Proceedings
of the 4th International Modelica Conference,
Hamburg, March 7-8, 2005,
Gerhard Schmitz (editor)

F. Casella
Politecnico di Milano, Italy
Exploiting Weak Dynamic Interactions in Modelica
pp. 97-102

Exploiting Weak Dynamic Interactions in Modelica

Francesco Casella
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Piazza Leonardo da Vinci, 32 - 20133 Milano ITALY
e-mail: casella@elet.polimi.it

Abstract

The simulation of complex systems can be made more efficient by splitting the system into several, weakly interacting subsystems, and then integrating their equations separately. The paper discusses the required extension to the Modelica language, as well as the corresponding integration algorithms. Possible applications include real-time integration of large systems, distributed simulation, and the integration of Modelica with external simulators.

1 Introduction

Consider a complex dynamical system, obtained by connecting fast and slow subsystems. The simultaneous integration of the corresponding large set of DAEs can become highly inefficient, especially when a fixed-step algorithm is employed (e.g., for real-time simulation): the fast dynamic subsystems force the adoption of a very small integration time step, and the system of coupled DAEs to be solved at each step is very large.

In some cases, however, the system can be decomposed into two (or more) weakly interacting subsystems, whose describing DAEs can be solved independently at each time step. This allows to split the overall numerical integration task into many smaller tasks, which can be carried out more efficiently; moreover, it opens the way to distributed simulation, where each integration task runs on a different CPU. This method can offer very significant performance improvements in real-time, fixed time-step simulators, e.g. for hardware-in-the-loop and training applications.

The weak dynamic interaction method has been extensively investigated in our research group in the past 10 years. The first notable example is the general-purpose, object-oriented modelling and simulation environment MOSES [1], which relied on an object-oriented database for system modelling, and on DASSL-RT to perform variable step-size integration. The second example is the ProcSim package [2], a simulation environment for power plant simulation, based on the visual LabView environment, relying on ad-hoc, implicit Euler integration algorithms.

A brief paper on the basic concepts of weak interactions in object-oriented modelling appeared several years ago on the Eurosim Simulation News Europe magazine [3]; the purpose of this paper is to propose an implementation of those concepts in the Modelica framework. The mathematical foundations of weak dynamic interaction are first introduced in Section 2. The extension of the Modelica language is then discussed in Section 3, with reference to a simple example; the problem of generating the corresponding numerical simulation code, as well as the application to distributed simulation are also dealt with. Section 4 follows with further discussion, including the comparison of the proposed approach to other existing methods to speed up real-time simulations. Section 5 concludes the paper with some proposals for future work.

2 Weak Dynamic Interaction: Mathematical Foundations

The concept of weak dynamic interaction is introduced with the aid of a simple example. Consider the electrical circuit represented in Fig. 1.

The complete set of the component and connection DAEs is:

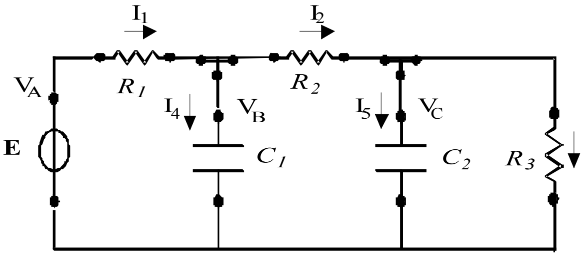

Figure 1: A simple electrical circuit

$$
\begin{aligned}
&E - V_A = 0\\
&V_A - V_B = R_1 I_1\\
&C_1 \dot{V}_B = I_4\\
&V_B - V_C = R_1 I_2\\
&C_2 \dot{V}_C = I_5\\
&V_C = R_3 I_3\\
&I_1 - I_4 - I_2 = 0\\
&I_2 - I_5 - I_3 = 0
\end{aligned}
\tag{1}
$$

In this particular case, it is straightforward to solve the algebraic equations for the algebraic variables, and to rewrite the system in ODE form:

$$\dot{x} = Ax + Bu \tag{2}$$

where $x = \begin{bmatrix} V_B & V_C \end{bmatrix}'$ and $u = E.$

Suppose that the system (2) is solved by Euler's *implicit* formula; at each time step, the following linear system has to be solved:

$$H x_{k+1} = x_k + f_{k+1} \tag{3}$$

where $H = I - A\Delta t$, $f = Bu\Delta t$, and Δt is the integration time step. The integration algorithm is A-stable for any Δt. Note that, in the case of a non-linear system, the role of matrix H would be played by the system Jacobian, which should be inverted at (almost) each time step.

