Partial Differential Equations in Modelica

26th August 2014

Contents

Chapter 1

Modelica extension for PDE

1.1 Requests on language extension and possible approaches

Space & coordinates

What should be specified

- Dimension of the problem (1,2 or 3D)
- ?? Coordinates (cartesian, cylindrical, spherical ...) – where this information will be used (if at all):
  - in differential operators as grad, div, rot etc.
  - in visualization of results
  - ?? in computation – perhaps equations should be transformed and the calculation would be performed in cartesian coordinates
- Names of independent (coordinate) variables (x, y, z, r, φ, θ...)

Perhaps all these should be specified within the domain deffinition.
Dimension can be infered from number of return values of shape-function or different properities of the domain in other cases.
The base coordinates would be cartesian and they would be always implicitly defined in any domain. Besides that other coordinate systems could be defined also.
Names of independent variables in cartesian coordinates should be fixed *x, (x,y), (x,y,z)* in 1D, 2D and 3D domains respectively.

Domain & boundary

What should be specified

- the domain where we perform the computation and where equations hold
- boundary and its subsets where particular boundary conditions hold
- normal vector of the boundary

Possible approaches

Parametrization of the domain with shape function and intervals – from The Book (Principles of ...), section 8.5.2

Example from the book:

```
model HeatCircular2D
        import DifferentialOperators2D.*;
        parameter DomainCircular2DGrid omega;
        FieldCircular2DGrid u(domain=omega, FieldValueType=SI.Temperature);
equation
        der (u) = pder (u,D.x2)+ pder (u,D . y 2 )      in omega.interior;
        nder(u) = 0               in omega.boundary;
end HeatCircular2D;

record DomainCircular2DGrid "Domain being a circular region"
        parameter Real radius = 1;
        parameter Integer nx = 100;
        parameter Integer ny = 100;
        replaceable function shapeFunc = circular2Dfunc "2D circular region";
        DomainGe2D interior(shape=shapeFunc,interval={{O,radius},{O,l}},geom= ..
        DomainGe2D boundary (shape=shapeFunc, interval={{radius, radius), { 0,1}
        function shapeFunc = circular2Dfunc "Function spanning circular region";
end DomainCircular2DGrid;

function circular2Dfunc "Spanned circular region for v in interval 0..1"
        input Real r,v;
        output Real x,y;
algorithm
        x : = r*cos (2*PI*v);
        y := r*sin(2*PI*v);
end circular2Dfunc;

record FieldCircular2DGrid
        parameter DomainCircular2DGrid domain;
```

```
        replaceable type FieldValueType = Real;
        replaceable type FieldType = Real[domain.nx,domain.ny,domain.nz];
        parameter FieldType start = zeros(domain.nx,domain.ny,domain.nz);
        FieldType Val;
end FieldCircular2DGrid;
```

And modified version, where all numerical stuff (grid, number of points – this should be configured using simulation setup or annotations) omitted, modified `pder` operator, `Field` as Modelica build-in type:

```
model HeatCircular2D
        parameter DomainCircular2D omega(radius=2);
        field Real u(domain=omega, start = 0, FieldValueType=SI.Temperature);
equation
        pder(u,time) = pder(u,x)+ pder(u,y) in omega.interior;
        pder(u,omega.boundary.n) = 0    in omega.boundary;
end HeatCircular2D;

record DomainCircular2D
        parameter Real radius = 1;
    parameter Real cx = 0;
        parameter Real cy = 0;
        function shapeFunc
                input Real r,v;
                output Real x,y;
        algorithm
                x := cx + radius*r * cos(2 * C.pi * v);
                y := cy + radius*r * sin(2 * C.pi * v);
        end shapeFunc;
        Region2D interior(shape = shapeFunc, interval = {{O,1},{O,1}});
        Region1D boundary(shape = shapeFunc, interval = {{1,1},{0,1}});
end DomainCircular2D;
```

Description by the boundary Domain is defined by closed boundary curve, which may by composed of several connected curves. Needs new operator *interior* and type *Domain2d* (and *Domain1D* and *Domain3d*). (similarly used in FlexPDE – http://www.pdesolutions.com/.)

```
package BoundaryRepresentation
  partial function cur
    input Real u;
    output Real x;
    output Real y;
  end cur;
  function arc
```

```
  extends cur;
  parameter Real r;
  parameter Real cx;
  parameter Real cy;
algorithm
  x:=cx + r * cos(u);
  y:=cy + r * sin(u);
end arc;
function line
  extends cur;
  parameter Real x1;
  parameter Real y1;
  parameter Real x2;
  parameter Real y2;
algorithm
  x:=x1 + (x2 - x1) * u;
  y:=y1 + (y2 - y1) * u;
end line;
function bezier3
  extends cur;
  //start-point
  parameter Real x1;
  parameter Real y1;
  //end-point
  parameter Real x2;
  parameter Real y2;
  //start-control-point
  parameter Real cx1;
  parameter Real cy1;
  //end-control-point
  parameter Real cx2;
  parameter Real cy2;
algorithm
  x:=(1 - u) ^ 3 * x1 + 3 * (1 - u) ^ 2 * u * cx1 + 3 *
      (1 - u) * u ^ 2 * cx2 + u ^ 3 * x2;
  y:=(1 - u) ^ 3 * y1 + 3 * (1 - u) ^ 2 * u * cy1 + 3 *
      (1 - u) * u ^ 2 * cy2 + u ^ 3 * y2;
end bezier3;
record Curve
  function curveFun = line;
  // to be replaced with another fun
  parameter Real uStart;
  parameter Real uEnd;
end Curve;
record Boundary
  constant Integer NCurves;
```

```
    Curve curves[NCurves];
    //     for i in 1:(NCurves-1) loop
    //assert(Curve[i].curveFun(Curve[i].uEnd) = Curve[i
        +1].curveFun(Curve[i+1].uStart), String(i)+"th
        curve and "+String(i+1)+"th curve are not
        connected.",level = AssertionLevel.error);
    //     end for;
    //     assert(curves[NCurves].curveFun(curves[NCurves
        ].uEnd) =
    //                              curves[1].curveFun(curves
        [1].uStart),
    //                              String(NCurves)+"th curve
         and first curve are not connected.",
    //                              level = AssertionLevel.
        error);
  end Boundary;
  record DomainHalfCircle
    constant Real pi = Modelica.Constants.pi;
    arc myArcFun(cx = 0, cy = 0, r = 1);
    Curve myArc(curveFun = myArcFun, uStart = pi / 2,
       uEnd = (pi * 3) / 2);
    line myLineFun(x1 = 0, y1 = -1, x2 = 0, y2 = 1);
    Curve myLine(curveFun = myLineFun, uStart = 0, uEnd =
         1);
    line myLine2(curveFun = line(x1 = 0, y1 = -1, x2 = 0,
         y2 = 1), uStart = pi / 2, uEnd = (pi * 3) / 2);
    Boundary b(NCurves = 2, curves = {myArc,myLine});
    //new externaly defined type Domain2D and operator
        interior:
    Domain2D d = interior Boundary;
  end DomainHalfCircle;
end BoundaryRepresentation;
```

Constructive solid geometry used in Matlab PDE toolbox, http://en.wikipedia.org/wiki/Constructive_sol

Domain is build from primitives (cuboids, cylinders, spheres, cones, user defined shapes ...) applying boolean operations *union*, *intersection* and *difference.*

How to describe boundaries?

