PDE Modeling with Modelica via FMI Import of HiFlow3 C++ Components

Kristian Stavåker[1*], Stafan Ronnås[2], Martin Wlotzka[2], Vincent Heuveline[2], Peter Fritzson[1]

[1]Programming Environments Lab, Linköping University, Sweden;
*kristian.stavaker@liu.se

[2]Interdisciplinary Center for Scientific Computing (IWR) Heidelberg University, Germany

Simulation Notes Europe SNE 24(1), 2014, 11 - 20
DOI: 10.11128/sne.24.tn.10223
Received: Jan.10, 2014 (Selected SIMS 2013 Postconf. Publ.)
Accepted: February 15, 2014

Abstract. Despite an urgent need and desire in academia as well as in industry for modeling Partial Differential Equations (PDEs) using the increasingly popular Modelica modeling and simulation language, there is limited support for this available at the moment. In this work, we propose a solution based on importing PDE models with PDE solvers implemented using the general-purpose parallel finite element library HiFlow3 as models into the Modelica environment using the standard Functional Mock-up Interface. In contrast to methods based on language extensions or automatic semidiscretizations in space, this approach requires no change to the language, and enables the use of specialized PDE solvers. Furthermore, it allows for full flexibility in the choice of geometry, model parameters, and space discretization between simulation runs without recompilation needed. This makes it possible to exploit advanced features of the PDE solver, such as adaptive mesh refinement, and to build complex multi-physics simulations by coupling different models, of both PDE and DAE type, in a straightforward manner using Modelica. We illustrate our method with an example that couples a Modelica Proportional-Integral-Derivative controller to a PDE solver for the unsteady heat equation in a 3D domain.

Introduction

We discuss numerical simulation of models that couple Partial Differential Equations (PDEs) and Differential-Algebraic Equations (DAEs) in the context of the Modelica modeling and simulation language [15, 7]. Modelica originated around the idea of solving complex, coupled dynamical systems, which can be described by systems of Ordinary Differential Equations (ODE) or DAE. Up to now, there is only limited support for working with PDEs, despite the fact that the number of Modelica users in academia and in industry has grown significantly lately. One of the first attempts to incorporate PDE support into Modelica is described in [19], [20], and in Chapter 8 of [7], which investigates two different approaches: (1) expressing the PDEs using a combination of new language constructs and a supporting Modelica PDE library using the methodof-lines; (2) exporting the PDE part to an external PDE FEM C++ tool which solves the PDE part of the total problem. Based on this work, an experimental implementation of PDE support was added to the OpenModelica [4] compiler. However, this implementation has not been maintained, even though there have recently been discussions in the OpenModelica community about re-activating these features. Only one simple PDE operator is currently in the Modelica language specification: spatial distribution for 1D PDEs.

In [3] a Modelica library with basic building blocks for solving onedimensional PDE with spatial discretizations based on the method of lines or finite volumes is described. Although this approach is attractive due to its simplicity, it is not clear how it could be extended to higher dimensions, without increasing the complexity significantly. Another approach is described in [11], which extends the modeling language with primitives for geometry description and boundary/initial conditions, and uses an external pre-processing tool to convert the PDE model to a DAE based on the method of lines. In both of these two works, the PDE system is expanded early on in the compilation process. In this way important information of the PDE structure is lost, information that could have been used for mesh refinement and adjustment of the run-time solver.

Another similar option is to use the commercial MapleSim environment [14]: It means to write the PDEs in a Maple component, to export this component to DAE form using a discretization scheme 1 and to use the resulting component in MapleSim, which supports the Modelica language. An overview of how to use Maple and MapleSim together for PDE modeling can be found in [10]. Apart from the cost for licenses, this method again has the same drawback, arising from the loss of information regarding the original model.

In this work, we propose a way to allow for PDE modeling with Modelica by importing C++ components, written with the HiFlow3 multi-purpose finite element software [22], into Modelica using the Functional Mock-Up Interface (FMI) [6] import. FMI is a standard for model exchange and co-simulation between different tools. FMI supports only C but with correct linking it is possible to execute with C++ code. We use the OpenModelica [4] development environment but the same approach can be adapted to other Modelica environments. A similar approach was used in [13, 12], which describes a simulation of the energy supply system of a house using Dymola, ANSYS CFD and the TISC co-simulation environment. Some of the products used in that work are proprietary, however, whereas our environment is based on open standards and open source software. Furthermore, we use the 'model import' approach for the coupling between components, instead of the 'cosimulation' approach applied in those works.

The method described in this paper has several advantages:

1. HiFlow3 is well maintained and has strong support and capabilities for PDE modeling and solving;
2. HiFlow3 and OpenModelica are free to download and use;
3. The PDE structure is not lost but is maintained throughout the actual run-time simulation process. This allows for mesh refinement, solver run-time adjustments, etc.;
4. It is possible to mix PDE and DAE systems in the same system setting. This is also possible in [19].

As a proof-of-concept to demonstrate the Modelica-HiFlow3 integration, we have implemented and tested a coupled model which consists of solving the heat equation in a 3D domain and controlling its temperature via an external heat source. This source consists of a Modelica Proportional-Integral Derivative controller (PID controller), which is taken directly from the Modelica standard library.

