Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters

Farid Dshabarow
ABB Turbo Systems AG
Switzerland
Farid.Dshabarow@CH.ABB.Com

François E. Cellier, Dirk Zimmer
ETH Zürich
Switzerland
{FCellier,DZimmer}@Inf.ETHZ.CH

Abstract

This paper introduces a new Modelica library for modeling and simulation of systems with distributed parameters in one space dimension. The resulting partial differential equations of either the parabolic or hyperbolic types are being converted to sets of ordinary differential equations using either the method of lines or the finite volume approach. Some simple examples serve to document the utilization of the new library.

Keywords: *Distributed Parameter Systems, Numerical PDEs, Method of Lines, Finite Volume Method*

1 Introduction

1.1 History of General-purpose PDE Solvers

Lumped parameter systems have been successfully modeled and simulated using general-purpose simulation software for several decades. With the advent of Modelica, it has become unnecessary to model and simulate any physical systems with lumped parameters using either general programming languages, like C++, or special-purpose simulation languages, like Adams or Spice. Modelica is capable of converting any lumped parameter model of a physical system to executable code that is as efficient in its execution as the best manually coded spaghetti programs of the past. Modelica can also successfully compete with special-purpose simulation codes, like Spice or Adams, in the simulation of electronic circuits [5] and multi-body systems [15].

The modeling and simulation of distributed parameter systems using general-purpose simulation software has not been as successful. In the 70s and early 80s, a number of general-purpose simulation codes, like FORSIM VI [4], were developed for the purpose of modeling and simulating at least some classes of systems with distributed parameters. FORSIM VI, for example, was designed for simulating parabolic PDEs in one or two space dimensions. Hyperbolic PDEs could be simulated as well, but the resulting simulation code was not as efficient. Elliptic PDEs could sometimes be converted to equivalent parabolic problems using invariant embedding.

Around the same time frame, another program, ELLPACK [11], was developed that was designed for solving elliptic PDEs in two or three space dimensions. The ELLPACK project was very ambitious, and the code grew rapidly to a size that made the code difficult to use and maintain. In order to make ELLPACK easier to use, the designers of the code developed a preprocessor for translating an abstract model description down to a set of Fortran subroutine calls. Yet, as new algorithms were added constantly to the software, maintenance of the preprocessor became soon too difficult. Hence a compiler-compiler was developed that could be used to generate a new version of the preprocessor from an abstract description thereof. Yet in spite of all of those efforts, the resulting simulation programs were highly inefficient at run time.

Whereas one of the primary mantras of modeling and simulation environments is to be able to protect the user from having to fully understand the numerical properties of the underlying solver algorithms, this demand could never be fully satisfied when dealing with PDEs. The run-time efficiency of the resulting simulation code depends too heavily on the chosen discretization method, and no logic was found that could relieve the user from having to make hard choices manually.

Sometimes codes like ELLPACK have been used to quickly try out different combinations of algorithms and compare them with each other. In this way, the user could more quickly determine, which combination of algorithms might work best. However, once this decision has been reached, the final code nevertheless had to be hand-coded, because the

real problems were not limited to the PDE solvers themselves, but more often than not were related to how the code dealt with complex geometries, i.e., how physical boundary conditions were converted to boundary conditions that the PDE solver could make use of [14].

For all of these reasons, the use of general-purpose simulation software for the simulation of distributed parameter systems became unfashionable again. The researchers dealing with these types of systems simply gave up, and most of today's simulation codes are specially designed codes for very small classes of problems only.

1.2 A Renaissance for General PDE Solvers

One technique that has proven to be more robust than other approaches is the finite element method [9]. The success of this technique is based on its ability for dealing effectively with complex geometries. Originally developed for simulating elliptic 2D and 3D problems, finite element methods have quickly also been adapted to the discretization of parabolic and hyperbolic PDEs [12].

FEMLAB is a general-purpose numerical PDE solver based on the finite element method. FEMLAB was developed in recent years for the simulation of multi-physics applications. The code is capable of simulating models involving multiple PDEs [13].

FEMLAB started out as a MATLAB toolbox. Yet, its developers learnt quickly the same truth that the ELLPACK developers had learnt before them: a general-purpose PDE solver becomes soon unmanageable without a preprocessor capable of interpreting an abstract model definition. They also learnt that they needed to offer CAD support for entering the device geometry.