Listing of points – export from CAD

Inequalities

Boundary representation (BRep) (NETGEN, STEP)

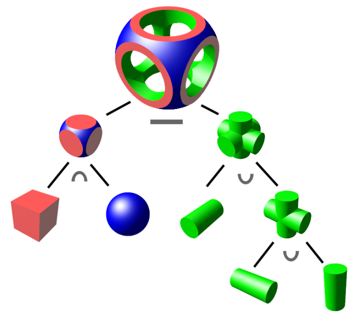

Figure 1.1: constructive solid geometry

Fields

Partial derivative

$\frac{\partial^2 u}{\partial x \partial y}$ ... `pder(u,x,y)`
directional derivative ... `pder(u,omega.boundary.n)`

Equations, boundary and initial conditions

Use the *in* operator to express where equations hold.

1.2 New concepts and language elements

1.2.1 Domain deffinition

Three concurent aproaches - one has to be chosen:

1.2.1.1 By boundary

This is the aproach from [12]. Defining domains using curves (that have one parameter) to build up the boundary works very well in 2D. Parameters of these curves are bounded in one dimensional interval. In 3D we have to use surfaces (haveing two parameters) instead of curves. Parameters must be bounded in some subsets of R^2 now. If the boundary is composed of several surfaces, parameters of these surfaces must be often bounded not just in Cartesian product of two intervals but in some more complex sets so that these surfaces are continuously connected. There is no way to write this in the current extension. It is possible to just slightly generalize this method and allow specification of more general sets for these parameters. But this is questionable, as user would have to calculate boundaries where surfaces mutually intersect. This could be often quite hard.

```
type Domain
  parameter Integer ndims;
  Real cartesian[ndims];
  Real coord[ndims] = cartesian;
  replaceable Region interior;
  replaceable function shape
    input Real u[ndims];
    output Real coord[ndims];
  end shape;
end Domain;
```

Domain type

There is also a built-in Domain type here to be inherited in all other domain types, but it has different members. It is defined as follows:

1.2.1.2 Using shape-function

Other approach was introduced in [3]. A so called shape-function is used to map Cartesian product of intervals onto the interior and boundaries of the domain. These functions allows to simply generate points in the domain or if inverted gives straightforward rule to determine whether any given point belongs to the domain. This would later simplify generating the computational grid to simulate the model. This description is also perhaps closer to the way how such a region (subsets of $\mathbb{R}^n$) is usually described in mathematics. To allow this approach we need some changes to the language.

1.2.2 Domain type

Any domain type extends built-in `type Domain`, that has two members `replaceable Region interior;` and `parameter Integer ndim;`. Other domains extends this general domain and redeclares `Region interior` into `Region1D`, `2D` or `3D`. During translation are domains treated in special way. There will be packege `PDEDomains` containing library of common domains `DomainLineSegment1D`, `DomainRectangle2D`, `DomainCircular2D`, `DomainBlock3D` and others. User can define other new domain classes.
Needs OMC modification.

1.2.3 Region type

How to evaluate ndimr (equality operator for Reals problematic)?

```
type Region
  parameter Integer ndims;    //dim of space
  parameter Integer ndimr;    //dim of region
  parameter Real[ndims][2] interval;
  replaceable function shape;
    input Real u[ndim];
    output Real coord[ndims];
  end shape;
end Region;
```

1.2.4 `coordinate`

Space coordinate variables are of a different kind than other variables. They are similar to the time variable in Modelica. Both coordinates and time are independent variables. They can get any value from the domain resp. time interval. Other (dependent) variables are actually functions of time and as for fields also of space coordinates. Thus coordinate variables should not hold any particular value of the physical quantity they represent. They have rather a symbolic meaning.

New keyword `coordinate` is used as a modifier to define coordinates. The syntax is

```
"coordinate Real" coordName;
```

Or without Real?

1.2.5 interval

to define parameter interval for a shape-function. E.g. `interval={{0,1},0}`. Used in domain records. (Previously called range.)
New language element.

1.2.6 shape function

one-to-one map of points in k-dimensional interval (usualy cartesian product of intervals) to points in n-dimensional domain and thus deffine a coordinate system in domain
.

region types and regions `Region0D`, `Region1D`, `Region2D`, `Region3D` used in domain records to define interior, boundaries and othere regions where certain equations hold (e.g. connection of PDE and ODE). Two mandatory attributes are `shape` and `interval`. E.g. `Region2D left (shape = shapeFunc, interval = {0,{0,1}})`.
New language element.

1.2.7 fields

A variable whose value depends on space position, is called field. It is defined with keyword `field`. Field can be of either `Real`, `Integer` or `Boolean` type. It can be defined also as a parameter. Field may be an array to represent vector field. Mandatory attribute is `domain`. Other attributes are same as for corresponding regular type (e.g. for `Real: start`, `fixed`, `nominal`, `min`, `max`, `unit`, `displayUnit`, `quantity`, `stateSelect`. (Not shure about `fixed` and `stateSelect`.) Fields may be initialized in initialEquation section or using start attribute in declaration as other variables. Because higher derivatives are allowed for fields it is sometimes necessary to specify start value for some field derivative. This is not a problem in the initialEquation. To initialize field derivative using start attribute we can treat it as an array. Here it is confusing that arrays are indexed starting with 1, so that start[1] is start value the field itself, start[2] is for first derivative etc. They can be assigned either constant values or field literals..
see `3.2.2`

New language element.

> ??Use `in` operator to specifie the domain in field declaration instead of `domain` attribute:
> `field Real u in omega;`

1.2.8 field literal constructor

`"field" "(" expr1 "in" expr2 ")"`,
or just shortcut
`"{" expr1 "in" expr2 "}"`
where `expr2` is a domain and `expr1` may depend on coordinates defined in this domain. E.g.

```
field Real f = field(2*dom.x+dom.y in omega.interior);
```

operations and functions on fields All operators (= , :=, +, -, *, /, ^, <, <= ,> ,>= ,== , <>) and functions can be applied on fields. The result is also a field. If a binary operator or function of more arguments is applied on two (or more) fields, these fields must be defined within the same domain.
If some binary operator or function with more arguments is performed on field and regular variable (it means a variable that is not a field), the operation is performed as if the regular variable is field that is constant in space.

`pder()` operator for partial and directional derivative of real field. Higher derivatives are allowed. E.g. `pder(u,omega.x,omega.x,omega.y)` means $\frac{\partial^3 u}{\partial x^2 y}$. Directional derivative: `pder(u,omega.left.n)`. Time derivative of field must be written also using `pder` operator not `der`.

normal vector implicitly defined for all N-1 dimensional regions in N dimensional domain. (e.g. `omega.left.n`) Used in boundary condition equations. How to write domain and region independently - perhaps `region.n`? New language element.

