is often advocated as an approach that is superior to component-based modeling and hence alleviates the need for the latter. However, in practice both approaches have their own distinct merits. Flattening allows for constructing highly efficient monolithic executable models, however the model compilation process tends to be prohibitively time-consuming for large models, and may even lead to depletion of memory in extreme cases. Component-based modeling on the other hand leads to less efficient executable models (due to inter-component communication overhead and the lack of potential for optimization), but tends to be more scalable for large coupled models.

In this article, it is argued that a software tool for modeling and simulation of complex systems that aims to be versatile, needs to provide for both flattening and component-based modeling in a fully transparent manner. Depending on the goal of the modeling exercise, *e.g.* design versus optimization of a treatment plant, either flattening or component-based modeling will be more suitable. For instance, for the design of a new plant, component-based modeling will be advantageous as it causes little overhead when the engineer makes a sequence of modifications to the coupled model representing the plant. However the simulation of each of his designs will be relatively slow. In contrast, in the scope of optimization of an existing plant, one can afford to spend more time on the generation of an efficient executable model through flattening, since this will provide for faster execution of the multitude of simulation runs that is typically required for such exercises.

The article presents the results of the extension of Tornado, an advanced framework for modeling and virtual experimentation originally focused on flattening, with support for component-based modeling. Thanks to this extension, Tornado now allows for transparent application of both modes of operation, on the basis of the same high-level object-oriented model descriptions, and the same virtual experiment descriptions.

Keywords: *Executable models, hierarchical modeling, flattening, component-based modeling, model compilers*

1. INTRODUCTION

In order to master the complexity of nowadays dynamic models, *e.g.* those occurring in the scope of water quality studies (*i.e.* bio-chemical processes in treatment plants, sewers and river catchments), hierarchical composition is often adopted. Through hierarchical composition, new coupled models can be constructed from sub-models that are either atomic, or coupled in their own right. Most software tools that support hierarchical composition allow for graphically constructing coupled models. However, for describing atomic models (that typically contain Ordinary Differential Equations (ODEs) and/or algebraic equations), procedural general purpose programming languages (*e.g.* FORTRAN, C or C++), procedural special purpose languages (*e.g.* MATLAB[1]), procedural modeling languages (*e.g.* ACSL (Mitchell *et al.*, 1976)) or declarative modeling languages (*e.g.* MSL (Vanhooren *et al.*, 2003) or Modelica[2] (Fritzson, 2004)) are used.

In order to be able to use coupled models for model evaluation purposes (*e.g.* time-based simulation), executable code has to be produced from the model description. Historically, the technique that has been used to this end can be labeled as **component-based modeling** (CBM), as it relies on run-time communication between self-contained binary units (components) that implement sub-models. The manner in which these binary components are implemented is usually of no importance, as long as each component implements a specific well-defined inter-component interfacing protocol. A well-known tool that has successfully implemented CBM is MATLAB's Simulink toolbox.

In recent years, another approach related to the generation of executable model code has gained popularity. As it relies on the construction of one optimized, comprehensive executable model (*cf.* Claeys *et al.*, 2007 or Mattsson *et al.*, 1998) in which the boundaries between sub-models are no longer clearly visible, it is commonly referred to as **flattening**. Flattening allows for generating executable models that are more efficient (both in terms of speed and in terms of size) than those that are typically generated through CBM. Unfortunately, flattening is not the ultimate solution to the problem of generating executable code from model descriptions. The automated analysis (named "model compilation") that leads to the generation of code is convoluted, and is hence both time and memory intensive. Examples of tools that adopt flattening are Dynasim's Dymola[3] and the OSMC's OpenModelica[4].

In fact, CBM and flattening can be regarded as complimentary techniques that both have their own distinct merits and drawbacks. In our opinion, a software tool that aims to provide support for different types of modeling exercises should therefore implement both. For instance, for a treatment plant design exercise it is important to be able to quickly analyze simulation results after modifying the coupled model (*e.g.* to study the effect of adding an additional activated sludge tank). Design exercises are therefore best served by CBM, as no complex processing is needed to come to an updated executable model based on a different arrangement of already existing components. On the other hand, an optimization exercise (*e.g.* to find the most efficient operational settings for a plant that satisfy given budgetary constraints) that requires the repeated execution of a multitude of simulation runs, will be best served by an executable model generated through flattening. In this case, the additional time that is to be spent on the generation of the flat model is easily over-compensated by the faster simulation times.