FEMLAB was more successful than ELLPACK, in part, because the computers have meanwhile become faster, and in part, because they were less ambitious in the sense that they didn't insist on incorporating each and every algorithm that has ever been developed for the numerical solution of PDEs.

FEMLAB has recently changed its name to COMSOL. This software represents currently the gold standard of general-purpose numerical PDE solvers for multi-physics applications.

1.3 A Role for Modelica?

Modelica has become the de facto standard for modeling and simulation of physical systems with lumped parameters. Does it have a role to play in numerical PDEs also?

Modelica, or rather its implementations, such as Dymola, offer not much that is unique or special w.r.t. their simulation engine. The only feature worth mentioning in this respect is a fairly robust root solver (discontinuity handler). The true power of Modelica lies in its ability to deal with differential and algebraic equations (DAEs) in a very flexible and truly object-oriented manner.

Today's numerical PDE solvers, including COMSOL, offer numerically advanced algorithms, but are very primitive w.r.t. their user interface. The complexity and elaboration of the user interface is at approximately the same level that the Continuous System Simulation Languages (CSSLs) were prior to the advent of the CSSL standard [2].

Would a language like Modelica have made a big impact in the 1960s, had it been available? The answer to this question is no. The computers of those times were far too small and too slow to adequately host a language like Modelica. The researchers of those days dealt with much simpler models, models that could be handled by the tools available to them, not because they lacked a better understanding of physics, but simply, because their computers couldn't handle more complex models.

Are distributed parameter problems structurally simpler than lumped parameter problems? The answer to that question is also no. Physics in general deals with 3D fields, and lumped parameter models are simply abstractions of distributed parameter problems.

If we wish to bend a pipe, we first heat up the area where the pipe is to be bent. If we were to simulate the physics of bending a pipe, we would have to solve a 3D distributed parameter problem with one PDE describing the heat diffusion problem and another PDE describing the mechanical stresses and strains within the material. These PDEs would furthermore have to be solved in a geometry that changes over time as a function of the numerical solution of the two PDEs.

Researchers aren't currently simulating such processes, they aren't dealing with partial differential and algebraic equations (PDAEs) yet, because the computers of today are too small and too slow to adequately handle such problems.

Yet, it is not too early to ponder about the language constructs and numerical algorithms that will be needed in support of such endeavors, once the computers shall have advanced to a level, where they can deal with such models effectively and efficiently.

2 PDELib for 1D Numerical PDEs

Since the Standard Modelica Library doesn't offer any support yet for modeling distributed parameter systems, we decided to take a first, and very modest, step towards the much larger and more grandiose aim outlined in the introduction.

To this end, we revisited some of the programs of the past, in particular FORSIM VI, and decided to re-implement some of the algorithms and capabilities offered by FORSIM VI in a Modelica experimental library. The results of that effort are being presented in this paper.

In order to keep things simple, we decided to limit the tool to the numerical solution of parabolic and/or hyperbolic PDEs in a single space dimension, the class of 1D numerical PDEs.

Since 1D PDEs are solved on a straight line between point A and point B, the geometry plays no role yet in these problems. The spatial discretization is straightforward; finite elements aren't needed or even useful yet for the spatial discretization; and the resulting simulation code can still be simulated fairly efficiently and rapidly using almost any half-way suitable numerical algorithm.

The aim of the project was to create an experimental tool that can be used to study some properties of numerical PDEs that haven't received much coverage yet in the open literature.

One of the numerical problems to be studied is the propagation of discontinuities through a 1D hyperbolic PDE. Such discontinuities cause a new class of numerical problems. Once the discontinuity has reached the boundary condition of the PDE, it can no longer be isolated in time. At any moment in time, the discontinuity exists somewhere within the spatial domain covered by the PDE. Thus, traditional event handling cannot be used to deal with this type of discontinuities.

A structural problem to be studied concerns the numerical solution of 1D PDAEs. Can Modelica help in translating a 1D PDAE into a simulation code that can be simulated effectively and efficiently?

Two algorithms were implemented in the first official release of PDELib: the method of lines [6], and a dialect of the finite volumes approach [10].