1.2.9 `in` operator

is used to define where PDE, boundary conditions and other equations hold and to acces field values in particular point (see 3.5). On left is an equation on right is a region where the equation hold.
E.g. `x=0  in omega.left`
New language element.

`dom` **keyword** stands for current domain specified with `in` operator.

```
pder(u,dom.x)=0 in omega.interior;
```

is equivalent to

```
pder(u,omega.x)=0 in omega.interior;
```

Is useful to write equations domain independent.
New language element.

`region` **keyword** stands for current region specified with `in` operator.

```
v*region.n=0 in omega.left, omega.right;
```

is equivalent to

```
v*omega.left.n=0 in omega.left; v*omega.right.n=0 in omega.right;
```

Is useful to get normal vector in domain independent equations and on anonymous regions.
New language element. Not shure if it will be needed.

region addition?? + operator can be used to add regions. Can be used in domain record to form a new region, e.g.

```
boundaries = left + right;
```

or on right side of `in` operator, e.g.

```
x = 0  in omega.left + omega.right;
```

New language element. Not shure if it will be needed.

vector differential operators `grad, div, rot`

1.3 Changes and additions to Levon Saldamlis proposals

Field literal constructor

is modified to handle several different coordinate systems.

Previously

```
"field" "(" expr1 "for" iter "in" aDomain ")"
```

or

```
"{" expr1 "for" iter "in" aDomain "}"
```

where `iter` is one variable or touple of variables of arbitrary name that are binded to coordinates in `aDomain.`

Now

see 1.2.8

Disable accessing field values in function-like style

Accessing field values in function-like style should not be allowed, if possible, for two reasons: first it is not allowed in current Modelica for regular (non-field) variables (that are unknown functions of time) in ODE. We should be in agreement with current Modelica. Second, if more than one coordinate system are defined in a domain, it is not indicated which coordinates are used in the argument. Regions consisting of one point and the `in` operator may be used instead, see 1.2.9.

Initialization of derivatives of fields

in case of field that is differentiated at least twice (in constructor, not in initial equation section). Previously not solved, now see 1.2.7.

Coordinates

new keyword `coordinate`. They were previously member of built-in class `domain`, that was inherited by all other domains.

Chapter 2

Numerics

Goals

1. advection equation in 1D and eulerian coordinate, dirichlet BC, explicit solver
2. numann BC
3. automatic dt
4. diffusion or mixed equation
5. implicit solver
6. systems of equations
7. 2D (rectangle), 3D (cube)
8. lagrangian coordinate
9. general domain

difference schemes separated from the rest of solver

Difussion eq:

$$u_t = \alpha u_{xx}$$

or

$$\begin{aligned} u_t &= -w_x \\ w &= -\alpha u_x. \end{aligned}$$

String eq:

$$y_{tt} = k y_{xx}$$

or

$$
\begin{aligned}
s_x &= kv_t \\
y_t &= v \\
y_x &= s
\end{aligned}
$$

The description without higher derivative is ugly, we need higher derivatives.

Representation

Explicit

$$u_t = f(u, u_x, t) \tag{2.1}$$

resp.

$$u_t = f(u, u_x, u_{xx}, \dots, t)$$

..

Implicit

$$F(u, u_t, u_x, t) = 0 \tag{2.2}$$

resp.

$$F(u, u_t, u_x, u_{xx}, \dots, t) = 0$$

Solvers

Difference schemes for explicit solver
U denotes discretized u
Time difference from Lax-Friedrichs in explicit form (i.e. with the u_m^{n+1} on LHS):

$$u_m^{n+1} = D_t^{exp}(v, U, n, m) = v\Delta t + \frac{1}{2}(u_{m+1}^n + u_{m-1}^n) \tag{2.3}$$

Space difference from Lax-Friedrichs:

$$D_x(U, n, m) = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \tag{2.4}$$

Explicit solver Lax-Friedrichs
We solve equation (2.1) substituing space difference (2.4) and applying time difference in explicit form (2.3):

$$
\begin{aligned}
u_m^{n+1} &= D_t^{exp}(f((u_m^n, D_x(U, n, m), t^n))) = \\
&= \Delta t \cdot f(u, \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x}, t) + \frac{1}{2}(u_{m+1}^n + u_{m-1}^n)
\end{aligned}
$$

Difference schemes for implicit solver space difference from Crank-Nicolson

$$
\begin{aligned}
D_x(u^n_{m-1}, u^n_m, u^n_{m+1}, u^{n+1}_{m-1}, u^{n+1}_m, u^{n+1}_{m+1}) &= \frac{1}{2}\left(\frac{u^{n+1}_{m+1} - u^{n+1}_{m-1}}{2\Delta x} + \frac{u^n_{m+1} - u^n_{m-1}}{2\Delta x}\right) \\
D_{xx}(u^n_{m-1}, u^n_m, u^n_{m+1}, u^{n+1}_{m-1}, u^{n+1}_m, u^{n+1}_{m+1}) &= \frac{1}{2}\left(\frac{u^{n+1}_{m+1} - 2u^{n+1}_m + u^{n+1}_{m-1}}{2(\Delta x)^2} + \frac{u^n_{m+1} - 2u^n_m + u^n_{m-1}}{2(\Delta x)^2}\right)
\end{aligned} \tag{2.5}
$$

time difference from Crank-Nicolson

$$
D_t(u^n_{m-1}, u^n_m, u^n_{m+1}, u^{n+1}_{m-1}, u^{n+1}_m, u^{n+1}_{m+1}) = \frac{u^{n+1}_m - u^n_m}{\Delta t} \tag{2.6}
$$

Implicit solver Crank-Nicolson
With nonlinear solver:
We solve equation (2.2) substituting space (2.5) and time (2.6) differences

$$
F\left(u^n_m, D_t(u^n_{m-1}, ...), D_x(u^n_{m-1}, ...), t^n\right) = 0, \ m \in \hat{M} \tag{2.7}
$$

and than solving the whole system for all unknown $u^{n+1}_{:}$. System has 3-band Jacobian. If F is linear in u_x and u_t, system is also linear with 3-band matrix eventhou is given generaly. Is there any solver eficient in solving linear equations with banded matrix given implicitly? (I hope Newton-Raphson is.) As initial guess for the solution we can use extrapolated values. If solving fails we can try value from the node on left or right (this could help on shocks).

With linear solver:
If F is linear, we expres (2.7) as

$$
\boldsymbol{A}\bar{u}^{n+1}_{:} = \bar{b}.
$$

$\boldsymbol{A}$ is $M \times M$ 3-diagonal. Functions for evaluation of $\boldsymbol{M}$ and $\bar{b}$ are generated during compilation. In runtime we solve just the linear system. In this aproach difference schema must be chosen before compilation of model.

Implicit solver and systems of PDE If we solve e.g. system with three variables u, v, w, se can sort difference equations in order

$$
u_1,\ v_1,\ w_1,\ u_2,\ v_2,\ w_2,\ u_3,\ v_3,\ w_3,\ ...
$$

so that the system is stil banded.

