Automating Dynamic Decoupling in Object-Oriented Modelling and Simulation Tools

Alessandro Vittorio Papadopoulos   Alberto Leva

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy,
{papadopoulos,leva}@elet.polimi.it

Abstract

This manuscript presents a technique that allows Equation-based Object-Oriented Modelling Tools (EOOMT) to exploit Dynamic Decoupling (DD) for partitioning a complex model into "weakly coupled" submodels. This enhances simulation efficiency, and is naturally keen to parallel integration or co-simulation. After giving an overview of the problem and of related work, we propose a method to automate DD by means of a novel structural analysis of the system – called "cycle analysis" – and of a mixed-mode integration method. Also, some considerations are exposed on how the presented technique can be integrated in EOOMT, considering as representative example a Modelica translator. Simulation tests demonstrate the technique, and the realised implementation is released as free software.

Keywords dynamic decoupling, model partitioning, efficient simulation code generation

1. Introduction and Motivation

Equation-based Object-Oriented (EOO) modelling languages are known to possess a number of interesting advantages, and in the context of this work, two are particularly relevant. First, the EOO modelling paradigm is inherently suited for building modular, multi-physic models. Second, the model designer has not to take care of how the system will be simulated, just focusing on how to write the equations of its components. In one word, with EOO Modelling Tools (EOOMT) one handles the complete model by just aggregating components and acting on them. The translator included in typical EOOMT is then in charge of manipulating all the gathered equations, and producing efficient simulation code [5, 8].

As long as the obtained simulation efficiency is sufficient, the possibility of managing complexity at the component level has practically no cost. However, there are some cases where to achieve the desired efficiency, approximations need introducing, and EOOMT are neither meant nor suited for that. As will be discussed in this work, approximations can be introduced either by altering some equations in some components, or by acting on the numerical solution of the complete model. And while with EOOMT the first action is natural, the second is not at all. This rules out several powerful approximation techniques aimed at enhancing simulation speed, e.g., the Dynamic Decoupling (DD) one [2–4] treated herein.

To enter the subject, it is convenient to specify why introducing approximation at the level of the solution is "unnatural" in EOO modelling. The main reason is that the possibility/opportunity of doing so depends on properties of the *whole* model, not of the individual components, and in the typical toolchain of EOOMT no user interaction is envisaged at that point. Moreover, assuming that the use of any approximation technique requires some parameters, it is necessary to provide the user with the necessary information to give them a value, and to accept his/her choices, in a comprehensible manner, manageable by people who are more experts of physics than of simulation theory.

In this work we refer as "EOO Modelling Tool" to a Modelica translator, to allow exemplifying the (more general) presented ideas. For a Modelica translator, the EOO modelling toolchain can be synthetically depicted as in Figure 1.

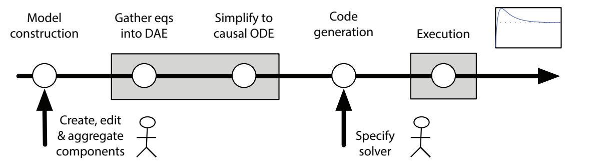

Figure 1. The typical EOO modelling toolchain.

For our purposes, said toolchain has to be extended – and in some sense "opened" – as suggested in Figure 2, introducing some (clearly optional) automatic system-wide analysis, and taking care of having the user interact with simple enough information despite operating at the whole system level. In this work we present a solution assuming that the desired type of approximation is DD, therefore tailoring the analysis and the use of the produced information to that case, but nonetheless the way of acting on the toolchain is general with respect to the approximation type.

5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. 19 April, 2013, University of Nottingham, UK.

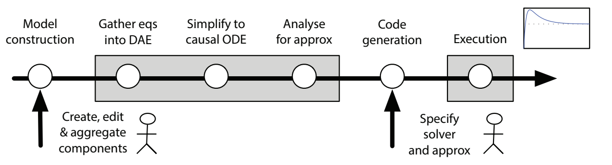

Figure 2. Extending the EOO modelling toolchain.

The rest of the manuscript is organised as follows. Section 2 reviews some relevant literature and provides a more detailed motivation for the presented work, specialising to the DD technique. Section 3 is devoted to the DD technique, and particularly to how it is structured – into an analysis and a simulation part – so as to be applicable to the addressed context. A few illustrative examples are then reported in Section 4. Some more general considerations are made in Section 5 on how to integrate the presented technique in modern EOOMT, ending with some details on an implementation offered as free software to the community. Finally, Section 6 draws some conclusions, and sketches out future research developments.

2. Related Work and Contribution

In this section, we motivate the presented research by relating DD to other approximation techniques for improving simulation efficiency, with specific emphasis on their applicability and convenience in EOOMT.

Referring to Figure 1, the chain of operations of EOO modelling translators, from component equations to simulation code, can be broadly divided into two parts.

The first part, which we call *acting on the continuous-time equations*, transforms the DAE system coming from the flattening phase, into a causal ODE one. This can be done without altering the equations' semantic, by resorting to techniques, such as the Tarjan algorithm, alias elimination, index reduction and so forth [5]. The same operation can also be done by accepting some semantic alteration in exchange for an efficiency improvement. The techniques of election for such a purpose are, e.g., MOR [1] or scenario-based [13] approximations.