The outline of the paper is as follows. In Section 1, we introduce the physical scenario we wish to simulate, and provide a mathematical overview of the two main components in the model. Section 2 describes in detail the realization of the simulation based on coupling of existing software components. In Section 3, we provide numerical simulation results for the example model, followed by a discussion in Section 4, and conclusions in Section 5.

1 Simulation Scenario

We consider the evolution of the temperature distribution in a rectangular piece of copper. Figure 1 shows the setup of the system. A heat sensor is attached on the right side of the copper bar, and on the bottom side there is an adjustable heat source. The system is exposed to environmental influences through time-varying boundary conditions at the top and left sides. The goal is to control the temperature in the material by adjusting the heat source, so that a desired temperature is reached at the point of measurement. The regulation of the heat source is done by a PID controller. It uses the sensor value and a reference temperature to compute the heat source strength. In our simulation, the two entities in this system PID controller copper bar measuring point heat source are realized by reusing existing software components.

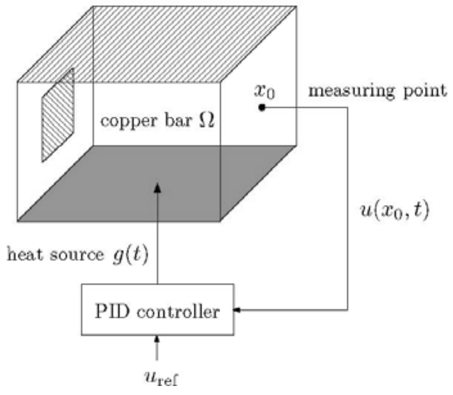

Figure 1: System consisting of a copper bar connected to a temperature regulator based on a PID controller.

The temperature of the copper bar is computed using a HiFlow3 solver, and the 'LimPID' controller using a model from the Modelica standard library. The components are coupled by importing the HiFlow3 solver as a Modelica model using FMI, and then creating a third Modelica model, which connects these two components as well as some additional components.

1.1 Computing the temperature distribution

The evolution of the temperature distribution is modeled by the unsteady heat equation. In this subsection, we give the mathematical problem formulation, discuss the numerical treatment of the heat equation, and describe the discretization we used in the computations.

Heat Equation

We consider the copper to occupy a domain

$$\Omega := (0{,}0.045) \times (0{,}0.03) \times (0{,}0.03) \subset \mathbb{R}^3,$$

where the boundary of Ω is denoted by $\partial\Omega$. The heat problem formulation for our simulation scenario is the following:

Find a function $u : \Omega \times (0,T) \to \mathbb{R}$ as the solution of

$$\partial_t u - \alpha\Delta u = 0 \text{ in } \Omega \times (0,T), \tag{1a}$$

$$u(0) = u_0 \text{ in } \Omega \tag{1b}$$

$$u = g \text{ on } \Gamma_{src} \times (0,T), \tag{1c}$$

$$u = u_{top} \text{ on } \Gamma_{top} \times (0,T), \tag{1d}$$

$$u = u_{left} \text{ on } \Gamma_{left} \times (0,T), \tag{1e}$$

$$\frac{\partial u}{\partial n} = 0 \text{ on } \Gamma_{iso} \times (0,T) \tag{1f}$$

The unsteady heat equation (1a) is a parabolic PDE. Its solution, the unknown function u, describes the evolution of the temperature in the copper bar Ω during the time interval $(0,T)$. Here, $\alpha = 1.11 \times 10^{-4} \left[\frac{m^2}{s}\right]$ denotes the thermal diffusivity of the copper. u_0 in equation (1b) is the initial state at time $t = 0$. The sensor is placed at the point $x_0 := (0.045, 0.015, 0.015)$, where the temperature $u(x_0, t)$ is taken as measurement value for time $t \in (0,T)$. The heat source is modeled by the Dirichlet boundary condition (1c). The controlled temperature $g(t)$ is prescribed on the source part of the boundary

$$\Gamma_{src} := [0{,}0.045] \times [0{,}0.03] \times \{0\} \subset \partial\Omega$$

The environmental influence is modeled by the Dirichlet boundary conditions (1d) and (1e). At the top boundary part

$$\Gamma_{top} := [0{,}0.045] \times [0{,}0.03] \times \{0.03\} \subset \partial\Omega$$

and the left boundary part

$$\Gamma_{left} := \{0\} \times [0.01{,}0.02] \times [0.01{,}0.02] \subset \partial\Omega$$

the temperature is given by the functions $u_{top}(t)$ and $u_{left}(t)$ for $t \in (0,T)$, respectively. The rest of the boundary

$$\Gamma_{iso} := \partial\Omega \setminus (\Gamma_{\text{top}} \cup \Gamma_{\text{left}} \cup \Gamma_{\text{src}})$$

is isolated through the homogeneous Neumann boundary condition (1f).

Variational Formulation

A well-established method for numerically solving PDEs like the heat equation is the finite element method. Here we briefly describe the numerical treatment of problem (1). The methods of this section are taken from [2] and [5].