Chapter 3

Example models

3.1 Package PDEDomains

Modelica code of domain definitions:

```
package PDEDomains
  import C = Modelica.Constants;

  type Domain  //Domain is built-in, but this is his "
     interface"
    prameter Integer ndim;
    Coordinate coord[ndim];
    replaceable Region interior;
    replaceable function shapeFunc
      input Real u[ndim-1];
      output Real coord[ndim];
    end shapeFunc;
  end Domain

type Region //Region is built-in, looks like
  parameter Integer ndimS; //dimension of the space,
     where the region exists
  parameter Integer  ndim; //dimension of the region
  //e.g. sphere in 3D has ndimS = 3, ndim = 2
  replaceable function shape;
    input Real u[ndim];
    output Real coord[ndimS];
  end shape;
  parameter Real[ndim][2] interval;
equation
```

```
  assert(ndim <= ndimS, "Dimension of region must be
     lower or equal to dimension of space where it is
     defined.");
end Region;

type Region0D = Region(ndim = 0);
type Region1D = Region(ndim = 1);
type Region2D = Region(ndim = 2);
type Region3D = Region(ndim = 3);




//approach 1:
  class DomainLineSegment1D
    extends Domain;
    parameter Real l = 1;
    parameter Real a = 0;
    redeclare function shapeFunc
      input Real v;
      output Real x = l*v + a;
    end shapeFunc;
    Coordinate x(name = "cartesian") = coord[1];
    Region1D interior(shape = shapeFunc, interval =
       {0,1});
    Region0D left(shape = shapeFunc, interval = 0);
    Region0D right(shape = shapeFunc, interval = 1);
    Region0D boundary = left + right; //{left, right};
  end DomainLineSegment1D;

//approach 2:
  class DomainLineSegment1D
    extends Domain;
    parameter Real l = 1;
    parameter Real a = 0;
    parameter Real b = a + l;
    Coordinate x (name = "cartesian");
    Region1D interior(x in (a,b));
    Region0D left(x = a);
    Region0D right(x = b);
    Region0D boundary = left + right;
  end DomainLineSegment1D;

//approach 1:
  class DomainRectangle2D
```

```
    extends Domain;
    parameter Real Lx = 1;
    parameter Real Ly = 1;
    parameter Real ax = 0;
    parameter Real ay = 0;
    function shapeFunc
      input Real v1, v2;
      output Real x = ax + Lx * v1, y = ay + Ly * v2;
    end shapeFunc;
    Coordinate x (name = "cartesian");
    Coordinate y (name = "cartesian");
    Coordinate r (name = "polar");
    Coordinate phi (name = "polar");
    equation
      r = sqrt(x^2 + y^2);
      phi = arctg(y/x);
    Region2D interior(shape = shapeFunc, interval =
       {{0,1},{0,1}});
    Region1D right(shape = shapeFunc, interval =
       {1,{0,1}});
    Region1D bottom(shape = shapeFunc, interval =
       {{0,2},0});
    Region1D left(shape = shapeFunc, interval =
       {0,{0,1}});
    Region1D top(shape = shapeFunc, interval = {{0,1},1})
       ;
    Region1D boundary = right + bottom + left + top;
    Region1D boundary(union = {right,  bottom, left, top
       });
  end DomainRectangle2D;

//approach 2:
  class DomainRectangle2D
    extends Domain;
    Coordinate x (name = "cartesian");
    Coordinate y (name = "cartesian");
//    Coordinate r (name = "polar");
//    Coordinate phi (name = "polar");
    parameter Real L1 = 1;  //rectangle length, assign
       implicit value
    parameter Real L2 = 1;  //rectangle height, assign
       implicit value
    parameter Real a1 = 0;  //x-coordinate of left side,
       implicitly 0
    parameter Real a2 = 0;  //y-coorinate of lower side,
       implicitly 0
```

```
    parameter Real b1 = a1 + L1;  //x-coordinate of right
         side
    parameter Real b2 = a2 + L2;  //y-coorinate of upper
        side
//     equation
//       r = sqrt(x^2 + y^2);
//       phi = arctg(y/x);
    Region2D interior (x in (a1,b1), y in (a2,b2));  //or
         rather (x,y) in (a1,b1)@(a2,b2)??
    Region1D right    (x = a, y in (a2,b2));
    Region1D bottom   (x in (a1,b1), y = b1);
    Region1D left     (x = a1, y = (a2,b2));
    Region1D top      (x in (a1,b1), y = b2);
    Region1D boundary = right + bottom + left + top;
  end DomainRectangle2D;

//approach 1:
  class DomainCircular2D
    extends Domain;
    parameter Real radius = 1;
    parameter Real cx = 0;
    parameter Real cy = 0;
    function shapeFunc
      input Real r,v;
      output Real x,y;
    algorithm
      x:=cx + radius * r * cos(2 * C.pi * v);
      y:=cy + radius * r * sin(2 * C.pi * v);
    end shapeFunc;
    coordinate x (name="cartesian");
    coordinate y (name="cartesian";
    coordinate cartesian[2] = {x,y};
    // Coordinate r (name="polar");
    // Coordinate phi (name="polar");
    // equation
    //    r = sqrt(x^2 + y^2);
    //    phi = arctg(y/x);
    Region2D interior(shape = shapeFunc, interval = {{O
        ,1},{O,1}});
    Region1D boundary(shape = shapeFunc, interval =
        {1,{0,1}});
  end DomainCircular2D;

//approach 2:
  class DomainCircular2D
    extends Domain;
```

```
    parameter Real radius = 1;
    parameter Real cx = 0;
    parameter Real cy = 0;
    coordinate x (name="cartesian");
    coordinate y (name="cartesian";
    coordinate r (name="polar");
    coordinate phi (name="polar");
    coordinate cartesian[2] = {x,y};
    coordinate polar[2] = {r,phi};
    equation
      x = r*cos(phi) + cx;
      y = r*sin(phi) + cy;
    Region2D interior(phi in (O,2*C.pi), r in (O,radius))
        ;
    Region1D boundary(phi in (O,2*C.pi), r = radius);
  end DomainCircular2D;

//approach 2:
  type DomainElliptic2D
    extends Domain(ndim=2);
    parameter Real cx, cy, rx, ry; //x/y center, x/y
        radius
    coordinate Real cartesian[ndim], x = cartesian[1], y
        = cartesian[2];
    coordinate modPolar[ndim], r = modPolar[1], phi =
        modPolar[2];
    equation
      x = rx*r*cos(phi) + cx;
      y = ry*r*sin(phi) + cy;
    Region2D interior(phi in (O,2*C.pi), r in (O,1));
    Region1D boundary(phi in (O,2*C.pi), r = 1);
  end DomainElliptic2D

//approach 1:
  class DomainBlock3D
    extends Domain(ndim=3);
    parameter Real Lx = 1, Ly = 1, Lz = 1;
    parameter Real ax = 0, ay = 0, az = 0;
    redeclare function shapeFunc
      input Real vx, vy, vz;
      output Real x = ax + Lx * vx, y = ay + Ly * vy, z =
          az + Lz * vz;
    end shapeFunc;
    Coordinate x (name="cartesian");
    Coordinate y (name="cartesian");
    Coordinate z (name="cartesian");
```

```
    coord = {x,y,z};
    Region3D interior(shape = shapeFunc, interval =
        {{0,1},{0,1},{0,1}});
    Region2D right(shape = shapeFunc, interval =
        {1,{0,1},{0,1}});
    Region2D bottom(shape = shapeFunc, interval =
        {{0,1},{0,1},1});
    Region2D left(shape = shapeFunc, interval =
        {0,{0,1},{0,1}});
    Region2D top(shape = shapeFunc, interval =
        {{0,1},{0,1},1});
    Region2D front(shape = shapeFunc, interval =
        {{0,1},0,{0,1}});
    Region2D rear(shape = shapeFunc, interval =
        {{0,1},1,{0,1}});
  end DomainBlock3D;

  //and others ...

end PDEDomains;
```