The second part of the EOO modelling toolchain, which we call *acting on the discrete-time solution*, consists of taking the mentioned ODE model as the basis to generate suitable routines that, once linked to the numeric solver of choice, result in the required simulation code. Assuming that acting on the discrete-time solution is done "correctly", i.e., preserving numerical stability, also in this case two ways of operating can be distinguished. The first way does not alter the solution semantic, and the chosen discretisation method is applied as is. In this case, errors in the solution only come from the inherent imperfection of the considered method. The second way conversely alters the semantic of the discrete-time system, by deliberately deviating from the natural application of the chosen discretisation method. Notice that most of the co-simulation techniques fall in this class naturally.

In this manuscript we concentrate on this last type of operation, where a technique of election is DD [2, 14]. For the purpose of this section, suffice to say that this technique aims at partitioning the monolithic system into submodels, based on time-scale separation. The method is particularly of interest – as will be detailed better in Section 3 – because it can be divided into two well separated phases: an analysis part performed on the overall model, and a simulation part that either can be monolithic or makes use of co-simulation technique.

To motivate the choice of focusing on DD, we now briefly consider the major possible alternatives, and evidence the advantages of our proposal.

2.1 Alternatives Approaches

As already stated, among the techniques that act on the continuous-time equations, MOR ones are the most adopted, and there exists a vast literature on the matter. MOR is based on the idea of approximating a certain part of a high-dimensional state space of the original system with a lower-dimensional state space, performing a projection. Very roughly speaking, the main differences among MOR techniques come from the way the projection is performed.

Most MOR techniques have been developed for linear systems [1], and this hampers their application to object-oriented models, that usually are high-dimensional and nonlinear; developing effective MOR strategies for nonlinear systems is quite a challenging and relatively open problem.

In the literature, some extension to the nonlinear case are present, e.g., based on linearisation or Taylor expansion [7], or bilinearisation [15], as well as functional Volterra series expansion [10], followed by a suitable projection. Other interesting extensions worth mentioning are those based on Proper Orthogonal Decomposition (POD) [6], that produce approximate truncated balanced realisations for nonlinear systems [16], often exploiting POD to find approximate Gramians [11]. The main problem with those extensions is that, of the former, practical implementations typically stick to quadratic expansions, strongly limiting the simplification capabilities. As for the latter, the cost of evaluating the projected nonlinear operator often remains very high, and reduces computational performance.

Recently, other works dealing with model reduction specifically conceived for object-oriented models have appeared [12, 13]. The main idea is that one can define some operation to be performed on the nonlinear system, e.g., neglecting a "term", linearise a part of the model, and so on, and use some ranking metrics to identify *a priori* which is the "best" (single) manipulation that can be done on the model. Apparently, the limit of this approach lies in the fact that ranking all the possible manipulation combinations is not feasible. Moreover, there is no guarantee that performing the manipulations in the ranked order will bring to the optimal manipulation. Another problem is the high cost of generating the reduced order models, due to necessity of computing "snapshots" in the time domain, which in turn requires performing numerous simulations of the original nonlinear system. Furthermore, this approach is scenario-based, i.e., the simplified model is guaranteed to be good only for a set of initial conditions, a set of inputs and a time span. If the scenario is changed, the overall manipulation must be performed again, limiting again the applicability of the method.

The old idea of DD has also been recently reconsidered, for example by the Transmission Line Modelling (TLM) approach of [18]. This, however requires that the analyst introduces decoupling by deliberately acting on the model based on his/her intuition. This work conversely aims at having decoupling emerge from an automated analysis of the model.

2.2 A Brief Comparison

Based on the previous discussion, we now point out the advantages of the proposed technique with respect to the analysed alternatives.

In comparison with MOR, our proposal does not alter the state vector, nor does it involve base changes in the state space. Also, instead of attempting to simplify the model in a view to monolithic solution, we go exactly in the opposite direction, as the model is not reduced but *partitioned*. This can in turn be exploited in two ways. One is to ease a monolithic solution, in some sense adapting the model to the used (single solver) architecture. The other is to conversely tailor the solution architecture to the model *as analysed and partitioned by the method*; this can be used to fruitfully employ parallel simulation, or even co-simulation. Finally, the proposed method is naturally keen to be applied in a nonlinear context.

With respect to scenario-based approximations, the most computing-intensive part of the proposal (as will be explained later on) is simply not scenario-based: information related to the considered *scenarii* come into play only at a later stage, and this separation results in lightening the computing effort. Furthermore, the proposal does not alter the model equations, thus being less exposed to the possible unpredictable effects of local modifications at the overall system level.

The next section will delve into details on the proposed technique, thereby providing evidence for the statements made so far.

3. Dynamic Decoupling

Multi-physics systems are usually made of parts evolving within different time-scales. For example, in mechatronic systems a "slow" mechanical part is often controlled by "fast" electric circuits or by a hydraulic drive. The underlying idea of DD is to find a way to separate the different dynamics present in the model and to numerically integrate them with a suitable mixed-mode method, in order to improve simulation efficiency.

DD is composed of two subsequent phases, termed here *analysis* and *decoupled integration*. The former consists in performing an offline structural analysis of the system and in identifying which are the time-scales involved in the model. The latter exploits the information coming from the analysis to improve simulation efficiency.

Both phases can be carried out with multiple techniques. For the analysis phase, we propose here a novel method, called *cycle analysis*, that carries most of the merit for the applicability of the entire technique to the nonlinear case. For the latter, we conversely resort to mixed-mode integration.