3 Method of Lines

Given the 1D diffusion equation:

$$\frac{\partial u}{\partial t} = \sigma \cdot \frac{\partial^2 u}{\partial x^2} \tag{1}$$

The method of lines discretizes the spatial derivatives, while keeping the temporal derivatives continuous. In a first approximation, we may write:

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x=x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2} \tag{2}$$

Plugging Eq.(2) into Eq.(1), we find:

$$\frac{\partial u_i}{\partial t} \approx \sigma \cdot \frac{u_{i+1} - 2u_i + u_{i-1}}{\delta x^2} \tag{3}$$

In this way, we have converted a PDE into a stiff set of ODEs that can now be simulated using any off-the-shelf stiff ODE solver, such as DASSL.

The method of lines is fairly easy to implement. The chosen approximation is third order accurate. If the user wishes to use a more accurate approximation formula, this can be done easily.

Care must be taken in a correct implementation of the boundary conditions. As the discretization approaches the boundary, biased discretization formulae in place of central formulae must be used in order not to make use of grid points outside the simulated domain.

The approach works fairly well, especially in the case of parabolic PDEs such as the diffusion equation. The spatial discretization of a parabolic PDE by means of the method of lines leads invariably to a stiff set of ODEs, but modern numerical ODE solvers are good at dealing with those.

This is the approach that FORSIM VI took. In order to relieve the user of having to remember different discretization formulae, FORSIM VI offered a set of Fortran subroutines for computing spatial derivatives both in the bulk and in the vicinity of the domain boundaries.

PDELib also hides the details of the discretization formulae from the user, but does so using a Graphical User Interface (GUI) as shown in Fig.1.

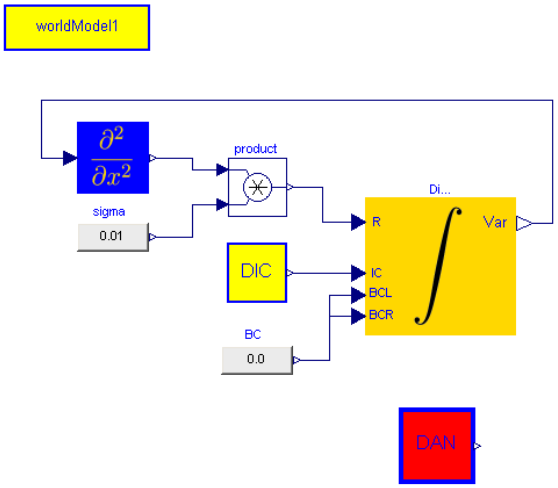

Figure 1: Model of 1D diffusion equation in PDELib

PDELib offers a method-of-lines (MOL) integrator block. This is a vector integrator block that integrates the vector of temporal derivatives, du_i/dt (marked as "R" on the integrator block), into the vector of states, u (marked as "Var"), while considering the vector of initial conditions (IC) as well as the left and right boundary conditions (BCL and BCR).

The blue box computes the spatial derivatives. In its parameter window, the user can select the approximation order to be used. Biased formulae of suitable approximation accuracy automatically replace the central formulae in the vicinity of the two domain boundaries.

The WorldModel box is used to provide general information, such as the grid width of the spatial discretization.

Since the diffusion equation with the chosen initial and boundary conditions has an analytical solution, that solution is also computed in the block DAN for comparison.

Simulation results are shown in Fig.2.

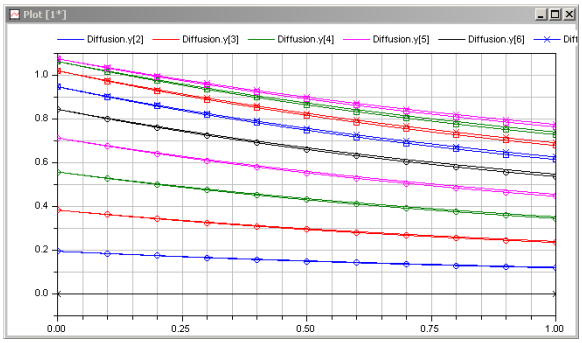

Figure 2: Diffusion equation simulation results

Since Dymola hasn't been designed for simulating PDEs, there is currently no support for 3D graphics in Dymola. The graph shows the temperature, u, at different space locations as a function of time.

The analytical results were superposed with the simulation results. In the simulation, the space was discretized into 40 segments of equal size. With 40 segments, the simulation results are still noticeably different from the analytical results.

The MOL approach is less well suited for dealing with hyperbolic PDEs, because their discretization leads to marginally stable ODE systems, rather than stiff ODE systems. Unfortunately, the numerical ODE solvers provided with Dymola and most other ODE simulators are not geared to accurately integrate marginally stable systems of ODEs.