The finite element discretization is based on a variational formulation, which can be derived as follows. We denote by $C^k(X;Y)$ the set of all k times continuously differentiable functions from X to Y, and by $C_0^\infty(X;Y)$ the set of all smooth functions with compact support. In the common case $X = \Omega, Y = \mathbb{R}$, we just write $C^k(\Omega)$. Assuming that there is a classical solution

$$u \in C^1(0,T; C^2(\Omega) \cap C(\bar{\Omega}))$$

of problem (1), equation (1a) is multiplied by a test function $v \in C_0^\infty(\Omega)$ and integrated over Ω:

$$\int_\Omega (\partial_t u) v \, dx - \alpha \int_\Omega (\Delta u) v \, dx = 0 \tag{2}$$

Green's first identity [9] is applied to the second term of (2), giving

$$\int_\Omega (\Delta u) v \, dx = \int_{\partial\Omega} \frac{\partial u}{\partial n} v \, ds - \int_\Omega \nabla u \cdot \nabla v \, dx$$

$$= -\int_\Omega \nabla u \cdot \nabla v \, dx,$$

where n is the outer unit normal on $\partial\Omega$. The boundary integral vanishes since v is zero on $\partial\Omega$. This leads to

$$\int_\Omega (\partial_t u) v \, dx + \alpha \int_\Omega \nabla u \cdot \nabla v \, dx = 0 \tag{3}$$

For equation (3) to be well-defined, weaker regularity properties of u and v than in the classical context are sufficient. The problem can be formulated in terms of the Lebesgue space $L^2(\Omega)$ of square-integrable functions and the Sobolev space $H^1(\Omega)$ of functions in $L^2(\Omega)$ with square-integrable weak derivatives. We define the solution space

$$V := \{v \in H^1(\Omega): v = 0 \text{ on } \Gamma_{left} \cup \Gamma_{top} \cup \Gamma_{src}\}$$

and the bilinear form

$$a: H^1(\Omega) \times H^1(\Omega) \to \mathbb{R},$$

$$a(u,v) := \alpha \int_\Omega \nabla u \cdot \nabla v$$

Note that a is symmetric, continuous and V-elliptic. We denote by $(u,v)_{L^2} = \int_\Omega uv\,dx$ the standard inner product in $L^2(\Omega)$. Now we can state a variational formulation of problem (1):
Find $u \in \bar{u} + C^1(0,T;V)$ as the solution of

$$(\partial_t u, v)_{L^2} + a(u,v) = 0 \qquad \forall v \in V \tag{4a}$$

$$u(0) = u_0 \tag{4b}$$

where $\bar{u} \in C^1(0,T;H^1(\Omega))$ is a given function fulfilling the Dirichlet boundary conditions

$$\begin{aligned} \bar{u} &= g && \text{on } \Gamma_{left} \times (0,T), \\ \bar{u} &= u_{top} && \text{on } \Gamma_{top} \times (0,T), \\ \bar{u} &= u_{src} && \text{on } \Gamma_{src} \times (0,T), \end{aligned}$$

This variational formulation admits a unique solution u, which is called a weak solution of the heat equation.

Finite element discretization in space

Let $T_h := \{K_1, \dots, K_N\}$ be a triangulation of Ω with N tetrahedral cells $K_i (i = 1, \dots, N)$. We define the finite element space of piecewise linear functions

$$V_h := \{v \in V : v|_K \text{is linear } (K \in T_h)\}. \tag{5}$$

V_h has the finite dimension $n := \dim(V_h)$. We give the problem formulation for a conforming finite element approximation of (4):
Find $u_h \in \bar{u}_h + C^1(0,T;V_h)$ as the solution of

$$(\partial_t u_h, v_h)_{L^2} + a(u_h, v_h) = 0 \qquad \forall v_h \in V_h, \tag{6a}$$

$$(u_h(0), v_h)_{L^2} - (u_0, v_h)_{L^2} = 0 \qquad \forall v_h \in V_h. \tag{6b}$$

Let $\{\varphi_1, \dots, \varphi_n\}$ be a basis of V_h. We define the ansatz

$$u_h(x,t) := \sum_{i=1}^{n} w_i(t)\varphi_i(x)$$

and insert it into (6a), yielding

$$\sum_{i=1}^{n} \dot{w}\,(\varphi_i, \varphi_j)_{L^2} + \sum_{i=1}^{n} w_i a(\varphi_i, \varphi_j) = 0 \quad (j = 1, \dots, n).$$

This can be written as

$$M\dot{w} + Aw = 0,$$

where

$$M := \left((\varphi_j, \varphi_i)_{L^2}\right)_{i,j=1,\dots,n} \in \mathbb{R}^{n \times n}$$

is the mass matrix and

$$A := \left(a(\varphi_i, \varphi_j)\right)_{i,j=1,\dots,n} \in \mathbb{R}^{n \times n}$$

is the stiffness matrix of the problem, and

$$w: (0,T) \to \mathbb{R}^n$$

is the vector of the time-dependent coefficients. From (6b), an initial condition for w is derived as

$$(Mw_0)_i = (u_0, \varphi_i)_{L^2} \qquad (i = 1, \dots, n)$$
$$\Leftrightarrow w_0 = M^{-1}b,$$

where $b_i = (u_0, \varphi_i)_{L^2}$ $(i_1, \dots, n)$. Thus, the finite element discretization in space leads to the initial value problem

$$M\dot{w} + Aw = 0, \tag{7a}$$
$$w(0) = w_0 \tag{7b}$$

for the coefficient vector w.