3.1 Cycle Analysis

Cycle analysis is based on the idea that explicit integration methods are the most suited to enhance simulation speed [5]. Their computational effort is relatively low and constant, and the number of calculations per step can be easily estimated. However, those methods show their limits when dealing with stiff-systems. For simplicity, in the following we consider the Explicit Euler (EE) integration method, but similar (and less restrictive) results can be obtained for other explicit single-step methods, e.g., Explicit Runge-Kutta of any order.

The main idea of the cycle analysis – and of DD, in general – is that in a causal ODE model, both each variable and each equation can be associated with a characteristic time-scale.

A first possible idea to achieve this is to perform an eigenvalue analysis of the linearised system, and to partition the state space, as spanned by the eigenvectors, on the basis of the corresponding time constants (or natural frequencies). However, this is not always a good idea, because it involves a coordinate transformation; if the linear system involved in the eigenvalue analysis comes from the linearisation of a nonlinear one, the management of that transformation results in additional computations at each integration step. Different criteria for state space partitioning are thus advisable, like that proposed in [17].

Coming to our proposal, consider the state space form of a continuous-time ODE system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$$

that discretised with EE with an integration step h yields

$$\mathbf{x_{k+1}} = \mathbf{x_k} + h \cdot \mathbf{f}\left(\mathbf{x_k}, \mathbf{u_k}\right). \qquad (1)$$

Suppose now that the system is at an asymptotic stable equilibrium, i.e., $\mathbf{x_{k+1}} = \mathbf{x_k}$. If a small perturbation is applied to a single state variable $x_\mathbf{k}$, a transient occurs, and two things may happen:

1. the perturbation affects the other state variables, however without in turn re-affecting $x_\mathbf{k}$;
2. the perturbation, after some integration steps, re-affects $x_\mathbf{k}$.

In the first case, no numerical instability can occur, but in the second case there is a "dependency cycle" among some state variables that may lead to unstable behaviours, depending on how the perturbation propagates.

The proposed method detects the dependency cycles that are present in the system, and defines conditions under which the perturbation cannot lead to numerical instability.

The first step in cycle analysis is to build the dependency digraph $G = (N, E)$ associated with the ODE model. In particular, there is a node $n \in N$ for each state variable, and the set of edges $E \subseteq N \times N$ is formed as

$$e_{i,j} = h \cdot \frac{\partial f_i}{\partial x_j}.$$

In other words, the Jacobian of the model corresponds to the adjacency matrix of the weighted graph. The weights come from (1) and characterise the way the perturbation propagates.

The second step is to detect the set $\mathcal{C}$ of all cycles contained in the digraph G. Unfortunately, the problem of finding all the cycles in a directed graph has complexity $\mathcal{O}\left(2^{|E|-|N|+1}\right)$, as shown by a vast research in the operation research domain [9, 19, 20]. This is of course a limitation with strongly connected graphs, but sparse ones are more common in real applications, especially if weak couplings exist. On the other hand, it is worth stressing that the cycle analysis must be performed only once for a given model, during an offline phase before the simulation is started. According to practical experience, spending some additional time to perform a structural analysis aimed at speeding up the simulation is an acceptable tradeoff, especially when the simulation is run many times. Anyway, in all the performed tests (with up to 100 state variables), the analysis phase always took less than one second on a notebook with a 2.4GHz Intel Core 2 Duo processor and 4GB of RAM.

At this point, for every cycle $c \in \mathcal{C}$ detected in G a *cycle gain* can be defined.

DEFINITION 1. *A* cycle gain μ_c *of a cycle* $c \in \mathcal{C}$ *is*

$$\mu_c = \prod_{x_i, x_j \in c} e_{i,j} = h^L \cdot \prod_{x_i, x_j \in c} \frac{\partial f_i}{\partial x_j}$$

where $e_{i,j}$ *are the edges involved in the cycles and* L *is the length of the cycle.*

The meaning of this gain is related to how the perturbation propagates. Starting from the computed cycle gains, for each cycle an inequality in the form

$$|\mu_c| \leq \alpha \quad \Rightarrow \quad 0 < h \leq \sqrt[L]{\alpha} \cdot \left| \prod_{x_i, x_j \in c} \frac{\partial f_i}{\partial x_j} \right|^{-\frac{1}{L}} \qquad (2)$$

is written, where $\alpha > 0$ is a design parameter of the method, related – as discussed later on – to the required simulation accuracy. Suffice for now to say that lower values of α make the method less keen to consider a certain coupling “weak”.

So far, each cycle has been associated with a constraint on the integration step, i.e., with an upper bound for the integration step which prevents the perturbation from producing unstable behaviours.