The numerical condition of the model can sometimes be improved by using upwind discretization schemes [3]. In these schemes, the spatial derivatives are on purpose computed using biased formulae also in the bulk. FORSIM VI and PDELib offer optional upwind discretization schemes.

4 Finite Volume Method

Another discretization technique that has been successfully applied to numerically simulating hyperbolic PDEs is the Finite Volume Method (FVM). Just like the MOL technique, also the FVM approach comes in many different variants. Hence it may be useful to provide a toolkit that enables a user to compose a FVM from a set of component models.

In one space dimension, the FVM consists in subdividing the spatial domain into intervals, also called cells or finite volumes. The integral of the unknown function, u, is approximated over each of these cells at each time step. Let us denote the i^{th} cell by:

$$C_i = (x_{i-1/2}, x_{i+1/2}) \tag{4}$$

The average value of the function u over this cell is then:

$$U_i = \frac{1}{\Delta x} \int_{C_i} u(x,t)dx \tag{5}$$

How can we estimate the value of U_i? Considering the mass conservation law, we note that the average within the cell can only change due to fluxes at the boundaries, assuming that neither source nor sink

is present in the cell. Mass conservation can be expressed mathematically in the following form:

$$\frac{d}{dt}\int_{C_i} u(x,t)dx = f(u(x_{i-1/2},t)) - f(u(x_{i+1/2},t)) \quad (6)$$

where f denotes the flux function. The change of total mass inside the cell equals the flux entering the cell minus the flux leaving it.

Let us integrate Eq.(6) over time from t to $t+\Delta t$ and divide the equation by Δt and Δx. We obtain:

$$\frac{1}{\Delta t\cdot\Delta x}\int_{C_i} u(x,t+\Delta t)dx - \frac{1}{\Delta t\cdot\Delta x}\int_{C_i} u(x,t)dx = \quad (7)$$

$$\frac{1}{\Delta t\cdot\Delta x}\left(\int_t^{t+\Delta t} f(u(x_{i-1/2},t))dt - \int_t^{t+\Delta t} f(u(x_{i+1/2},t))dt\right)$$

Plugging Eq.(5) into Eq.(7), we obtain:

$$\frac{U_i(t+\Delta t)-U_i(t)}{\Delta t} = -\frac{1}{\Delta x}(F^t_{i+1/2} - F^t_{i-1/2}) \quad (8)$$

where:

$$F^t_{i-1/2} = \frac{1}{\Delta t}\int_t^{t+\Delta t} f(u(x_{i-1/2},t))dt \quad (9)$$

is the average flux over one time step.

We can reinterpret Eq.(8) as a discrete approximation of a differential equation:

$$\frac{d}{dt}U_i = -\frac{1}{\Delta x}(F^t_{i+1/2} - F^t_{i-1/2}) \quad (10)$$

Using this simple trick, we have reduced also the FVM to a Continuous-Time/Discrete-Space (CTDS) method.

How do we approximate the average flux? Different approximations have been proposed. A simple approximation is the upwind flux:

$$F_{i-1/2} = \begin{cases} cU_{i-1}, & \text{if } c < 0; \\ cU_i, & \text{if } c > 0. \end{cases} \quad (11)$$

i.e., the average flux across a border between cells during one integration step is proportional to the average value of u in the upwind cell.

5 Examples

In the following section of the paper, some of the models currently available as examples in PDELib are shown.

5.1 Linear Advection Equation

The advection equation is one of the simplest PDEs to be found. Given a constant speed, c, the linear advection equation can be written as:

$$u_t = -c \cdot u_x \quad (12)$$

This problem was encoded using the MOL approach with the initial condition:

$$u(x,0) = cos(x) \quad (13)$$

and with the boundary condition:

$$u(x_1,t) = -cos(-ct) \quad (14)$$

applied at the left boundary of the domain. The MOL model is shown in Fig.3.