Implicit Euler discretization in time

As will be discussed in Section 2.6, limitations in the used technology restrict us to use a relatively simple ODE solver for the time discretization. For the heat equation the implicit Euler scheme is suitable, due to its good stability properties [8]. Let $\{0 = t_0 < t_1 <\ldots< t_m = T\}$ be a partition of the time interval with step sizes $\partial t_k = t_{k+1}$ $(k = 0, \dots, m-1)$. The implicit Euler time stepping method for problem (7) is defined as

$$[M + \partial t_k A]w(t_{k+1}) = Mw(t_k) \tag{8}$$

for $k = 0, \ldots, m-1$. This method is convergent and has first-order accuracy in terms of the step size ∂t_k.

1.2 PID controller

A Proportional-Integral-Derivative controller (PID controller) is a form of loop feedback controller, which is widely used to control industrial processes. The controller takes as input a reference value u_{ref}, which represents the desired temperature, and the measured value $u(x_0, t)$. It uses the error $e(t) := u_{ref} - u(x_0, t)$ to compute the output signal $g(t)$. As the name PID suggests, there is a proportional part that accounts for present errors, an integral part that accounts for the accumulation of past errors, and a derivative part that predicts future errors:

$$g(t) = w_P e(t) + w_I \int_0^t e(\tau)\,d\tau + w_D \frac{d}{dt} e(t)$$

Here, w_P, w_I and w_D are weight parameters. By tuning these parameters the performance of the controller can be adapted to a specific process. A PID controller is widely regarded as the best controller when information of the underlying process is lacking, but the use of a PID controller does not guarantee optimal control.

The tuning of the parameters can be done manually or by using a formal method such as Ziegler-Nichols or Cohen-Coon. There are also software tools available. Sometimes one or several of the parameters might have to be set to zero. For instance, derivative action is sensitive to measurement noise thus this part of the controller might have to be omitted in some situations, resulting in a PI controller. PID controllers are linear and can therefore have problems controlling non-linear systems, such as Heating, Ventilation and Air Conditioning (HVAC) systems. [1]

2 Coupled Implementation

In this section, we first briefly introduce the technologies used in the present work. We then describe the coupled simulation setup, as well as its two main constituent components in more detail.

2.1 The Modelica language

Modelica is a language for equation-based object-oriented mathematical modeling which is being developed and standardized through an international effort in the Modelica Association [15]. The equation parts of Modelica requires a lot of the compiler developer: knowledge in compiler construction, symbolic manipulation of equations and associated mathematics, as well as knowledge of numerical mathematics. The simulation run-time system is also an important part and can be complex; various solver techniques for solving the differential equations can be applied. Modelica allows high-level concepts such as object-oriented modeling and component composition. Multi-domain modeling is also possible in Modelica with the possibility of combining model components from a variety of domains within the same application. There exist several mature and well-maintained Modelica development environments, such as Dymola, OpenModelica, MapleSim,Wolfram SystemModeler, Simulation X, and JModelica.org.

2.2 The OpenModelica environment

OpenModelica is a modeling and simulation environment, which is developed and supported by an international consortium, the Open Source Modelica Consortium (OSMC) [4]. This effort includes an open-source implementation of a Modelica compiler, a simulator and a development environment for research, education and industrial purposes.

2.3 The HiFlow3 Finite Element Library

HiFlow3 [22] is a multi-purpose finite element software providing powerful tools for efficient and accurate solution of a wide range of problems modeled by partial differential equations (PDEs). Based on object-oriented concepts and the full capabilities of C++ the Hi- Flow3 project follows a modular and generic approach for building efficient parallel numerical solvers. It provides highly capable modules dealing with mesh setup, finite element spaces, degrees of freedom, linear algebra routines, numerical solvers, and output data for visualization. Parallelism - as the basis for high performance simulations on modern computing systems - is introduced at two levels: coarse-grained parallelism by means of distributed grids and distributed data structures, and fine-grained parallelism by means of platform-optimized linear algebra back-ends.

2.4 The Functional Mock-Up Interface (FMI)

The Functional Mock-Up Interface (FMI) [6, 16] is a tool-independent standard to support both model exchange and co-simulation of dynamic models which can be developed with any language or tool. Models that implement the FMI can be exported as a Functional Mock-Up Unit (FMU). Such a FMU consists mainly of two parts: (1) XML file(s) describing the interface and (2) the model functionality in compiled binary or C code form. Other tools or models, which also implement the FMI, can import Functional Mock- Up Units. The initial FMI development was done in the European ITEA2 MODELISAR project [17].

2.5 Simulation overview

Figure 2 gives an overview of the simulation setup. To create the PDE component, we reused an existing HiFlow3 application, which solves the boundary value problem for the heat equation (1). In order to import this code into Modelica, we converted it into a Dynamic Shared Object (DSO), which implements the FMI functions and interface descriptions necessary to build a Functional Mock-Up Unit. We then loaded this FMU via FMI into our Modelica model. The details of the PDE component are described in Section 2.6.