Finally, each variable x_i is associated with the most restrictive constraint on $\overline{h}_{x_i}$ among the set of cycles $\mathfrak{C}_{x_i} = \{c \in \mathcal{C} | x_i \in c\}$, i.e., formally

$$\begin{aligned} \overline{h}_{x_i} = \max \quad & h \\ \text{s.t.} \quad & h > 0, \\ & 0 < \overline{h}_i \leq \sqrt[L]{\alpha} \cdot \left| \prod_{x_j, x_k \in c} \frac{\partial f_j}{\partial x_k} \right|^{-\frac{1}{L}}, \forall c \in \mathfrak{C}_{x_i}. \end{aligned}$$

3.2 Decoupled Integration

The second phase of DD is decoupled integration, which exploits the partition coming from cycle analysis in a view to improve simulation efficiency. Among the various possible ways to do so, we consider here the use of a mixed-mode integration method. The underlying idea is that implicit methods are able to simulate stiff systems with larger integration periods, at the cost of solving a nonlinear set of algebraic equations at each step, while explicit ones are better in terms of performance but cannot deal with stiff systems equally well. Having separated the system in (at least) two parts with different time scales, it is possible to use an implicit method for the fast part(s), and an explicit one for the slow part(s), exploiting the advantages of both kinds of integration algorithms.

Coming back to the proposal, the discrete-time system associated with the continuous-time one reads

$$\begin{cases} \mathbf{x}^s_{\mathbf{k}+1} = \mathbf{x}^s_{\mathbf{k}} + h \cdot \mathbf{f}\left(\mathbf{x}^s_{\mathbf{k}}, \mathbf{x}^f_{\mathbf{k}}, \mathbf{u}_{\mathbf{k}}\right) \\ \mathbf{x}^f_{\mathbf{k}+1} = \mathbf{x}^f_{\mathbf{k}} + h \cdot \mathbf{f}\left(\mathbf{x}^s_{\mathbf{k}+1}, \mathbf{x}^f_{\mathbf{k}+1}, \mathbf{u}_{\mathbf{k}+1}\right) \end{cases}$$

showing that the fast and the slow parts are integrated with the Implicit Euler (IE) and the EE method, respectively.

This means that the fast component $\mathbf{x}^f_{\mathbf{k}+1}$ can be computed considering $\mathbf{x}^s_{\mathbf{k}+1}$ as an input. Figure 3 shows the resulting mixed-mode integration scheme.

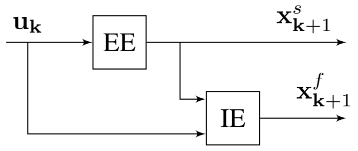

Figure 3. Explicit/Implicit Euler integration scheme.

4. Application Examples

4.1 DC Motor

The DC motor is a very simple example of system with two well separated time scales (electric and mechanic), and can be represented by a third order model in the form

$$\begin{cases} L \cdot \dot{I} &= -R \cdot I - k_m \cdot \omega + u(t) \\ J \cdot \dot{\omega} &= k_m \cdot I - b \cdot \omega - \tau(t) \\ \dot{\varphi} &= \omega \end{cases} \qquad (3)$$

where $L = 3\,\mathrm{mH}$ is the armature inductance, $R = 50\,\mathrm{m\Omega}$ is the armature resistance, $J = 1500\,\mathrm{kg\,m^2}$ is the inertia, $b = 0.001\,\mathrm{kg\,m^2\,s^{-1}}$ is the friction coefficient, and $k_m = 6.785\,\mathrm{V\,s}$ is the electro-motorical force (EMF) constant of the motor. These parameter values correspond to those of a real system. The inputs are the armature voltage, $u(t)$, and the torque load, $\tau(t)$, respectively. In the given example, $u(t)$ is 500 V, and the torque is of 2500 N m.

The cycle analysis leads to the constraints

$$\begin{aligned} I:&\quad h \leq 0.060\\ \omega:&\quad h \leq 0.313\\ \varphi:&\quad h \leq +\infty \end{aligned}$$

Hence, choosing an integration step $h = 0.3$ induces a partition of the system that is natural, as it clearly separates the electric components from the mechanic ones. The constraint associated with φ comes from the fact that there is a pure integral action that does not influence the cycle analysis. Figure 4 shows the simulation results.

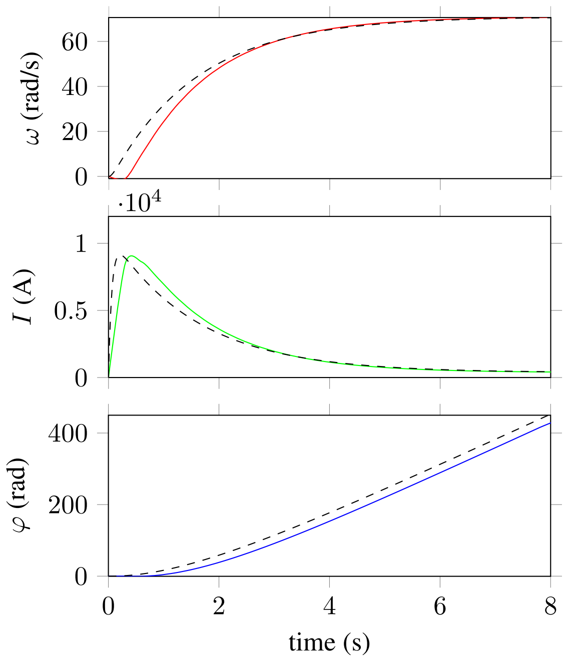

Figure 4. Simulation results of Model (3). Dashed lines represent the real trajectories, while the solid lines are the trajectories obtained with DD.

Table 1 shows the simulation statistics for different integration methods. It is worth noticing that the dimension of the system the Newton iteration has to solve is reduced from 3 to 1 in the mixed-mode method. Notice also that the EE method needs a smaller step size, hence $h = 0.05$ was chosen for numerical stability reasons. Apparently, using DD improves simulation efficiency also in this very simple case.