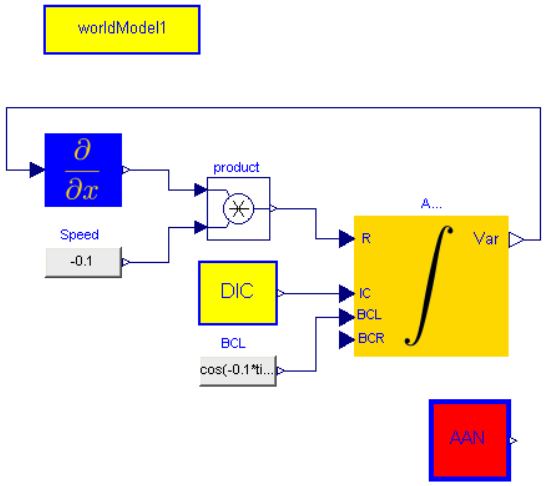

Figure 3: MOL model of linear advection equation

Some simulation results are shown in Fig.4.

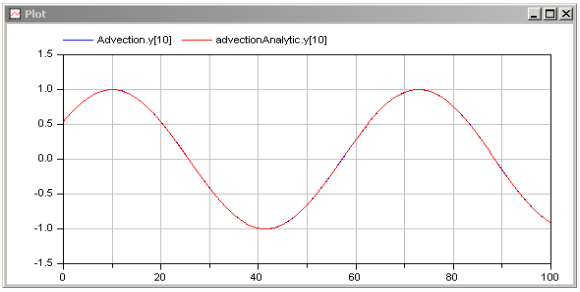

Figure 4: MOL simulation of linear advection equation

The same problem was also solved using the FVM technique with upwind flux computation. The model is shown in Fig.5.

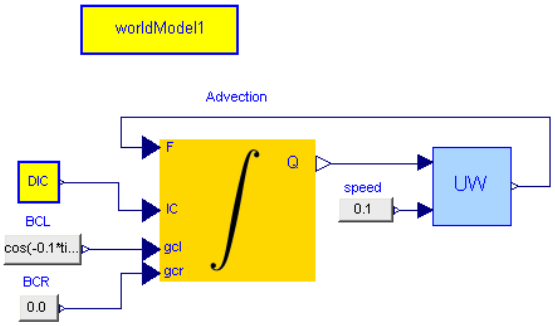

Figure 5: FVM model of linear advection equation

This model generates the simulation results shown in Fig.6:

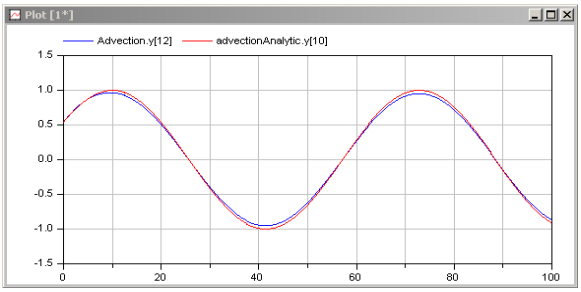

Figure 6: FVM simulation of linear advection equation

The index of the FVM solution is off by two segments due to the ghost cells used in this approach for computing the solution in the vicinity of the domain boundary [8].

5.2 Burger's Equation

The inviscid Burger´s equation is the non-linear PDE

$$u_t + (\frac{u^2}{2})_x = 0 \qquad (15)$$

If we choose as initial condition:

$$u(x,0) = x \qquad (16)$$

and as boundary conditions:

$$u(x_1,t) = 0 \qquad (17)$$

$$u(x_n,t) = 0 \qquad (18)$$

the problem has the analytical solution:

$$u(x,t) = \frac{x}{1+t} \qquad (19)$$

The MOL implementation of Burger´s equation is shown in Fig.7.

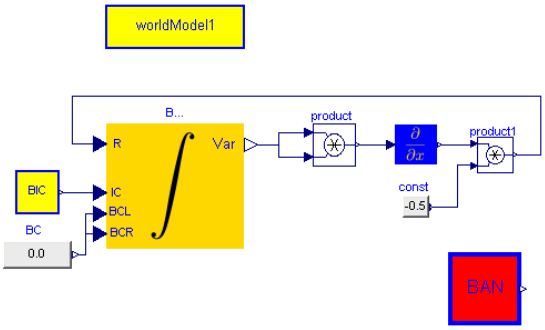

Figure 7: MOL model of Burger's equation

Some simulation results are shown in Fig.8.