If the system is solved by Euler's *explicit* formula, the solution is given by

$$x_{k+1} = F x_k + f_k \tag{4}$$

where $F = I + A\Delta t$. In this case, no matrix inversion is needed; on the other hand, the algorithm is A-stable only if $\Delta t < 2T_{min}$, where T_{min} is the minimum time constant of matrix A. Explicit integration formulae are not convenient if the system is stiff, i.e. if there are mixed fast and slow dynamics, as the fast one dictates the maximum possible time step.

Suppose now that both R_2 and C_2 are sufficiently large: the variation of the voltage V_C within a time step is likely to be small, compared to the voltage drop across the resistor, thus having a *weak* influence on the current I_2; conversely, the current I_2 cannot vary V_C substantially over a single time step, and thus has a *weak* influence on V_C. With reference to the R2-C2 connection, the voltage is thus a *weak connection variable* on the resistor side, while the current is a *weak connection variable* on the capacitor side. In other words, an approximate dynamic decoupling can be applied at the connection between R2 and C2. It is then possible split the model in two parts, as shown in Fig. 2.

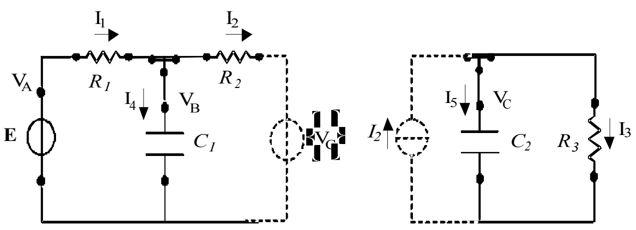

Figure 2: The electrical circuit split into two interacting subsystems.

The left part of the circuit is connected to a fictitious voltage generator V_C, and the right part of the circuit is connected to a fictitious current generator I_C.

The idea is now to adopt a mixed implicit-explicit integration algorithm to the system of Fig. 2. At each time-step, the equations of each sub-circuit are integrated using Euler's *implicit* formula, while taking into account the *last computed value* of the boundary variable I_C or V_C. The corresponding integration formula is:

$$G_L x_{k+1} = G_R x_k + f_{k+1} \tag{5}$$

where G_L and G_R are the following triangular matrices:

$$G_L = \begin{bmatrix} 1 + \dfrac{\Delta t}{R_1 C_1} & 0 \\ -\dfrac{\Delta t}{R_2 C_2} & 1 + \dfrac{\Delta t}{R_3 C_2} \end{bmatrix}$$

$$G_R = \begin{bmatrix} 1 & \dfrac{\Delta t}{R_2 C_1} \\ 0 & 1 - \dfrac{\Delta t}{R_2 C_2} \end{bmatrix}$$

As matrix G_L is now triangular (due to the dynamic decoupling), the solution of equation (5) is trivial, and has almost the same computational weight of the explicit formula (4). However, the integration formula remains A-stable for much larger time steps than the fully explicit formula; for large values of R_2, it is even unconditionally stable. For example, if the following values are taken:

R_1=0.1; C_1=1; R_3=1; C_2={1,3,10}; R_2=[0.1-10];

the stability regions shown in Fig. 3 and 4 are obtained; the stable region is below the limit curve for each value of C_2. It is apparent how the algorithm based on the weak interaction method allows far larger integration time steps than the explicit algorithm, while requiring a comparable computational time, as no matrix inversion is required.