	Mixed-mode	**BDF**	**IE**	**EE**
# Steps	**28**	136	**28**	162
# Function ev.	**86**	157	**86**	–
# Jacobian ev.	**2**	3	**2**	–
# Fun. ev. in Jac. ev.	**4**	9	8	–
# Newton iterations	**58**	153	**58**	–
# Newton fail	0	0	0	–
Accuracy	**1.118**	–	1.213	10.043
Sim time	**0.04s**	0.05s	0.06s	**0.04s**

Table 1. Simulation statistics for Model (3).

4.2 Counterflow Heat Exchanger

This example refers to a counterflow heat exchanger with two incompressible streams (Figure 5).

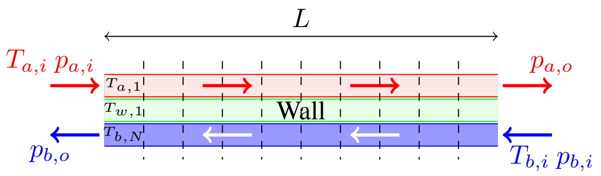

Figure 5. Counterflow heat exchanger scheme.

Both streams and the interposed wall are spatially discretised with the finite volume approach, neglecting axial diffusion in the wall – as is common practice – and also in the streams (zero-flow operation is not considered for simplicity). Taking ten volumes for both streams and the wall, with the same spatial division (again, for simplicity) leads to a nonlinear dynamic system of order 30, having as boundary conditions the four pressures at the stream inlets and outlets, and the two temperatures at the inlets. More precisely, the system is

$$\begin{cases} p_{a,i} - p_{a,o} = 2c_{f,a}L/(\rho_a\pi^2r^5)\cdot w_a|w_a| \\ p_{b,i} - p_{b,o} = 2c_{f,b}L/(\rho_b\pi^2r^5)\cdot w_b|w_b| \\ c_a\rho_a\pi r^2\frac{L}{N}\dot{T}_{a,j} = w_a c_a\cdot(T_{a,j-1} - T_{a,j}) \\ \qquad +\gamma_a\left(\frac{w_a}{w_{a,nom}}\right)^{0.8}\pi r\frac{L}{N}\cdot(T_{w,j} - T_{a,j}) \\ c_w\rho_w\pi r^2\frac{L}{N}\dot{T}_{w,j} = -\gamma_a\left(\frac{w_a}{w_{a,nom}}\right)^{0.8}\pi r\frac{L}{N}\cdot(T_{w,j} - T_{a,j}) \\ \qquad -\gamma_b\left(\frac{w_b}{w_{b,nom}}\right)^{0.8}\pi r\frac{L}{N}\cdot(T_{w,j} - T_{b,N-j+1}) \\ c_b\rho_b\pi r^2\frac{L}{N}\dot{T}_{b,j} = w_b c_b\cdot(T_{b,j-1} - T_{b,j}) \\ \qquad +\gamma_b\left(\frac{w_b}{w_{b,nom}}\right)^{0.8}\pi r\frac{L}{N}\cdot(T_{w,N-j+1} - T_{b,j}) \end{cases} \tag{4}$$

where T stands for temperature, w for mass flowrate, p for pressure, c_f for the friction coefficient, c and ρ for (constant) specific heat and density, and γ the coefficient of heat transfer; the a, b and w subscripts denote respectively the two streams and the wall, while $j \in [0, N]$ ($j = 0$ for boundary conditions) is the volume index, counted for both streams from inlet to outlet, the wall being enumerated like stream a; the i and o subscripts, finally, stand for "inlet" and "outlet". Table 2 shows the parameter values used in the example.

Parameters					
$p_{a,i}$	1.216kPa	r	0.1m	ρ_b	3500kg/m^3
$p_{b,i}$	1.216kPa	s	0.005m	ρ_w	4200kg/m^3
$p_{a,o}$	1.013kPa	$c_{f,a}$	0.1	γ_a	100W/(m^{2}K)
$p_{b,o}$	1.013kPa	$c_{f,b}$	0.2	γ_b	100W/(m^{2}K)
$T_{a,i}$	323.15K	c_a	4200J/(kg K)	$w_{a,nom}$	0.5kg/s
$T_{b,i}$	288.15K	c_b	3500J/(kg K)	$w_{b,nom}$	0.5kg/s
N	10	c_w	3500J/(kg K)		
L	30m	ρ_a	4200kg/m^3		

Table 2. Parameter values of Model (4).

In this example, contrary to the previous one, there is no neat physical separation between the time scales of the involved dynamics. In fact, the cycle analysis leads to the constraints

$$\begin{aligned} T_{a,j}: &\quad h \leq 10.383 \\ T_{b,j}: &\quad h \leq 13.327 \\ T_{w,j}: &\quad h \leq 13.658 \end{aligned}$$

which show that the time scales associated with the variables are quite close one to another. Due to the physical nature of this system, DD is not expected to take particular advantage of the partition.

Choosing an integration step $h = 13.0$ yields a partition that considers the $T_{a,j}$ as the fast while the $T_{b,j}$ and $T_{w,j}$ as the slow states. Figure 6 shows the simulation results — notice that the temperatures are reported with different scales.