For the PID controller, we used the LimPID component from the Modelica standard library. This was connected to the PDE component in a new Modelica model, which is described in Section 2.7.

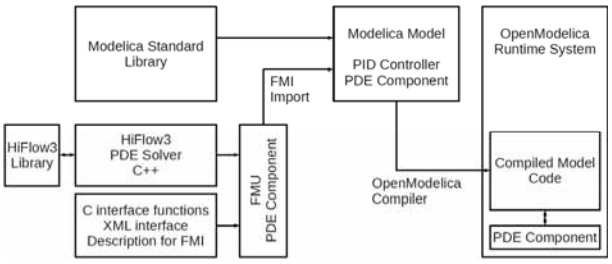

Figure 2: Overview of the creation and coupling of the simulation components.

This model was then compiled with the OpenModelica compiler and executed using the OpenModelica run-time system. By choosing the *Euler* solver, the runtime system provides the time-stepping algorithm according to the implicit Euler scheme, and additionally solves the equations for the PID controller component in each time step. Figure 3 illustrates on a time axis the calls made to the compiled model code, which in turn makes calls to the PDE component.

2.6 HiFlow3-based PDE component

The main sub-part of the PDE component is the `HeatSolver` class, which is a slightly modified version of an existing HiFlow3 application. This class uses data structures and routines from the HiFlow3 library to solve the heat problem (1) numerically. It uses a finite element discretization in space and an implicit Euler scheme in time as described before.

Furthermore, this class provides functions for specifying the current time, the controlled temperature of the heat source, the top and bottom temperatures, and for retrieving the temperature at the measuring point. The top level routine of the `HeatSolver` class is its `run()` function, see Listing 1. This function computes the solution of the heat equation.

We prepared the triangulation Th in a preprocessing step and stored it in a mesh file. When the `run()` function is called for the first time, it reads the mesh file and possibly refines the mesh. It also creates the data structures representing the finite element space V_h from (5), the linear algebra objects representing the system matrix, the right-hand-side vector and the solution vector. Then, the `run()` function assembles the system matrix $M + \delta t_k A$ and the right-hand-side vector $Mw(t_k)$ according to (8). It computes the solution vector $w(t_{k+1})$ for the new time step $t = t_{k+1}$ using the conjugate gradient method [18]. The solution is stored in *VTK* format [21] for visualization.

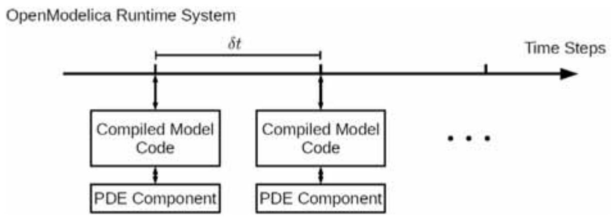

Figure 3: Interaction between the OpenModelica runtime system and the coupled model with the implicit Euler solver.

```
HeatSolver_run() {
  // if this is the first call
  if (first_call) {
  // read mesh file and eventually refine it
   build_initial_mesh();
  //initialize the finite element space and
  //the linear algebra
  //data structures
   prepare_system();
   first_call = false;
  }
  //compute the system matrix and
  //the right-hand-side vector
  assemble_system();
  // solve the linear system
  solve_system();
  // visualize the solution
  visualize();
  //keep solution and time in memory
  CopyFrom(prev_solution,old_solution);
}
```

Listing 1: Run function of the HeatSolver class.

It is important to note that the solution vector and the current time are kept in memory inside the PDE component, since this data is required for computing the solution at the next time step. Although it has been planned for a future version, at present the FMI standard does not directly support arrays, which prevents passing the solution vector back and forth between the PDE component and the Modelica environment as a parameter [16]. Although this use of mutable internal state in the PDE component might be preferable from a performance point of view, it has the drawback of making the function calls referentially opaque: two calls with the same parameters can yield different results, depending on the current internal state. This imposes a strong restriction on the solver used, which must assure that the sequence of time values for which the function `run()` is called is non-decreasing.

For this reason, we only tested the method with the simple implicit Euler solver, and verified that the calls were indeed performed in this way. For more complicated solvers, such as DASSL, this requirement is no longer satisfied.

The entry point of the PDE component is the `PDE_component()` function, see Listing 2. This function is called within the Modelica simulation loop. When it is called for the first time, it creates a `HeatSolver` object. It sets the input values for the heat source, the temperatures at the top and bottom boundary, and the current time. Then the `run()` function of the `HeatSolver` is called to compute the solution of the heat equation. Finally, the run counter is incremented and the measurement value is returned.

2.7 Modelica model

Our Modelica model is shown schematically in Figure 4. It contains the PDE component, the PID controller, and four source components. Two of the sources represent the environmental influences, which are given by sinusoidal functions.