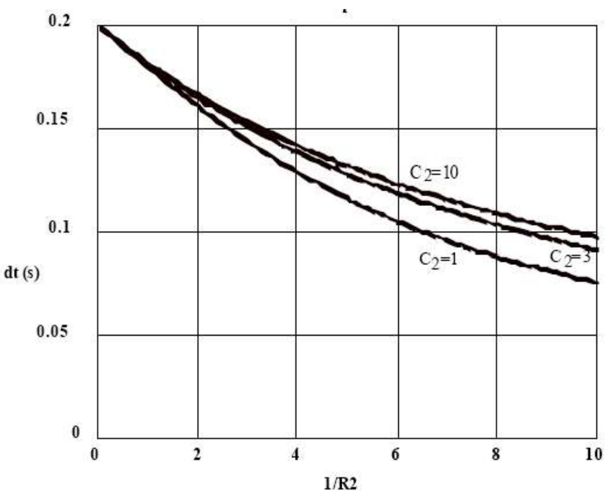

Figure 3: Stability regions of explicit Euler's integration algorithm.

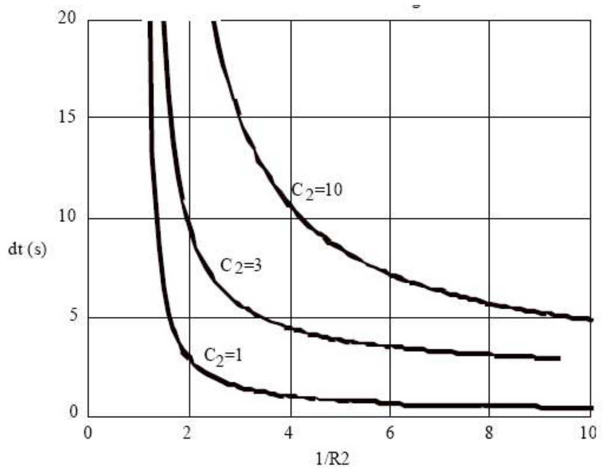

Figure 4: Stability regions of the weakly interacting integration algorithm.

Going beyond the simple example shown here, the advantages of the proposed modelling and numerical integration strategy become more and more substantial if the systems are nonlinear, and as the order of the weakly coupled subsystems grows larger.

Finally, if the dynamics of the weakly coupled subsystems is very different (i.e. one is fast and one is slow), a *multirate* integration algorithm could be adopted, in which the time step of the slow subsystem is a multiple of the time step of the fast subsystem. This kind of strategy, whose detailed analysis is outside the scope of this paper, can lead to even larger computational savings, if the fast subsystems have a few states, while the slow subsystems contains the majority of the states.

3 Weak Interaction in the Modelica Framework

3.1 The `weak` modifier

The *weak interaction* method will now be introduced in the Modelica framework with the help of a representative example, i.e., the simulation of a power generation system. The system (see Fig. 5) is composed by the connection of two main sub-systems: the mechanical power generation unit (e.g. a gas turbine unit, or a boiler-steam turbine unit), and a synchronous generator, connected either to the grid or to local loads. The aim of the simulation is to simultaneously represent the control system dynamics of both units, with time scales ranging from less than a tenth of a second (swing dynamics of the synchronous generator), to several minutes (boiler pressure dynamics).

Assuming that the rotational inertia has been lumped on the `MechGen` side, the high mechanical inertia of the gas turbine shaft provides a dynamic decoupling between the thermo-mechanical system on the left side of the connection and the electro-mechanical system on the right side of the connection. In other words, a step variation of the torque applied to the shaft by the electrical generator cannot vary the shaft speed substantially over time scales smaller than 0.2-0.5 seconds; this means that the torque is a weak connection variable on the mechanical generator side. On the other hand, since the shaft speed cannot vary substantially over such a time scale, the shaft angle is a weak connection variable on the electrical generator side.

If the two subsystems are connected by means of standard Modelica.Mechanics.Rotational connectors, this situation could be described by adding the **weak** modifier to the Modelica language, and by introducing the concept of *weak connection*:

```
connect(MechGen.Shaft(weak tau),
        ElecGen.Shaft(weak phi));
```

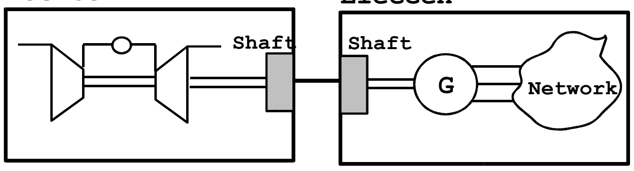

Figure 5: Model of a power generation system.