Listing 3.1: Standard domains deffinitions: 1D – Line segment, 2D – Rectangle, Circle, 3D – Block

3.2 Simple models

3.2.1 Advection equation (1D)[19]

L .. length

c .. constant, assume $c > 0$

$u \in \langle 0, L\rangle \times \langle 0, T\rangle \to \mathbb{R}$

equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

initial conditions

$$u(x, 0) = 1$$

boundary conditions

$$u(0, t) = \cos(2\pi t)$$

Modelica code

```
model advection "advection equation"
  import PDEDomains.*
  import C = Modelica.Constants;
  parameter Real L = 1; // length
  parameter Real c = 1;
  parameter DomainLineSegment1D omega(length = L);
  field Real u(domain = omega, start = 1);
equation
  pder(u,time) + c*pder(u,dom.x) = 0   in omega.
     interior;
  u = cos(2*C.pi*time)                 in omega.
     left;
end advection;
```

Listing 3.2: Advection equation in Modelica

Flat model

```
/*TODO: finish it!!*/
function PDEDomains.DomainLineSegment1D.shapeFunc
  input Real v;
  output Real x = l*v + a;
end PDEDomains.DomainLineSegment1D.shapeFunc;


model advection_flat "advection equation"
  import C = Modelica.Constants;
  parameter Real L = 1; // length
  parameter Real c = 1;
//  parameter DomainLineSegment1D omega(length = L
   );
  parameter Real omega.l = L;
  parameter Real omega.a = 0;




  Domain1DInterior DomainLineSegment1D.interior(
     shape = shapeFunc, range = {0,1});
```

```
  Domain1DBoundary DomainLineSegment1D.left(shape
     = shapeFunc, range = {0,0});
  Domain1DBoundary DomainLineSegment1D.right(shape
      = shapeFunc, range = {1,1});

  field Real u(domain = omega, start = 1);
equation
  pder(u,time) + c*pder(u,x) = 0  in  omega.
     interior;
  u = cos(2*pi*time) in omega.left;
end advection_flat;
```

Listing 3.3: Advection equation – flat model

3.2.2 String equation (1D)[26]

L .. length

$u \in \langle 0, L\rangle \times \langle 0, T\rangle \to \mathbb{R}$ (string position)

c .. constant

equation:

$$\frac{\partial^2 u}{\partial t^2} - c\frac{\partial^2 u}{\partial x^2} = 0$$

initial conditions

$$\begin{aligned} u(x,0) &= \sin\left(\frac{4\pi}{L}x\right) \\ \dot{u}(x,0) &= 0 \end{aligned}$$

boundary conditions

$$u(0,t) = 0, \quad u(L,t) = 0$$

Modelica code

```
model string "model of a vibrating string with fixed ends
   "
  import C = Modelica.Constants;
  parameter Real L = 1; // length
  parameter Real c = 1; // tension/(linear density)
  parameter DomainLineSegment1D omega(length = L);
  parameter field Real u0 = {sin(4*C.pi/L*dom.x) for dom.
     x in omega.ingerior};
```

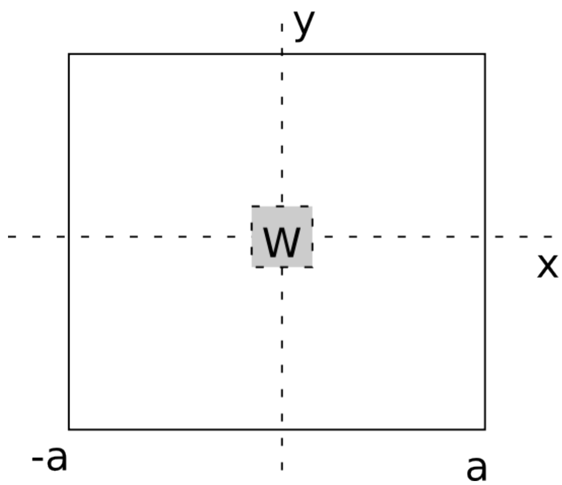

Figure 3.1: Heat eq.

```
  field Real u(domain = omega, start[0] = u0, start[1] =
      0);
equation
  pder(u,time,time) - c*pder(u,x,x) = 0    in omega.
      interior;
  u = 0                                    in omega.left +
       omega.right;
end string;
```

Listing 3.4: String model in Modelica

3.2.3 Heat equation in square with sources (2D)

a .. domain square side hlaf length
c .. conductivity quocient
T .. temperature

$$W(x,y) \quad = \quad \begin{cases} 1 & \text{if } |x| < a/10 \text{ and } y < a/10 \\ 0 & \text{else} \end{cases}$$

equation

$$\frac{\partial T}{\partial t} - c\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \quad = \quad W$$

initial conditions

$$T(x, y, 0) = 0$$

boundary conditions insulated walls (top, left, bottom)

$$\begin{aligned}\frac{\partial T}{\partial \bar{n}}(x, a, t) &= 0 \\ \frac{\partial T}{\partial \bar{n}}(-a, y, t) &= 0 \\ \frac{\partial T}{\partial \bar{n}}(x, -a, t) &= 0\end{aligned}$$

fixed temperature (right)

$$T(a, y, t) = 0$$

3.2.4 3D heat transfer with source and PID controller [15, 16]

new problems:

- system of ODE and PDE
- `in` operator used to acces field value in a concrete point (PID controler equation defining T_s).
- vector field
- differential operators `grad` and `diverg`

l_x, l_y, l_z .. room dimensions ($6m$, $4m$, $3.2m$)
T .. temperature (scalar field)
W .. thermal flux (vector field)
c .. specific heat capacity ($1012\,J \cdot kg^{-1} \cdot K^{-1}$
ϱ .. density of air ($1.2041\,kg \cdot m^{-3}$)
λ .. thermal conductivity ($0.0257\,W \cdot m^{-1}K$)
T_{out} .. outside temperature (0 °C)
κ .. right wall heat transfer coefficient ($0.2\,W \cdot m^{-2} \cdot K^{-1}$
T_s .. temperature of the sensor placed in middle of the right wall
P .. power of heating
k_p, k_i, k_d .. coefficients of the PID controller (100, 200, 100)
T_d .. desired temperature (20°C)
e .. difference between temperature of the sensor and desired temperature

heat equation

$$\begin{aligned}\frac{1}{c\varrho}\nabla \cdot W &= -\frac{\partial T}{\partial t} \\ W &= -\lambda \nabla T\end{aligned}$$