Tornado (Claeys *et al.*, 2006) is an advanced framework for modeling and virtual experimentation (*i.e.* any procedure that involves the execution of executable models, such as simulation, optimization, scenario analysis, *etc*.) in which support for both flattening and CBM has been included. The remainder of this article focuses on the elements of the software architecture of Tornado that have made it possible to support both flattening and component-based modeling in a transparent manner. In fact, Tornado has implemented two versions of CBM, respectively named **compiled component-based modeling** (CCBM) and **interpreted component-based modeling** (ICBM), which both have their own distinct merits.

The following sections respectively provide a general introduction to Tornado, an overview of the architectural aspects of flattening, CCBM and ICBM, and some experimentation results that illustrate the advantages and disadvantages of each approach. Finally, some conclusions are formulated, and a reference to future work is made.

[1] http://www.mathworks.com

[2] http://www.modelica.org

[3] http://www.dynasim.com

[4] http://www.openmodelica.org

2. THE TORNADO FRAMEWORK

Tornado (Claeys, 2008) is an advanced framework for modeling and virtual experimentation with complex environmental systems. It consists of software kernels, toolboxes and APIs (Application Programming Interfaces), and was jointly developed by MOSTforWATER NV (Belgium) and BIOMATH (Ghent University, Belgium), originally to support the WEST® simulator for wastewater treatment plants (Vanhooren *et al.*, 2003).

Tornado implements a wide variety of virtual experiment types ranging from atomic (non-decomposable) experiments such as simulation and root finding, to compound experiments such as optimization, scenario analysis and Monte Carlo analysis. Models can be described in the MSL and/or Modelica modeling languages, both of which are high-level declarative object-oriented languages. Tornado has been used for large-scale studies and has been applied to cluster and grid computing infrastructures (*e.g.* in the scope of the EU project CD4WC[5], where a total of 14,400 simulation experiments had to be executed, with an average execution time of 30 min each (Benedetti *et al.*, 2008)).

3. FLATTENING

Flattening was the first technique for generating executable models that was implemented in Tornado. It is performed by a model compiler that reads high-level model descriptions, resolves the composition and inheritance hierarchies, and produces a monolithic executable model. This executable model representation (which is named Model Specification Language - Executable or MSLE in the scope of Tornado) is made up of two parts:

- The **computational information** is a block of C code that represents the actual algebraic and/or differential equations. Equations from sub-models and base models are analyzed (with respect to the variability of the left-hand side variables), sorted (based on variable dependencies) and grouped into 4 functions:

    - *ComputeInitial*: Sorted set of equations that are to be computed before the start of the simulation process. The left-hand sides (LHS) of these equations are typically parameters and initial values of derived state variables.

    - *ComputeState*: Sorted set of equations that contribute to the state of the system and are hence to be computed at every (major or minor) simulation time point. The LHS of these equations are algebraic worker variables and derivatives.

    - *ComputeOutput*: Sorted set of equations that do not contribute to the state of the system and therefore only need to be computed whenever output is required.

    - *ComputeFinal*: Sorted set of equations that are to be computed after the end of the simulation process. The LHS are typically algebraic variables that represent objective values, to be used in the scope of an encapsulating optimization or scenario analysis loop.

  Parameters and variables used in the computational information are actually elements of large, flat arrays that are constructed by the model compiler on the basis of the union of all parameters and variables that occur in the original high-level code. However, it needs to be noted that there is no one-to-one mapping between the computational information's equation and parameter/variable sets, and those of the original high-level model (this issue is handled by the symbolic information, see below). For, through various types of optimizations (Claeys *et al.*, 2007), the model compiler will attempt to reduce the final equation and parameter/variable sets. The computational information is directly used by the integration solver.

- The **symbolic information** is an XML-based representation of the mapping between the original, hierarchical model information, and the final, flattened information as it occurs in the computational information. Next to this, the symbolic information also contains various meta-information items for

[5] http://www.tu-dresden.de/CD4WC

each parameter and variable, such as a description, default value, range of validity, unit, *etc*. The symbolic information is mainly intended for user interaction, and does not directly influence the simulation process.

Flattening produces highly efficient models, both from the point of view of simulation speed, as from the point of view of size. However, depending on the complexity of the model (in terms of composition hierarchy, inheritance hierarchy, number of equations, *etc*.), a number of undesirable situations may occur during the flattening process:

- Overly long processing times, ranging from tens of minutes to hours.
- Overly large consumption of memory during processing, ranging from tens of Megabytes to Gigabytes.
- Generation of overly large chunks of computational information that fail to compile with regular C compilers.