Alternatively, the **weak** attribute could be directly built in the two sub-system connectors, which should be declared as follows:

```
import Modelica.Mechanics.Rotational.*;
model MechGen
  Interfaces.Flange_a Shaft(weak tau);
...
end MechGen;

model ElecGen
  Interfaces.Flange_b Shaft(weak phi);
...
end ElecGen;
```

It should also be possible to apply the **weak** modifier to variables declared inside a model, if their influence on the object behaviour across a single time-step is small; for example, the air temperature at the gas turbine air intake.

3.2 Integration Algorithms

The weak variables can now be exploited by the integration algorithm. The basic idea is that every weak variable can be treated as an input variable by the numerical integration algorithm, regardless of its actual physical causality; the corresponding input value will correspond to last computed value available to the integration algorithm.

If the **weak** variables are replaced by **input** variables, the Modelica compiler could then easily decompose the dynamics of the whole system in two weakly interacting subsystems, as shown in Fig. 6. The numerical integration task could then be decomposed into two smaller sub-tasks. Each sub-task will read the last computed value available of its input variables to compute the next value of its output variable, as explained by the following pseudo-code (an explicit integration algorithm is assumed for simplicity):

```
T := 0.05; // Time step length
loop
  (phi,x_mech) :=
     MechGenInt(tau,x_mech,T);
  (tau,x_elec) :=
     ElecGenInt(phi,x_elec,T);
end loop;
```

where `x_mech` and `x_elec` are the state vectors of the two subsystems.

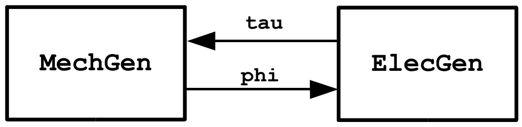

Figure 6: Weakly interacting subsystems.

Each subsystem can be integrated with the method of choice, represented here by the `MechGenInt` and `ElecGenInt` functions. If implicit algorithms are used, the dynamic decoupling leads to two smaller sets of nonlinear equations, to be solved sequentially. This has beneficial effects on the computation time, as already discussed in Section 2.

In the simple example shown above, the two integration algorithms are synchronous (i.e., they share the same time step). However, since the two corresponding subsystems are characterised by widely different dynamic time constants, it is also possible to choose different step lengths for each one, as shown in the following pseudo-code:

```
Tmech := 0.1;
Tel := Tmech/N;
loop
  (phi,x_mech) :=
    MechGenInt(tau,x_mech,Tmech);
  for h in 1:N loop
    (tau,x_elec) :=
      ElecGenInt(phi,x_elec,Tel);
  end for;
end loop;
```

This scheme has a big potential for the real-time, fixed time-step simulation of complex systems: instead of dealing with a huge coupled system, which must be integrated with a small time step dictated by its fastest subsystem, it is possible to split the integration task into many independent sub-tasks, each one having the appropriate step length. In this way, each subsystem is neither under-sampled or over-sampled, and the computational resources are used efficiently.

3.3 Distributed Simulation

If a system can be decomposed into several, weakly interacting subsystems, it is also straightforward to devise a distributed simulation strategy: the integration tasks of each sub-system can be allocated on different CPUs, communicating e.g. through a shared memory database or TCP/IP sockets (Fig. 7).

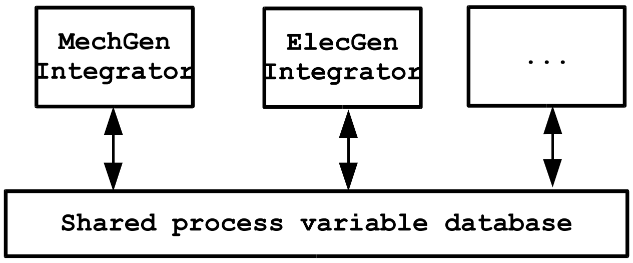

Figure 7. Distributed simulation architecture.