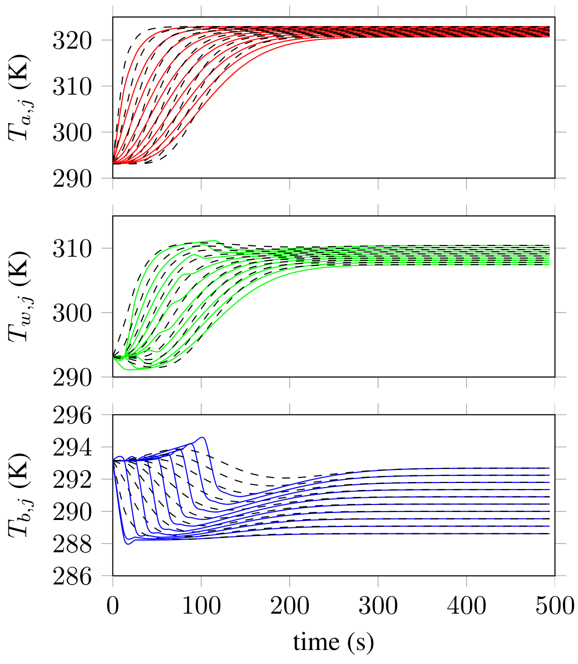

Figure 6. Simulation results of (4) with $\alpha = 1.0$. Dashed lines represent the real trajectories, while the solid lines are the trajectories obtained with DD.

Table 3 shows the simulation statistics for different integration methods. It is worth noticing that the dimension of the system is reduced from 30 to 10 in the mixed-mode method. Also, the EE method needs a smaller step size, hence $h = 10.0$ was chosen, again for numerical stability reasons.

	Mixed-mode	BDF	IE	EE
# Steps	**38**	212	**38**	50
# Function ev.	**114**	241	**114**	–
# Jacobian ev.	**2**	4	**2**	–
# Fun. ev. in Jac. ev.	**22**	120	62	–
# Newton iterations	**76**	237	**76**	–
Accuracy	0.017	–	**0.014**	0.059
Sim time	**0.04s**	0.15s	0.06s	0.08s

Table 3. Simulation statistics for Model (4) ($h = 13.0$).

As can be seen in (2), the integration step depends on the choice of α. Choosing a value of $\alpha = 1.0$ is usually a good choice—in fact the value of α used in the previous examples. This choice is however quite aggressive from the point of view of accuracy, even if the low frequency dynamics are caught (see, for instance, $T_{b,j}$ in Figure 6). Choosing a smaller value of α, e.g., $\alpha = 0.5$, will conversely yields a more conservative partitioning but more accurate a numerical solution. In particular, the output of the cycle analysis changes to

$$\begin{aligned} T_{a,j}: &\quad h \leq 5.191 \\ T_{b,j}: &\quad h \leq 6.664 \\ T_{w,j}: &\quad h \leq 6.829 \end{aligned}$$

and choosing $h = 6.0$ leads to the same partition as before, but producing more accurate solutions (see Figure 7). Apparently enough (see Table 4), the performance of the

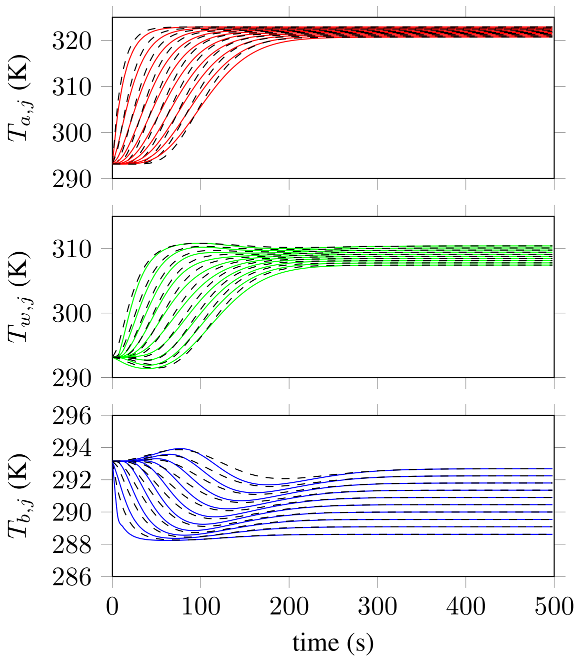

Figure 7. Simulation results of (4) with $\alpha = 0.5$. Dashed lines represent the real trajectories, while the solid lines are the trajectories obtained with DD.

mixed-mode method is still better in terms of simulation speed, and the accuracy is improved.

	Mixed-mode	BDF	IE	EE
# Steps	**83**	213	**83**	100
# Function ev.	**234**	243	243	–
# Jacobian ev.	**4**	**4**	**4**	–
# Fun. ev. in Jac. ev.	**44**	120	124	–
# Newton iterations	**151**	239	160	–
Accuracy	0.011	–	**0.008**	0.018
Sim time	**0.08s**	0.15s	0.16s	0.10s

Table 4. Simulation statistics for Model (4) ($h = 6.0$).

The presented examples have been kept as small as possible in order to improve results readability, but the method can be applied to larger models as well. For example, by changing in (4) the parameter N to 30, the model becomes of order 90, and the obtained simulation results are summarised in Table 5.