It must be noted that in relation to the last point, substantial progress was made since earlier versions of Tornado, where both computational and symbolic information were represented in C++ (Claeys, 2008). The fact that Tornado now uses XML for symbolic information has increased scalability and has reduced the potential for incompatibility issues between different executable model versions. However, in extreme cases, computational and/or symbolic information may still grow so large that they cannot be handled anymore.

As an example, Figure 1A shows the composition graph of a coupled model named *Model1* that is composed of an atomic model named *Model2* and a coupled model named *Model3*. The latter is composed of two atomic models respectively named *Model4* and *Model5*. After flattening (Figure 1B), one single monolithic executable model is generated that contains equation sets containing equations originating from model *Model1* through *Model5*.

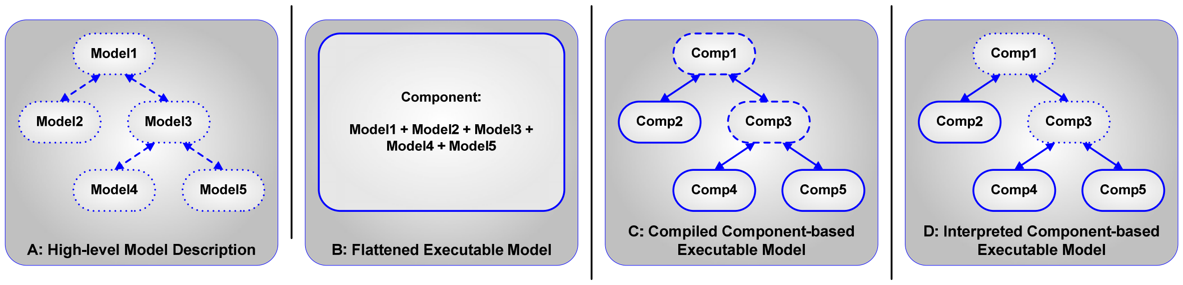
Model1
Model2
Model3
Model4
Model5
A: High-level Model Description
Component:
Model1 + Model2 + Model3 + Model4 + Model5
B: Flattened Executable Model
Comp1
Comp2
Comp3
Comp4
Comp5
C: Compiled Component-based Executable Model
Comp1
Comp2
Comp3
Comp4
Comp5
D: Interpreted Component-based Executable Model

Figure 1. Flattening vs. CCBM and ICBM.

4. COMPONENT-BASED MODELING (CBM)

As it gradually became clear that flattening – although very powerful as such – still has its limitations, support for component-based modeling was added to the Tornado framework. In relation to this, two major requirements were put forward:

- *Transparency with respect to model descriptions*: it should be possible to build flattened and component-based executable models from the exact same high-level model descriptions.
- *Transparency with respect to experiment descriptions*: it should be possible to use the exact same virtual experiments (simulation, optimization, *etc*.) on executable models generated from the same high-level description, regardless of the application of flattening or CBM.

In order to realize CBM, two types of entity representations are required: one for the components that make up the coupled model, and one for the coupled model itself. The component representation is trivial in the scope of Tornado, as components can simply be generated through traditional flattening. Indeed, instead of applying flattening to a coupled model that uses high-level sub-model representations from a model library, one can simply apply flattening to each of these high-level sub-model representations individually. As a result, a library of binary components is created that has a one-to-one mapping with the models from the original library of high-level language models. The fact that traditional flattening of coupled models and the generation of components can be done on the basis of the same high-level model descriptions satisfies the first requirement that was put forward for CBM.

The representation of coupled models in a CBM context is less trivial. In fact, it was decided to implement two different alternatives for Tornado: compiled CBM and interpreted CBM. The difference is that the coupled model in case of the first is compiled, while in the case of the latter it is interpreted at run-time by the simulation kernel.

4.1. Compiled Component-based Modeling (CCBM)

A compiled coupled model for CBM appears to the outside as any other executable model. As a result, the second requirement that was put forward for CBM is satisfied, as the same experiment descriptions can be applied to both flattened and CCBM executable models (since they appear the same to the outside). Internally, compiled coupled models for CBM differ substantially from flattened models. In the case of the first, the ComputeInitial, ComputeState, ComputeOutput and ComputeFinal functions do not contain any equations, but simply call the corresponding Compute functions of each underlying component in an appropriate sequence (the ComputeInitial function of a coupled model calls the ComputeInitial of each of its components, *etc*.). Also, the values of output variables from one component are transferred to the corresponding input variables of the next. The component call sequence is static and is determined when the coupled model code is generated. It depends on the input/output dependencies between sub-models and is related (but not equal) to the sequence of equations in the case of flattening. In order to maximize the efficiency of transferring output data to input data, de-referenced C pointer assignment is used. These pointers refer to the data containers, within the components' computational information, which correspond to the variables that are to be assigned. As a static pointer lookup table can be constructed during initialization of the executable model, very little time is lost due to data transfer during the simulation process itself.