A further possibility is to support co-simulation, by introducing “external simulator” objects, which are an extension of the already existing Modelica external functions. In this case, a Modelica “wrapper” model, defining the input, output, and weakly interacting connector variables, could be written and then linked to external simulation applications. The actual communication between the Modelica-based simulator and the other ones could then be performed via TCP/IP sockets, DLLs, DDE, or any other interprocess communication mechanism. In this way, it would become possible to merge parts of a simulator obtained from a Modelica model with other parts provided by different simulators (e.g. a CFD code, or a digital controller emulator), or by interactive user interfaces.

3.4 Further extensions to the proposed method.

When introducing the example in Section 3.1, the assumption was made that the shaft inertia was entirely contained in the `MechGen` model; in other words, that the electrical generator model inside `ElecGen` had no inertia at all. If this is not true, the weakening method cannot be applied directly as explained: the `ElecGen.Shaft.phi` variable, far from being weakly interacting with the `MechGenShaft.phi` variable, has some inertia rigidly connected to it. The corresponding decoupled integration algorithm would probably be unstable for rather small time steps.

This case could be easily recognised by the symbolic manipulation engine: the weakly interacting model (Fig. 6) would have more state variables corresponding to the weak connection variable `phi`, if compared with the original, rigidly connected model. It could then be possible to devise a symbolic manipulation procedure to remove the inertia contribution from the weak side (i.e., `ElecGen`) and add to the other side (i.e., `MechGen`). This could widen the range of applicability of the proposed method.

4 Discussion and Outlook

The proposed method requires a very limited modification to the Modelica language, i.e., introducing the **`weak`** modifier to the variable declarations. It also requires a very limited intervention to the user's models, i.e. redeclaring some connector variables as weak, in order to obtain faster simulation code. The **`weak`** modifier can be thought of as a modelling attribute, stating that a particular variable has a weak dynamic influence on the model behaviour, as well as a hint to the compiler on how to produce more efficient simulation code.

The main drawback is that an “expert” user is needed, who knows by experience which connection (or model) variables are good candidates to be considered as weak, in order to speed up the simulation. In fact, if the “wrong” weak variables are selected, the integration algorithm could become unstable even for small values of the time step.

The stability analysis of the simulation of an electrical or hydraulic network split into two weakly interacting sub-networks is discussed in detail in [4]. The stability criterion is formulated in terms of the impedances of the two sub-networks; a heuristic rule can then be derived, stating that the sub-network with a weak flow connection variable should have low resistance and/or high capacitance, while the one with a weak effort connection variable should have high resistance and/or low capacitance. A similar, physical-based analysis could be carried out for mechanical connections, such as the case discussed in Section 3.

Another possibility could be to devise numerical indicators (for generic models), based on the analysis of the linearised equations, to help the user determine which connection variables can be considered weak. An exhaustive search of all the potential candidates should not be computationally too expensive, as the number of connections is limited; moreover, only connections between higher-level subsystems could be considered, e.g. the shaft connection in the example of Fig. 5, while ignoring the connections inside the `MechGen` and `ElecGen` models.

The proposed numerical method has already been tested with both variable step-size [1] and fixed step-size [2] integration algorithm. Although a reduction around 30% has been reported in a variable step-size robotic simulation, the application of the weak interaction method to variable step-size, higher-order integration algorithm can only give a limited benefit, as it is hampers the lengthening of the time step, as well as the switching to higher order formulae; moreover, devising multirate, variable step-size algorithms is very difficult. The most significant performance enhancements can be obtained with fixed time-step algorithm, in particular for real-time applications. A typical case is a large physical system with mixed slow and fast dynamics, possibly controlled by digital control systems with widely different sampling times. Such systems are often encountered in industrial practice.

The proposed method can readily benefit from the inline integration method [5], which can be directly applied to the integration of the weakly interacting sub-systems.