They are connected to the PDE component to give the top and left boundary temperatures u_{top} and u_{left} in Equations (1d) and (1e), respectively. One source is connected to the PID controller and gives the reference value u_{ref} for the desired temperature.

```
PDE_component(
double in_Controlled_Temp,
double in_Top_Bdy_Temp,
double in_Bottom_Bdy_Temp,
double in_Time)
{
  // create HeatSolver object if this is the
  //first call
  if (run_counter == 0)
   heat_solver = new HeatSolver();
  // set input values
  heat_solver->set_time(in_Time);
  heat_solver->set_g(in_Controlled_Temp);
  heat_solver->set_top_temp(in_Top_Bdy_Temp);
  heat_solver-
        >set_bottom_temp(in_Bottom_Bdy_Temp);
  // run the HeatSolver
  heat_solver->run();
  // increment the run counter
   run_counter++;
  // return the measurement value
return heat_solver->get_u(); }
```

Listing 2: Main simulation routine of the PDE component.

The fourth source is connected to the dummy state variable of the PDE component. The dummy state variable and its derivative are in the model due to the fact that the OpenModelica implementation of FMI 1.0 import does not allow for an empty state variable vector. There is however nothing in the FMI 1.0 model exchange specification that disallows this. Additionally, the measurement value of the PDE component is connected to the input of the PID controller, and the output signal of the PID controller is connected to the heat source input of the PDE component.

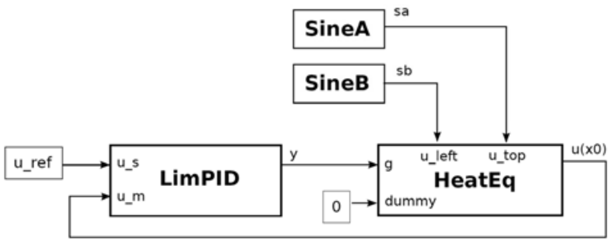

Figure 4: Schematic view of the coupled Modelica model used in the simulation.

The internal constant parameters of the components are summarized in Table 1.

Component Parameter	Value
LimPID	
proportional gain w_p	0.05
integral gain w_I	0.2
derivative gain w_D	0.0
HeatEquationFMU	
thermal diffusivity α	$1.11 \cdot 10^{-4} m^2 s^{-1}$
SineA	
amplitude	0.5°C
vertical offset	3.5°C
start time	$150.0s$
frequency	$0.001s^{-1}$
SineB	
amplitude	6.0°C
vertical offset	3.0°C
start time	$350.0s$
frequency	$0.002s^{-1}$

Table 1: Internal parameters of the components in the simulation model.

3 Results

We carried out a numerical experiment with the following setting. We took a simulation time of $T = 1500$ seconds and a time step of $\partial_t = 1$. We set the initial temperature $u_0 = 0$ everywhere in the computational domain, and we specified the desired temperature as $u_{ref} = 3$. On the upper part of the boundary Γ_{top} and on the left part of the boundary Γ_{left} we modeled environmental influences by the functions

$$u_{top}(t) = \begin{cases} 3.5 & \text{if } t < 150 \\ 3.5 + 0.5 \sin \dfrac{(t-150)\pi}{500} & \text{if } t \geq 150 \end{cases}$$

and

$$u_{left}(t) = \begin{cases} 3 & \text{if } t < 350 \\ 3 + 6 \sin \dfrac{(t-350)\pi}{250} & \text{if } t \geq 350 \end{cases}$$

which are shown in Figure 5.

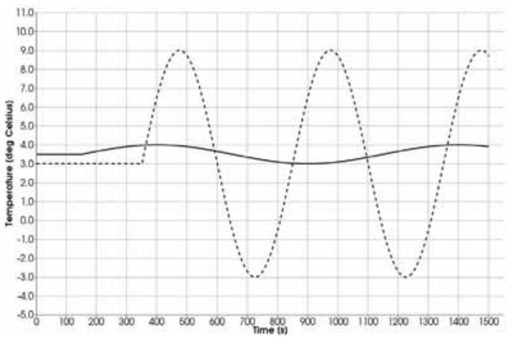

Figure 5: Environmental temperature prescribed on the boundary parts Γ_{left}left and Γ_{top}. Dashed: $u_{left}(t)$, solid: $u_{top}(t)$.

For comparison, we first performed a simulation run with a constant, uncontrolled heat source $g \equiv 3$ on the lower boundary Γ_{src}. Figure 6 shows that the temperature at the point of measurement deviates from the desired temperature $u_{ref} = 3$ due to the environmental influences.

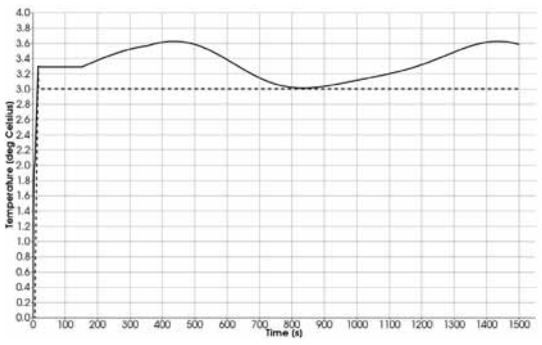

Figure 6: Simulation run with constant heat source $g \equiv 3$. The temperature $u(x_0, t)$ at the point of measurement deviates from the desired value. Dashed: g, solid: $u(x_0, t)$.