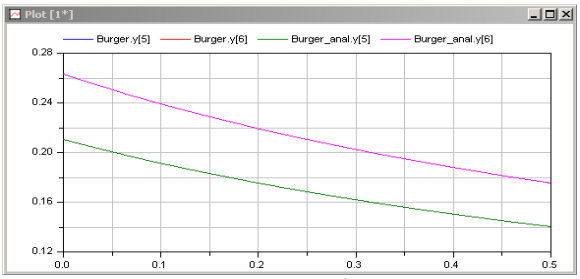

Figure 8: MOL simulation of Burger's equation

The results look excellent, but they are deceiving. The simulation here used 20 segments. Using 10 segments, the numerical results start deviating from the analytical results after only 0.2 seconds of simulated time. With 20 segments, the simulation results are more accurate, but the numerical simulation turns unstable after roughly 0.6 seconds. The more segments are being used, the faster the simulation becomes numerically unstable.

An FVM implementation of Burger´s equation is shown in Fig.9.

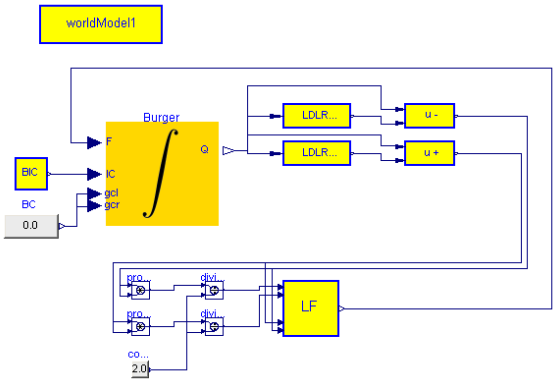

Figure 9: FVM model of Burger's equation

In this example, the FVM implementation uses a Lax-Friedrichs flux together with Local Double Logarithmic Reconstruction (LDLR) [1,7,10].

Some simulation results are shown in Fig.10.

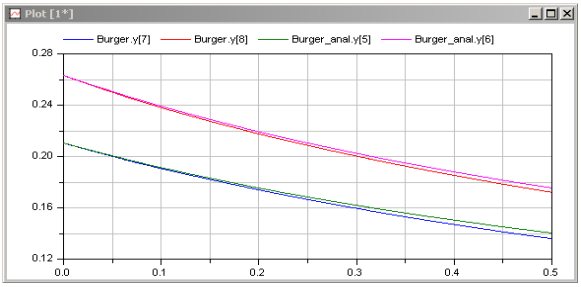

Figure 10: FVM simulation of Burger's equation

The FVM simulation remains numerically stable independent of the number of cells in use. Unfortunately, the results obtained are less accurate than using the MOL approach. The indices are again off by two because of the ghost cells.

6 Conclusions

What have we accomplished? We have been able to create an experimental library that enables experienced analysts to quickly try out different combinations of algorithms that can be used for the simulation of 1D parabolic and hyperbolic PDEs. Yet, we have failed in our aim to protect the user from having to understand the numerical properties of PDE solvers.

We chose a mathematical rather than a physical interface to our library, because it makes the tool more flexible and more general in its applicability. However, it was precisely that decision that made us fail in our endeavor of delivering a tool to the end user that can be applied blindly and reliably. This simply cannot be done at a mathematical level.

Yet, this is not a major problem. Modelica, due to its object-oriented philosophy, is good at information hiding. In the future, we shall be able to place a physical layer on top of the mathematical layer that offers solutions to particular subsets of PDEs, just as COMSOL does. Each physical module then decomposes its models internally into a combination of modules programmed at the mathematical layer.

We chose a blocks philosophy for our library. Each mathematical model is composed as a block diagram. In the long run, this decision will prove to have been a mistake. We shall need to learn to trust Modelica to make the right causality decisions for us. Otherwise, we shall never be able to solve PDAEs.

We had made the same mistake initially in the design of MultiBondLib [15], our multi-bond graph library. Initially, we formulated holonomic constraints between bodies using blocks from the Blocks library. If we use an adder:

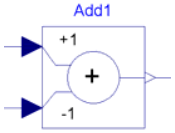

Figure 11: Adder of the Blocks library

from the Blocks library, we force Modelica to compute $y = u_1 - u_2$, but maybe the correct causality ought to be $u_2 = u_1 - y$. By using blocks from the Blocks library, we are tying Modelica's hands unnecessarily, which may lead to situations, where Modelica can no longer find a solution to the problem.

Yet for the time being, the decision to program PDELib using blocks rather than models helped us restrict the sources of errors. During the initial phase of the research, the phase of determining the most suitable numerical algorithms, the use of blocks may be a good thing.