There are some similarities between the weak dynamic interaction method and the mixed-mode integration method proposed in [6], as both try to exploit mixed implicit-explicit algorithms to break large non-linear systems of equations into smaller ones. The method proposed in [6] tries to suitable partition the overall model into a “fast” and a “slow” part, by linear eigenvalue analysis and some heuristics to limit the search space, while the method presented here focuses on a system-level approach, i.e. tries to exploit the weak coupling between sub-systems at their connections. It is nevertheless possible that the two methods could be suitably integrated, as their approaches are somehow complementary to each other.

Another method which has been proposed to decouple the equations of large systems is the Transmission Line Method [7, 8]. In this case, the finite propagation speed of physical quantities along connecting elements is exploited to allow the implicit integration of each connected subsystem, using only past values from the other ones. The main advantage in this case is that decoupling comes naturally from the system equations, which are exact; in some cases, such as hydraulic networks or high-frequency electrical circuits, this can be very convenient. On the other hand, there are cases where the “physical” propagation speed is too high, so that the TLM method could unnecessarily introduce fast dynamics when there is no need to; moreover, the exact value of the integration time step becomes tightly coupled with the connecting element parameters. It could be interesting to investigate if the stability analysis studies carried out for the TLM method could be somehow extended to the weak interaction method.

5 Conclusions

The introduction of the **`weak`** variable concept allows to widen the applicability of the Modelica framework in these directions:

- efficient fixed-step simulation of weakly interacting complex systems, possibly having different time scales;
- distributed simulation;
- co-simulation of Modelica-based simulators and other simulation engines;
- user interaction by means of standard SCADA tools;

while retaining all the advantages of a fully object-oriented description.

Possible applications range from real-time hardware-in-the-loop simulation, to interactive training simulators, to the interfacing of system-level models with detailed (e.g. CFD) models of specific system components.

The implementation of this concept would require the following steps:

1. Extend the definition of the Modelica language to include the **`weak`** modifier keyword.
2. Implement the symbolic manipulation algorithm to split weakly coupled subsystems, as well as decoupled numerical integration algorithms, into an existing Modelica compiler (e.g., Dymola or OpenModelica)
3. Validate the method on selected case studies (e.g. robotics, power generation systems).

Step 2. could be initially limited to very simple integration schemes, such as synchronous explicit Euler, and then possibly extended to more sophisticated solutions, such as implicit, high-order, multi-step, and multirate algorithms.

References

[1] C. Maffezzoni, R. Girelli “MOSES: Modular Modelling in an Object Oriented Database”, *Mathematical Modelling of Systems*, v. 4, pp 121-147, 1998.

[2] A. Leva, A. Bartolini, C. Maffezzoni: “A process simulation environment based on visual programming and dynamic decoupling”, *Simulation*, v.71, n. 3, pp.183-193, 1998.

[3] F. Casella, C. Maffezzoni: "Exploiting Weak Interactions in Object Oriented Modeling", *EUROSIM Simulation News Europe*, Mar. 1998, pp. 8-10.

[4] F. Casella: “Modelling, Simulation, and Control of a Geothermal Power Plant”, *Ph.D. Dissertation,* Politecnico di Milano, Italy, 1999, pp. 27-46. http://www.elet.polimi.it/upload/casella/tesi.pdf

[5] H. Elmqvist, S.E. Mattsson, H. Olsson: “New Methods for Hardware-in-the-loop Simulation of Stiff Models”, *Proceedings of the Modelica Conference 2002,* Oberpfaffenhofen, Germany, March 18-19 2003, pp. 59-64.

[6] A. Schiela, H. Olsson: “Mixed-Mode Integration for Real-Time Simulation”, *Proceedings of the Modelica 2000 Workshop,* October 23-24 2000, pp 69-75.

[7] D. M. Auslander: “Distributed System Simulation with Bilateral Delay-Line Models”, *Journal of Basic Engineering, Trans. ASME* pp 195-200, June 1968.

[8] B. Johansson, P. Krus: “Modelica in a Distributed Environment Using Transmission Line Modelling”, *Proc. Modelica Workshop 2000*, October 23-24 2000, pp. 193-198.