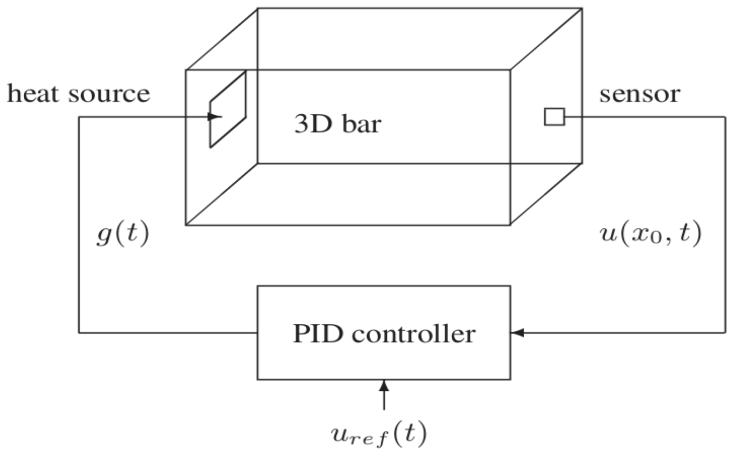

Figure 3.2: Heat transfer with source and PID controller

boundary conditions left wall ($x = 0$) - heat flux given by heating power

$$W_x = \frac{P}{l_y l_z}$$

rare ($y = 0$) and front ($y = l_y$), resp. bottom ($y = 0$) and top ($z = l_z$) insulated walls

$$W_y = 0, \text{ resp. } W = 0$$

right wall ($x = l_x$) - not fully insulated

$$W_x = \kappa(T - T_{out})$$

PID controler

$$\begin{aligned} T_s &= T(l_x, \frac{l_y}{2}, \frac{l_z}{2}) \\ e &= T_d - T_s \\ P &= k_p e + k_i \int_0^t e(\tau) d\tau + k_d \frac{d}{dt} e \end{aligned}$$

Modelica code:

```
model heatPID
  class Room
    extends DomainBlock3D;
    Region0D sensorPosition(shape = shapeFunc, range =
        {{1,1},{0.5,0.5},{0.5,0.5}})    ;
  end Room
```

```
  parameter Real lx = 4, ly = 5, lz = 3;
  Room room(Lx=lx, Ly=ly, Lz=lz);
  field Real T(domain = room, start = 15);
  field Real[3] W(domain = room);
  parameter Real c = 1012, rho = 1.204, lambda = 0.0257;
  parameter Real Tout = 0, kappa = 0.2;
  Real Ts, P, eInt;
  parameter Real kp = 100, ki = 200, kd = 100, Td = 20;
equation
  1/(c*rho)*diverg(W) = - pder(T,time) in room.interior;
  W = -lambda*grad(T)                  in room.interior;
  W*region.n = P/(lx*ly)               in room.left;
  W*region.n = 0                       in room.front,
      room.rare, room.top, room.bottom;
  W*region.n = kappa*(T - Tout)        in room.right;
  Ts = T                               in room.
      sensorPosition;
  e = Td - Ts;
  der(eInt) = e;
  P = kp*e + ki*eInt + kd*der(e);
end heatPid;
```

Listing 3.5: heat equation with PID controller

3.3 More complex realistic models

3.3.1 Henleho klička - protiproudová výměna

$c_{in}(x,t)$ .. koncentrace Na v sestupné části tubulu

$\bar{c}_{in}(x,t)$ .. koncentrace Na ve vzestupné části tubulu

$c_{out}(x,t)$ .. koncentraca Na v dření

$Q(x,t)$ .. tok vody v sestupné části tubulu

$f_{H_2O}(x,t)$ .. tok vody na milimetr délky z sestupné části tubulu do dřeně

f^*_{Na} .. tok sodíku ze vzestupné části tubulu do dřeně na milimetr délky – aktivní transport – parametr

L .. délka tubulu

P_{H_20} .. prostupnost cévy pro vodu (permeabilita)

$$
\begin{aligned}
\frac{\partial Q}{\partial x}(x,t) + f_{H_20}(x,t) &= 0 \\
(c_{out}(x,t) - c_{in}(x,t)) \cdot P_{H_2O} &= f_{H_20}(x,t) \\
f_{H_20}(x,t) &= \frac{dV}{dt}(t) \\
Q(L,t) \cdot c_{in}(L,t) &= f^*_{Na} \cdot L + Q(L,t) \cdot c^*(t) \\
\frac{\partial}{\partial x}\left(\bar{c}_{in}(x,t) \cdot Q(x,t)\right) &= f^*_{Na} \\
f^*_{Na} \cdot L &= \frac{dm_{Na}}{dt}
\end{aligned}
$$

3.3.2 Oxygen diffusion in tissue around vessel

polar coordinates (r, φ)

$$
\begin{aligned}
\frac{\partial \varrho}{\partial t} + q\left(\frac{1}{r}\frac{\partial \varrho}{\partial r} + \frac{\partial^2 \varrho}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 \varrho}{\partial \varphi^2}\right) + w &= 0 \\
\varrho(r_0, \varphi) &= \varrho_0 \\
\varrho(r, 0) &= \varrho(r, 2\pi) \\
\varrho_{nnn}(R, \varphi) &= 0 \; (= \varrho_{tn}(R, \varphi))
\end{aligned}
$$

ϱ .. oxygen concentration
ϱ_0 .. concentration in the vessel
q .. diffusion coefficient
w .. local oxygen consumption
R .. Ω diameter
The last equation should simulate infinite continuation of the domain.

3.3.3 Heat diffusion

domain boundary

$$
\partial \Omega \;=\; (a_b \cos(v),\, b_b \sin(v)),\; v \in \langle 0, 2\pi)
$$

equation [22]

$$
\frac{\partial T}{\partial t} + \frac{\lambda}{c\varrho}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) \;=\; W
$$

λ .. thermal conductivity
$W(x, y)$ .. heat power density of tissue (input)

$$
W(x,y) = \begin{cases} W_0 & \text{if } x^2 + y^2 \leq r^2 \\ 0 & \text{else} \end{cases}
$$