The advantage of compiled CBM is that it adds little overhead to the execution of the Compute functions of the individual components. However, for creating the coupled model, C code generation as well as compilation is still required. Evidently, the complexity of this code is far less than in the case of flattening, and scalability is therefore much better. Typically, the time needed to generate a compiled coupled model for CBM is less than 3s.

CCBM is especially useful in design exercises, where a user wants to be able to quickly evaluate the effects of interactively modifying a coupled model, *e.g.* when adding or eliminating a reactor of a treatment plant. For these types of interactive processing, a number of seconds to be spent on compilation before the start of a (relatively efficient) simulation can easily be afforded.

For the composition graph of Figure 1A, Figure 1C shows that the CCBM executable model consists of 3 components (*Model2*, *Model4* and *Model5*) that each have been generated through flattening, and 2 compiled coupled models (*Model1* and *Model3*). The Compute functions of *Model1* respectively call the corresponding Compute functions of *Model2* and *Model3* and perform the required data transfer. Similarly, the Compute functions of *Model3* respectively call the corresponding Compute functions of *Model4* and *Model5* and perform the required data transfer.

4.2. Interpreted Component-based Modeling (ICBM)

Although very short, there are situations in which one cannot afford to spend 2 or 3s on the generation of a compiled coupled model. In the case of the EU project AquaFit4Use[6] for instance, automated optimization of water networks is required. These water networks are modeled in Tornado as couplings of simple algebraic steady-state models. Whereas the computational complexity of each model is very low, care must be taken to efficiently deal with the large number (10,000 or more) of coupled model alternatives that are to be evaluated. Therefore, Tornado was recently provided with so-called interpreted CBM.

In ICBM, Compute functions of underlying components are again called in an appropriate sequence. However, this is not done from a compiled chunk of custom executable model code, but rather under run-time control of the simulation kernel. Before the simulation starts, the simulation kernel is provided with a sorted list of components, and information on all input/output links. On the basis of this, the kernel is able to loop over all components in the appropriate sequence at run-time, call the appropriate Compute methods, and perform the required assignments. Evidently, the mere fact that the simulation kernel has to loop over the list

[6] http://www.aquafit4use.org

of components for invoking Compute methods, as well as over the list of input/output links to perform assignments, incurs a loss of performance with respect to CCBM, where these lists were expanded during the code generation.

Figure 1D shows a similar situation as in Figure 1C for the example that we have considered. However in the case of ICBM, *Model1* and *Model3* do not have a persistent, compiled representation, but reside entirely in memory.

5. RESULTS AND DISCUSSION

Flattening, CCBM and ICBM were applied to a number of cases (all wastewater treatment plant models (*cf.* Claeys *et al*., 2007), except for *PredatorPrey*, which is a system dynamics model) with varying complexity. Table 1 lists the number of components for each case, as well as the number of parameters, algebraic variables and derived variables of the corresponding flat models.

	#Comps	#Params	#AlgVars	#DerVars
PredatorPrey	10	12	27	2
ASU	9	652	612	20
Benchmark	16	969	1111	108
Orbal	30	6600	3265	250
SBR	8	258	816	154
Umbilo	19	840	1620	70

Table 1. Complexity of test cases.

Table 2 respectively lists the compilation time, simulation time and relative speed of simulation for flattening, CCBM and ICBM, applied to each case. Each simulation was run with an advanced variable step size integrator (CVODE[7]) as well as with a traditional fixed step integrator (Runge-Kutta 4). As expected, flattening results in the longest compilation times, and shortest simulation times. ICBM requires no compilation, but has the longest simulation times. CCBM is a compromise between both extremes as it combines relatively short compilation times, with relatively short simulation times.

	Compilation time (s)				**Simulation time (s)**			**Relative speed of simulation**		
	Flat	**CCBM**	**ICBM**	**Integrator**	**Flat**	**CCBM**	**ICBM**	**CCBM vs. Flat**	**ICBM vs. Flat**	**ICBM vs. CCB**
PredatorPrey	1	1	0	*CVODE*	1	1	1	100	100	100
				RK4	17	28	108	61	16	26
ASU	2	1	0	*CVODE*	3	3.5	5.5	86	55	64
				RK4	36	41	59	88	61	69
Benchmark	3	1	0	*CVODE*	4	12	14	33	29	86
				RK4	143	147	197	97	73	75
Orbal	40	1	0	*CVODE*	29	33	47	88	62	70
				RK4	614	669	1034	92	59	65
SBR	5	1	0	*CVODE*	1	1	1	100	100	100
				RK4	6	7	7	86	86	100
Umbilo	5	1	0	*CVODE*	39	112	153	35	25	73
				RK4	4678	5521	7231	85	65	76

Table 2. Compilation and simulation times for test cases.