	Mixed-mode	BDF	IE	EE
# Steps	**125**	304	**125**	250
# Function ev.	**336**	337	345	–
# Jacobian ev.	**6**	**6**	**6**	–
# Fun. ev. in Jac. ev.	**186**	540	546	–
# Newton iterations	**211**	333	220	–
Accuracy	**0.014**	–	**0.014**	0.084
Sim time	0.21s	0.43s	0.41s	**0.20s**

Table 5. Simulation statistics for Model (4) ($h = 4.0$).

4.3 Remarks

The presented examples prove the usefulness of the approach, and show its potentialities, concluding the presentation of the DD technique. However, a last statement need motivating, i.e., the DD technique can complement existing EOOMT. To this end, we need discussing how to insert the presented technique in a manipulation toolchain of modern EOOMT (Section 5).

5. A Unifying Manipulation Toolchain

As already stated, nowadays MOR techniques do not allow to introduce approximations in the solution of DAE systems, neither acting on equations (e.g., MOR techniques) nor acting on the solution (e.g., DD). In this section we propose a complementing manipulation toolchain that bridges those concepts, without altering the classical manipulation framework, but adding some useful functionalities for the model designer (see Figure 8).

In particular, the idea is that if the analyst wants to perform some approximations in order to improve simulation speed, he/she needs to be able to specify high-level properties, e.g., upper bounds on the approximation error, and which technique must be used for it, e.g., the MOR technique as well as whether or not the use of DD is advisable.

Figure 8 depicts the proposed toolchain of model manipulations, from the EOO description to the simulation algorithm ready for code generation. The decision nodes (the diamond ones in the diagram) show where additional manipulation for simplification can be performed. If, in every decision node, the simplification is not performed, the classical manipulation toolchain comes out. Otherwise, a simpler model is produced at the end of the toolchain. The diagram also reports some coloured dashed boxes on the right side. Red boxes stand for already available methodologies that can be automatically applicable at this level of the manipulation, while green ones stand for potential methodologies which may be introduced as automatic procedures, but to date not exploited in the context of EOO modelling.

5.1 An Example Toolchain Implementation

To prove the feasibility of extending an EOO modelling toolchain as here suggested, the task was actually carried out by using JModelica[1] as the Modelica translator, exporting the model as a Functional Mockup Unit (FMU), and

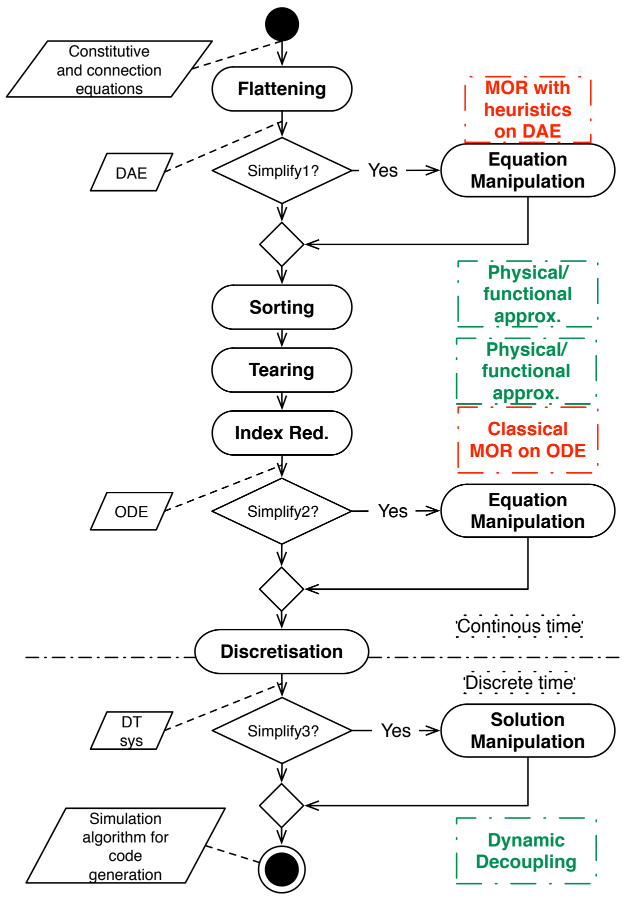

Figure 8. Activity diagram of the modified manipulation toolchain.

employing Assimulo[2] for the numerical integration, having developed the mixed-mode integrator *ad hoc*.

More in detail, the toolchain of Figure 8 was modified – for the case when "simplify" is desired – as shown in Figure 9: the output of the continuous-time part (the manipulated `model.mo`) is exported by means of the Functional Mockup Interface (FMI) to `model.fmu`, elaborated by the external python module `jd2.py` that performs the cycle analysis (i.e., takes care of the "discretisation" and the "solution manipulation" blocks); the partitioned model is then simulated with Assimulo, with the developed mixed-mode method. It is worth noticing that the integration of a new functionality (like DD) into an EOO modelling toolchain was greatly eased by adopting, for the various phases, tools that allow for some common interchange format—a feature of great importance indeed.

The developed code, including the reported examples, is available as free software, within the terms of the Modelica License v2, at the URL `http://home.dei.polimi.it/leva/jd2.html`.