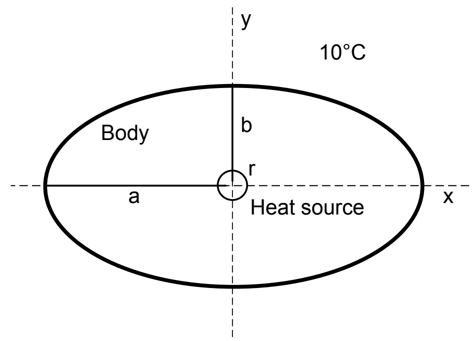

Figure 3.3: Scheme of heat diffusion in body

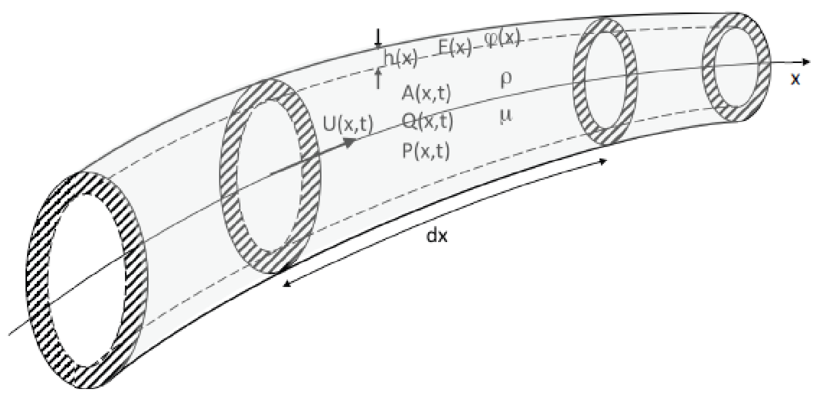

Figure 3.4: Arteria scheme

boundary condition

$$\lambda \frac{\partial T}{\partial n} = -\alpha(T - T_{out}),\ (x, y) \in \partial\Omega$$

α .. tissue-air thermal transfer coefficient [23]
initial condition

$$T(x, y, 0) = T_0(x, y)$$

3.3.4 Pulse waves in arteries caused by heart beats [2, 14, 17]

$A(x, t)$ .. crossection of vessel
$U(x, t)$ .. average velocity of blood
$Q(x, t)$ .. flux

$$Q = AU$$

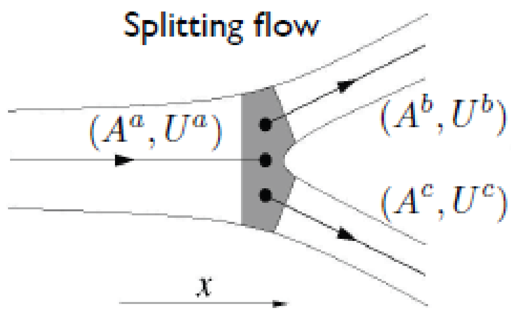

Figure 3.5: Arteria splitting

$P(x,t)$ .. pressure
P_{ext} .. external pressure
A_0 .. vessel crossection at ($P = P_{ext}$) (24mm)
$\beta = \frac{\sqrt{\pi} h_0 E}{(1-\nu^2) A_0}$
h_0 .. vessel wall thicknes (2mm)
E .. Young's modulus (0.24 - 6.55MPa)[7, 5, 4]
ν .. Poisson ratin (1/2)
$\varrho = 1050\,\mathrm{kg\,m^{-3}}$ .. blood density
$\mu = 4.0\,\mathrm{mPa\,s}$
α .. other ugly coefficient, let us say its 1
f .. frictional forces per unit length, let us assume inviscide flow $f = 0$, or $f = -AQ8\mu/(\pi r^4) = -8\pi\mu Q/A$[21]
μ.. dynamic viscosity of blood $(3\ 4) \cdot 10^{-3}$Pa·s[20]

$$
\begin{aligned}
\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} &= 0 \\
\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}\left(\alpha \frac{Q^2}{A}\right) + \frac{A}{\varrho}\frac{\partial P}{\partial x} &= \\
= \frac{\partial Q}{\partial t} + \alpha\left(2\frac{Q}{A}\frac{\partial Q}{\partial x} - \frac{Q^2}{A^2}\frac{\partial A}{\partial x}\right) + \frac{A}{\varrho}\frac{\partial P}{\partial x} &= \\
\frac{\partial Q}{\partial t} + 2\alpha\frac{Q}{A}\frac{\partial Q}{\partial x} + \left(\frac{\beta}{2\varrho}\sqrt{A} - \alpha\frac{Q^2}{A^2}\right)\frac{\partial A}{\partial x} &= \frac{f}{\varrho} \\
P_{ext} + \beta\left(\sqrt{A} - \sqrt{A_0}\right) &= P
\end{aligned}
$$

Three segment geometry – splitting arteria
We model arteria being splited into two minor arteries. Three same equation systems (super-indexes A, B, C) are solved on three different domains. Systems are connected via BC.

Boundary conditions

input

$$\begin{cases} P^A(0,t) = P_S & t \in \langle 0, \frac{1}{3}T_c) \\ Q^A(0,t) = 0 & t \in \langle \frac{1}{3}T_c, T_c) \end{cases}$$

T_c .. cardiac cycle period

junction

$$\begin{aligned} Q^A(L^A,t) &= Q^B(0,t) + Q^C(0,t) \\ P^A(L^A,t) &= P^B(0,t) \\ P^A(L^A,t) &= P^C(0,t) \end{aligned}$$

terminal we simulate the continuation of segments B and C with just a resitance

$$Q(L,t) = \frac{P(L,t)}{R_{out}}, \text{ for } B \text{ and } C$$

For check: the result should be in agreement with Moens–Korteweg equation.

3.3.5 Vocal cords

[6]

3.3.6 Vibrating membrane (drum) in air

Membrane [25]:

$\Omega_m = \{(x_m, y_m) | x_m^2 + y_m^2 < r^2\}$
$u(x,y,t)$ .. membrane displacement, $u : \Omega_m \times \langle 0, T\rangle \to \mathbb{R}$
r .. membrane radius
c_m .. membrane wave speed

$$\frac{\partial^2 u}{\partial t^2} = c_m^2 \nabla^2 u$$

Initial and boundary conditions

$$\begin{aligned} u(x,y,0) &= u_0(x,y) \\ u(x,y,t) &= 0 \ (x,y) \in \partial\Omega_m \end{aligned}$$

Air[18]:

$\Omega_a = \langle 0, l_x \rangle \times \langle 0, l_y \rangle \times \langle 0, l_z \rangle$
$\mathbf{v}(x, y, z, t)$ .. air speed, $\mathbf{v} : \Omega_a \times \langle 0, T \rangle \to \mathbb{R}^3$
$p(x, y, t)$ .. air pressure, $p : \Omega_a \times \langle 0, T \rangle \to \mathbb{R}$
ρ_0 .. density
c_a .. speed of sound

$$
\begin{aligned}
\rho_0 \frac{\partial \mathbf{v}}{\partial t} + \nabla p &= 0 \\
\frac{\partial p}{\partial t} + \rho_0 c_0^2 \nabla \cdot \mathbf{v} &= 0
\end{aligned}
$$

Initial and boundary conditions

$$
\begin{aligned}
\mathbf{v}(x, y, z, 0) &= \mathbf{0} \\
p(x, y, z, 0) &= p_0 \\
\mathbf{v}(x, y, z, t) \cdot \mathbf{n}(\partial \Omega_a) &= 0 \; (x, y, z) \in \partial \Omega
\end{aligned}
$$

Position of membrane in the room

Position of membrane centre $\mathbf{a} = (a_x, a_y, a_z)$ .. position vector of membrane centre in room

membrane lies in $\tilde{\Omega}_m = \left\{ (a_x + x,\, a_y + y,\, a_z) | x^2 + y^2 < r^2 \right\}$ in term of coordinates defined in Ω_a.