The results of our simulation run with a controlled heat source $g = g(t)$ are shown in Figure 7. At the beginning, we fixed the heat source at $g = 2.5$ to let the temperature distribution in the copper bar evolve from the initial state to an equilibrium, at which the measured temperature is slightly higher than desired. The PID controller was switched on at $t = 50$ to take control over the heat source. The curves show that the controller first cools the bottom to bring the temperature at the point of measurement down to the target value. During the rest of the simulation, the PID controller reacts to the environmental influences and adjusts the heat source dynamically over time, so that the temperature accurately follows the desired state. Figures 8-10 illustrate the temperature distribution in the copper bar.

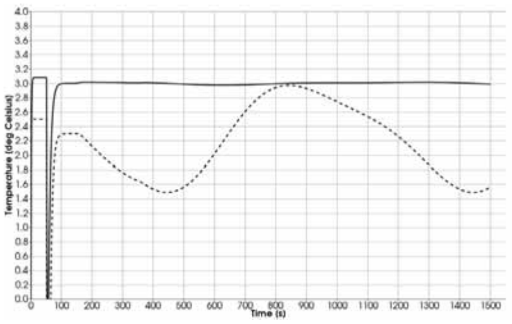

Figure 7: Simulation run with controlled heat source $g = g(t)$. The temperature $u(x_0, t)$ at the point of measurement accurately follows the desired value $u_{ref} = 3$. Dashed: $g(t)$, solid: $u(x_0, t)$.

4 Discussion

The numerical results for the example presented in the previous section show that our method of integrating the PDE solver from HiFlow3 into a Modelica simulation functions correctly. The realization of this particular scenario serves as an illustration of how one can integrate other, more complicated, PDE models into the complex dynamical simulations for which Modelica is especially suited.

The coding and maintenance effort for importing an existing PDE model implemented in HiFlow3 with the method presented here is minimal: in essence only a set of wrapper functions dealing with input and ouput of parameters and state variables is all that is required. The fact that HiFlow3 is free and open source software simplifies the process greatly, since it makes it possible to adapt and recompile the code. This is significant, since the FMI model import requires the component to be available either as C source code or as a dynamic shared object, which is loaded at run-time.

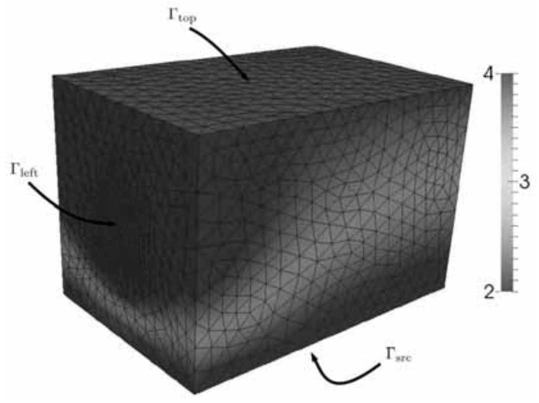

Figure 8: Computational domain of the copper bar with triangulation. The colors indicate the temperature distribution on the surface at time $t = 440$.

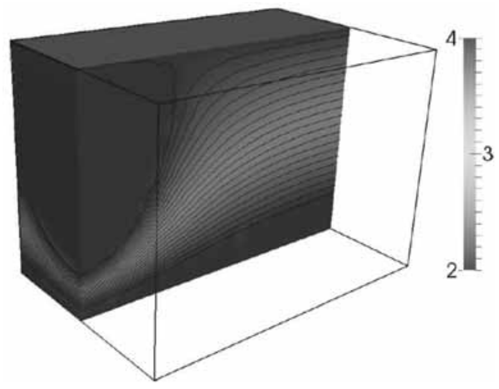

Figure 9: Sectional view with isothermal lines at time $t = 440$.

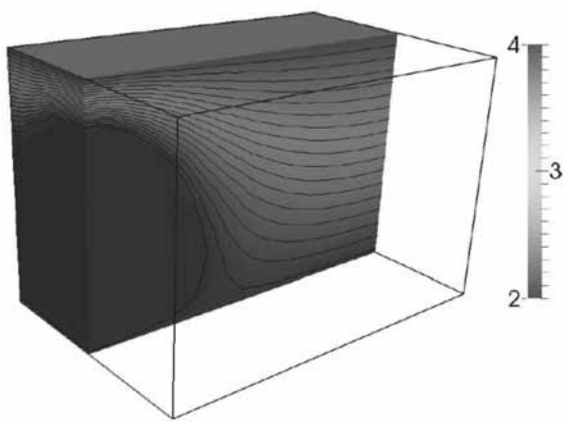

Figure 10: Sectional view with isothermal lines at time $t = 1250$.

Compared to the efforts aiming at extending the Modelica language with support for PDEs, we are working at a different level of abstraction, namely that of software components. The advantage of this is the ability to make use of the large wealth of existing implementations of solvers for various models, in the present case the multipurpose HiFlow3 library. Extending the Modelica language would also make it considerably more complex, since problems for PDEs generally require descriptions of the geometry and 7 the conditions applied to the different parts of the boundary. Furthermore, using this information to automatically generate a discretization and a solution algorithm would require a sophisticated classification of the problem, since different types of PDEs often require different numerical methods. A drawback of working at the software level is that the mathematical description of the problem is not directly visible, as it would be if it was part of the language.