6. Conclusion and Future Work

A technique was presented to allow equation-based EOOMT to take profit of DD, partitioning a complex model into "weakly coupled" submodels in a view to enhancing the

[1] `http://www.jmodelica.org`

[2] `http://www.jmodelica.org/assimulo`

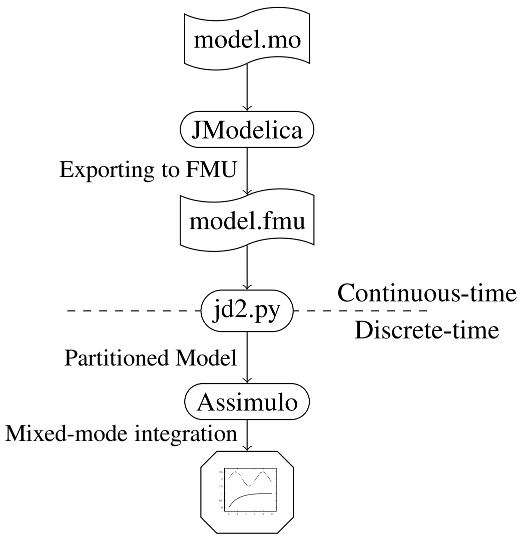

Figure 9. Integration of DD in the toolchain of Figure 8.

obtained simulation efficiency. Also, differently from some alternatives that were comparatively reviewed, the technique is naturally keen to the use of parallel simulation, or co-simulation.

The technique is based on a structural analysis of the system – called here "cycle analysis", and novel – coupled to a convenient use of mixed-mode integration. Some considerations were made on how the presented technique can be integrated in EOOMT, considering a Modelica translator as example. Simulation tests were reported to illustrate the achieved benefits, and the realised implementation was made available as free software to the community.

Future work will concern the exploitation of the mentioned keenness to co-simulation, and a tighter integration into EOOMT by developing a consistent and informative user interface. Also, the analysis will be deepened by defining convenient "separability indices" – on which some preliminary ideas are already available – to form the basis for said informative interfaces, and possibly to further automate the overall decoupling process. Finally, models of higher complexity will be addressed, so as to possibly improve the time performance of the software implementation.

Acknowledgements

The authors are grateful to Prof. F. Casella and Prof. J. Åkesson for the useful discussions and constructive criticisms, and for providing advice and cooperation to the integration of DD in the JModelica framework.

References

[1] A. Antoulas. *Approximation of large-scale dynamical systems*, volume 6 of *Advances in Design And Control*. SIAM, 2005.

[2] A. Bartolini, A. Leva, and C. Maffezzoni. A process simulation environment based on visual programming and dynamic decoupling. *Simulation*, 71(3):183–193, 1998.

[3] F. Casella, A. Leva, and C. Maffezzoni. Dynamic simulation of a condensation plate column by dynamic decoupling. In *Proc. EUROSIM '98, Espoo 1998*, pages 368–374, 1998.

[4] F. Casella and C. Maffezzoni. Exploiting weak interactions in object-oriented modeling. *Simulation News Europe*, 22:8–10, 1998.

[5] F. Cellier and E. Kofman. *Continuous system simulation*. Springer, 2006.

[6] J. Chen and S.-M. Kang. Model-order reduction of nonlinear MEMS devices through arclength-based Karhunen-Loeve decomposition. In *The 2001 IEEE Int. Symp. on Circuits and Systems*, volume 3, pages 457–460, 2001.

[7] J. Chen, S.-M. Kang, J. Zou, C. Liu, and J. Schutt-Aine. Reduced-order modeling of weakly nonlinear MEMS devices with Taylor-series expansion and Arnoldi approach. *J. of Microelectromechanical Systems*, 13(3):441– 451, 2004.

[8] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley, 2003.

[9] L. Goldberg and G. Ann. *Efficient algorithms for listing combinatorial structures*, volume 5. Cambridge Univ Pr, 2009.

[10] M. Innocent, P. Wambacq, S. Donnay, H. Tilmans, W. Sansen, and H. De Man. An analytic volterra-series-based model for a mems variable capacitor. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(2):124–131, 2003.

[11] S. Lall, J. Marsden, and S. Glavaski. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *Int. J. of Robust and Nonlinear Control*, 12:519–535, 2002.

[12] L. Mikelsons and T. Brandt. Symbolic model reduction for interval-valued scenarios. In *ASME Conf. Proc.*, volume 49002, pages 263–272. ASME, 2009.

[13] L. Mikelsons and T. Brandt. Generation of continuously adjustable vehicle models using symbolic reduction methods. *Multibody System Dynamics*, 26:153–173, 2011.

[14] A. V. Papadopoulos, J. Åkesson, F. Casella, and A. Leva. Automatic partitioning and simulation of weakly coupled systems. Technical report, Politecnico di Milano, 2013.

[15] J. R. Phillips. Projection frameworks for model reduction of weakly nonlinear systems. In *Proc. of the 37th Annual Design Automation Conf.*, DAC '00, pages 184–189, New York, NY, USA, 2000. ACM.

[16] J. Scherpen. Balancing for nonlinear systems. *Systems & Control Letters*, 21(2):143–153, 1993.

[17] A. Schiela and H. Olsson. Mixed-mode integration for real-time simulation. In *Modelica Workshop 2000 Proc.*, pages 69–75, 2000.

[18] M. Sjölund, R. Braun, P. Fritzson, and P. Krus. Towards efficient distributed simulation in modelica using transmission line modeling. In *3rd Int. workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 71–80, 2010.

[19] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. on Computing*, 1(2):146–160, 1971.

[20] R. Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM J. on Computing*, 2(3):211–216, 1972.