Interestingly, depending on the type of integrator used, the simulation times of CCBM appear to be either closer to those of flattening, or closer to those of ICBM. In order to explain this phenomenon, it is important to also look at the simulation results, *i.e.* the question whether the simulated values are the same in case of application of flattening, CCBM or ICBM. As CCBM and IBCM are based on exactly the same logic (same sequence of components, same variable assignments, *etc*.), it is not surprising that the simulation results of CCBM and ICBM are always the same. However, this does not apply to the results obtained through flattening. Flattening leads to monolithic Compute functions that contain equations that do not necessarily have the same sequence as in the case of CCBM/ICBM. As a result of the inherent limited numerical precision of floating point values, these differences in execution sequence lead to small differences (less than

[7] https://computation.llnl.gov/casc/sundials/main.html

1%) in the simulation results. As the simulation results are different, the number of steps taken by variable step size integrators such as CVODE may also be different. As we have observed that the number of simulation steps is often less in the case of flattening (e.g. for *Benchmark* and *Umbilo*), it is not surprising that the discrepancy between the simulation times of flattening vs. CCBM is larger in these cases. As a matter of fact, since the simulation results influence the number of integration steps, and hence the simulation time, one should only consider fixed step integrators to evaluate the inherent efficiency of flattening, CCBM and ICBM. By only considering the RK4 integrator, we see that in this case CCBM offers on average 84% of the performance of flattening, whereas ICBM only offers 60% of that performance.

6. CONCLUSION

After a number of years of experience with the application of flattening, we have concluded that – albeit very advantageous in the case of compound virtual experiments requiring many simulation runs – flattening does not offer the flexibility required in the scope of the interactive design of treatment plants and/or the automated optimization of plant layouts. Therefore, any software environment that attempts to provide support for various types of modeling exercises including design and optimization, should allow for the application of CBM next to flattening, in a fully transparent manner. In the scope of the Tornado framework, we have succeeded in successfully implementing two flavors of CBM (compiled and interpreted) next to the support for flattening that was already present, fully taking into account transparency-related requirements.

In future work, the effects of the differences in sorting of monolithic equations sets (flattening) versus components (CBM) will be studied further. As such, we hope to obtain more insight into the causes of the lower number of simulation steps that is required by variable step size integrators in the case of flattening, versus the number that is required by CBM.

ACKNOWLEDGMENTS

Peter A. Vanrolleghem holds the Canada Research Chair in Water Quality Modeling.

REFERENCES

Benedetti, L., D. Bixio, F. Claeys, and P.A. Vanrolleghem (2008), Tools to support a model-based methodology for emission/immission and benefit/cost/risk analysis of wastewater treatment systems. *Environmental Modelling and Software*, 23(8):1082-1091.

Claeys, F., D. De Pauw, L. Benedetti, I. Nopens, and P.A. Vanrolleghem (2006), Tornado: A versatile efficient modelling & virtual experimentation kernel or water quality systems. In *Proceedings of the iEMSs 2006 Conference*, Burlington, VT, July 9-13 2006.

Claeys, F.H.A., P. Fritzson, and P.A. Vanrolleghem (2007), Generating efficient executable models for complex virtual experimentation with the Tornado kernel. *Water Science and Technology*, 56(6):65-73.

Claeys, F.H.A., A generic framework for modeling and virtual experimentation with complex environmental systems (2008). PhD thesis. Department of Applied Mathematics, Biometrics and Process Control (BIOMATH), Ghent University, Belgium. January 2008. ISBN 978-90-5989-223-1. pp. 303.

Fritzson, P. (2004), *Principles of Object-Oriented Modeling and Simulation with Modelica*. Wiley-IEEE Press, February 2004. ISBN 0-471-47163-1.

Mattsson, S.E., H. Elmqvist, and M. Otter, Physical system modelling with Modelica (1998). *Control Engineering Practice*, 6:501-510.

Mitchell, E.E.L., and J.S. Gauthier (1976), Advanced continuous simulation language (ACSL). *Simulation*, 26(3):72-78.

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