Finally, Dymola doesn't offer any support yet for 3D graphics. Although it is possible to export simulation results to MATLAB and produce 3D graphics using that software, this is a hassle. Dynasim should develop a 3D graphics package that can be used to plot vectors of variables against time. The package should furthermore be tied to the 3D View Control window to give the users an opportunity to look at their 3D graphs from different angles.

References

[1] Artebrant, H., Schroll, J., Limiter-free Third Order Logarithmic Reconstruction. *SIAM Journal on Scientific Computation* 28 (2006) 359-381

[2] Augustin, D.C, Fineberg, M.S., Johnson, B.B., Linebarger, R.N., Sansom, F.J., Strauss, J.C.: The SCi Continuous System Simulation Lanugage (CSSL). *Simulation* 9 (1967) 281-303

[3] Carver, M.B., Hinds, H.W.: The Method of Lines and the Advective Equation. *Simulation* 31 (1978) 59-69

[4] Carver, M.B., Stewart, D.G., Blair, J.M., Selander, W.N.: *The FORSIM VI Simulation Pack-*

age for the Automated Solution of Arbitrarily Defined Partial and/or Ordinary Differential Equation Systems. Atomic Energy of Canada, Ltd., Chalk River, Ontario, 1978

[5] Cellier, F.E., Clauß, C., Urquía, A.: Electronic Circuit Modeling and Simulation in Modelica. In: *Proceedings of the Sixth Eurosim Congress on Modelling and Simulation*, Ljubljana, Slovenia (2007) Vol. 2, 1-10

[6] Cellier, F.E., Kofman E.: *Continuous System Simulation*. Springer-Verlag, New York, 2006

[7] Díaz López, J.: Shock Wave Modeling for Modelica.Fluid Library Using Oscillation-free Logarithmic Reconstruction. In: *Proceedings of the 5th International Modelica Conference*, Vienna, Austria (2006) Vol.2 641-649

[8] Dshabarow, F.: *Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters*. MS Thesis, ETH Zurich, Switzerland, 2007

[9] Hughes, T.J.R.: *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, 2000

[10] LeVeque, R.J.: *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2002

[11] Rice, J.R., Boisvert, R.F.: *Solving Elliptic Problems Using ELLPACK*. Springer-Verlag, New York, 1984

[12] Thomée, V.: *Galerkin Finite Element Methods for Parabolic Problems, 2nd Edition*. Springer-Verlag, Berlin, 1997

[13] van Schijndel, A.W.M.: Modeling and Solving Building Physics Problems with FemLab. *Building and Environment* 38 (2003) 319-327

[14] Wu, Q.M., Cellier, F.E.: Simulation of Bipolar High-voltage Devices in the Neighborhood of Breakdown. *Mathematics and Computers in Simulation* 28 (1986) 271-284

[15] Zimmer, D., Cellier, F.E.: The Modelica Multibond Graph Library. In: *Proceedings of the 5th International Modelica Conference*, Vienna, Austria (2006) Vol. 2, 559-568

Farid Dshabarow received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2007. He is now working at ABB Turbo Systems, where he deals with gas dynamics simulations in turbochargers and software for calculating and visualizing turbocharger characteristics.

François E. Cellier received his BS degree in electrical engineering in 1972, his MS degree in automatic control in 1973, and his PhD degree in technical sciences in 1979, all from the Swiss Federal Institute of Technology (ETH) Zurich. Dr. Cellier worked at the University of Arizona as professor of Electrical and Computer Engineering from 1984 until 2005. He recently returned to his home country of Switzerland. Dr. Cellier's main scientific interests concern modeling and simulation methodologies, and the design of advanced software systems for simulation, computer aided modeling, and computer-aided design. Dr. Cellier has authored or co-authored more than 200 technical publications, and he has edited several books. He published a textbook on Continuous System Modeling in 1991 and a second textbook on Continuous System Simulation in 2006, both with Springer-Verlag, New York.

Dirk Zimmer received his MS degree in computer science from the Swiss Federal Institute of Technology (ETH) Zurich in 2006. He gained additional experience in Modelica and in the field of modeling mechanical systems during an internship at the German Aerospace Center (DLR) in 2005. Dirk Zimmer is currently pursuing a PhD degree with a dissertation related to computer simulation and modeling under the guidance of Profs. François E. Cellier and Walter Gander. His research interests focus on the simulation and modeling of physical systems with dynamically changing structure.