In contrast to the use of 'co-simulation', in this work we have chosen to import the PDE component into the OpenModelica environment, and to make use of one of its solvers. The main benefit is again simplicity: very few changes were required to the PDE component itself, and it was possible to maximize the reuse of existing software. On the other hand, co-simulation, where each submodel has its own independent solver, which is executed independently of the others, also has its advantages. In particular, specialized, highly efficient solution algorithms can be applied to each part of the model, and it is possible to execute the various components in parallel. We are considering to extend the present work to make it usable in a co-simulation setting. Furthermore, we want to investigate the parallelization of the simulation both within and between components.

5 Conclusion

In this paper we have investigated a method of incorporating PDEs in the context of a Modelica model, by using FMI to import a PDE solver from the finite element library HiFlow3. Numerical results obtained using a simple coupled model controlling the heat equation using a PID controller demonstrate that this method works in practice. The main advantages of this type of coupling include its simplicity and the possibility to reuse existing efficient and already validated software. This approach allows to make use of more complex PDE models including high-performance, parallel computations. It has the potential of greatly simplifying the development of large-scale coupled simulations. In this case, however, an extension of the method presented here to support co-simulation might be necessary.

Acknowledgments

Work in this publication was funded by the ITEA2 European Union MODRIO Project, by SSF in the EDOp project, and by the Swedish National Graduate School of Computer Science (CUGS).

This contribution is a post-conference publication from SIMS 2013 Conference (54th SIMS Conference, Bergen University College, Norway, October 16-18, 2013). The contribution is a modified publication from the paper published in the Proceedings of SIMS 2013, to be found at http://www.scansims.org/sims2013/SIMS2013.pdf.

References

[1] Åström KJ, Hägglund T. *PID Controllers: Theory, Design, and Tuning.* Durham: The Instrumentation, Systems, and Automation Society, 1995. 343 p.

[2] Braess, D. *Finite Elemente*. Heidelberg: Springer, 2007. 376 p.

[3] Dshabarow, F. *Support for Dymola in the Modeling and Simulation of Physical Systems with Distributed Parameters* [master thesis]. [Department of Computer Science, Institute of Computational Science, (CH)] ETH Zürich; 2007.

[4] *The Open-Source OpenModelica Development Environment*. http://www.openmodelica.org.

[5] Ern A, Guermond JL. *Theory and Practice of Finite Elements*. New York: Springer; 2010. 526 p.

[6] *The Functional Mockup Interface* (FMI). https://www.fmi-standard.org/.

[7] Fritzson, P. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004. 944 p.

[8] Hairer E, Norsett SP, and Wanner G. *Solving Ordinary Differential Equations*. Springer, 2008. 528 p.

[9] Heuser, H. *Lehrbuch der Analysis 2*. Stuttgart: Teubner, 2002. 737 p.

[10] Khan, S. *Discretizing pdes for maplesim*. adept scientific plc. 2012.

[11] Li Z, Zheng L, and Zhang H. Solving pde models in modelica.In *ISISE*. Proceedings of a meeting; 2008, Dec; Shanghai. IEEE Computer Society; 53–57.

[12] Ljubijankic M, Nytsch-Geusen C. 3D/1D Co-Simulation von Raumlufströmungen und einer Luftheizung am Beispiel eines thermischen Modellhauses. *Fourth German-Austrian IBPSA Conference*, *BauSIM* 2012;167-175.

[13] Ljubijankic M, Nytsch-Geusen, Rädler J, Löffler M. Numerical coupling of Modelica and CFD for building energy supply systems. In: Clauß C, editor. *8th International Modelica Conference*; 2011 Mar; Dresden. Linköping: The Modelica Association and Linköping University Electronic Press. 286-294. doi: 10.3384/ecp11063

[14] Maple and MapleSim by MapleSoft. http://www.maplesoft.com/products.

[15] Modelica and the Modelica Association. http://www.modelica.org.

[16] Modelica Association. *Functional Mock-up Interface for Model Exchange and Co-Simulation*, v. 2.0 beta 4 edition, 2012. https://www.fmi-standard.org/downloads [Accessed 2013-08-06].

[17] *The ITEA2 MODELISAR Project*. http://www.itea2.org/project/index/view/?project=217.

[18] Saad, Y. *Iterative Methods for Sparse Linear Systems*. 2000.

[19] Saldamli L. *PDEModelica - A High-Level Language for Modeling with Partial Differential Equations* [dissertation]. [Department of Computer and Information Science]. Linköping University, 2006. http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-7281.

[20] Saldamli L, Bachmann B, Wiesmann HJ, Fritzson P. A framework for describing and solving PDE models in modelica. In: Schmitz G, editor. Proceedings of the *4th International Modelica Conference*; 2005 Mar; Hamburg, Germany. Hamburg: The Modelica Association and Hamburg University of Technology. 113-122.

[21] Schroeder W, Martin K, and Lorensen B. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. 4th ed. Kiteware, Inc., 2006. 528 p.

[22] *The HiFlow3 Multi-Purpose Finite Element Software*. http://www.hiflow3.org/.