Or coordinate transformation (shift)

$$
\begin{aligned}
& x_m = x + a_x \\
& y_m = y + a_y \\
& a_z ? = ? z \qquad\qquad \text{(holds just in } \Omega_m\text{)}
\end{aligned}
$$

Equation connecting membrane and air

$$
\mathbf{v}(x, y, z, t) \cdot \mathbf{n}_{\tilde{\Omega}_m} = \frac{\partial u}{\partial t}(x - a_x, y - a_y, t) \text{ in } \tilde{\Omega}_m
$$

```
model membraneInAir
  import C = Modelica.Constants;

  //room deffinitions:
  parameter Real lx = 5, ly = 4, lz = 3;
  coordinate Real x, y, z;
  DomainBlock3D room(cartesian = {x,y,z}, Lx=lx, Ly=ly,
     Lz=lz);
```

```
  parameter Real p_0 = 101300;  //mean pressure

  field Real v[3](domain=room, start = zeros(3)); //air
     speed
  field Real p(domain=rooom, start = p_0);   //air
     pressure

  parameter Real rho_0 = 1.2;   //air density
  parameter Real c_a = 340;     //speed of sound in air

  //membrane deffinitions
  parameter Point membranePos(x=lx/2,y=ly/2,z=lz/2); //
     position of membrane center in the room
  r = 0.15; //membrane radius

  CircularDomian2D membrane1(x=x-membranePos.x, y=y-
     membranePos.y, 0=z-membranePos.z in interior, radius
      = r);

  parameter Real c_m = 100; //wave speed traversing the
     membrane

  function u0
    input x, y;
    output u0 = cos(sqrt(x^2 + y^2)*C.pi/(2*r));
  end u0;

  field Real u(domain = membrane, start[0] = u0, start[1]
      = 0);


equation
  //algernative aproach to match multiple domains(first
     -- equations in domain constructor):
  //is it OK that fields from different domain appeare
     here?
  membrane.x = room.x-membranePos.x in membrane.interior;
  membrane.y = room.y-membranePos.y in membrane.interior;
  0 = room.z-membranePos.z          in membrane.interior;

  //membrane equations:
  pder(u,t,t) = c_m^2*grad(diverg(u))  in membrane.
     interior;
  u = 0  in membrane.boundary;
```

```
  //room equations:
  rho_0*pder(v,t) + grad(p) = 0           in room.interior
      ;
  pder(p,t) + rho_0*c_0^2*diverg(v) = 0   in room.interior
      ;
  v*region.n = 0   in room.boundary;

  v*region.n = pder(u,t)   in membrane.interior;
end membraneInAir;
```

```
//Another aproach - class defining coordinates encloses
    domains :
class RoomAndMembrane
  ...
  parameter Real lx = 5, ly = 4, lz = 3;

  coordinates x, y, z;
  coordinates shiftCoord[3] = {x-membranePos.x,y-
      membranePos.y,z-membranePos.z};
  DomainBlock3D room(x=x, y=y, z=z, Lx=lx, Ly=ly, Lz=lz,
      ax = 0, ay = 0, az = 0);

//3 options to define membrane and and inner and outer
    coordinate transformation:

//1st:

//2nd rotated membrane

 //3th rotated, in matrix notation
```

```
  //air:
  field Real v[3](domain=room, start = zeros(3)); //speed
  field Real p(domain=rooom, start = p_0);    //pressure
  //membrane:
  field Real u(domain = membrane, start[0] = u0, start[1]
      = 0);   //displacement
equation
  ...
  v*region.n = pder(u,t)   in room.membrane; //relation
     between membrane and air fields
  ...
end RoomAndMembrane
```

Listing 3.6: Vibrating membrane in air

3.3.7 Euler equations

$$
\begin{aligned}
\frac{\partial \varrho}{\partial t} &= -\frac{\partial}{\partial x}(\varrho v) \\
\frac{\partial}{\partial t}(\varrho v) &= -\frac{\partial p}{\partial x} \\
\varrho \frac{\partial \varepsilon}{\partial t} &= -p \frac{\partial v}{\partial x}
\end{aligned}
$$

ϱ .. density, v .. velocity, p .. pressure, ε .. specific internal energy
state equation

$$p = \varepsilon \varrho (\gamma - 1)$$

$\gamma = c_p / c_v$ .. gas constant (fraction of specific heat capacities at constant pressure and volume)

Appendix A

Articles and books

I want to read: Other parts of Saldamli's thesis, e.g. first sections of chapter 7 and 9.3.

A DIFFERENTIATION INDEX FOR PARTIAL DIFFERENTIAL-ALGEBRAIC EQUATIONS [10]

INDEX AND CHARACTERISTIC ANALYSIS OF LINEAR PDAE SYSTEMS [11]

Finite difference methods for ordinary and partial differential equations [8]

A Framework for Describing and Solving PDE Models in Modelica [13]

Solving pde models in modelica.[9]

Solid modeling on Wikipedia. [24]

OO Modeling with PDE, Saldamli, Modelica work shop 2000

Appendix B

Questions & problems:

important topics are written in bold

B.1 Modelica language extension

- is it necessary to specifie the domain using “in” within equations, when it is actualy determined by the fields used in equations?

Coordinates

- Should be some coordinate system defined by default within the domain deffinition? (Perhaps cartesian by default and others defined extra by user if needed?)
  - I would say no. If yes, user should have option to give them a name, so that they are not always x, y, z.
- How to call atribute of Coordinate variable saying the type of the coordinate (now called `name`) should be the value assigned to this attribut written in quotes? It is also related with the previous question.
  e.g. somethink like `Coordinate x (name = “cartesian”);`
- Is needed `Coordinate` type?
  - Could be used just `Real` instead and compiler would infer that it is coordinate as it distinguishes e.g. state and algebraic variable now? How it may be infered? If domain is defined using coordinate equations – coordinate variables are either in region deffinitions (e.g. `Region1D interior(x in (a,b));`) or appear in equations with these variables.
  - or should it be `coordinate Real x;` or `coordinate x;`? Coordinate isn't actualy a data type, as it doesn't hold any data, it has no value. It is symbolic stuff.

- Should coordinates of one system be placed in an array so that they are ordered? Than individual elements could have alieses with the usual name. E.g.
  `cartesian[1] = x; cartesian[2] = y;`
- How to map shape function return values on particular space variables (e.g. `x, y, z`) when they are not ordered? And what if there are more coordinate systems defined (e.g. cartesian and polar)?
- Avoid equations of coordinate transformations in equation section and write somethink like
  Coordinate r (name = "polar", deffinition = sqrt(x^2 + y^2));
  ?

Other

- **How to define domain: using boundary description, shape-function or shape-equations?**
- Should be built-in `class Domain` empty, or contain perhaps `interior` and `boundary` regions?
  - perhaps it should contain `replaceable interior` of general type `Region`. Ut would be redeclared to `Region1D, Region2D,` or `Region3D` later.
- How to define general differential operators (as `grad, div` ...) , if we use user defined coordinates?
- **How to write equations (boundary conditions) that combine field variables from different domains?**
  - Using a region that is subset of both domains – how to write this?
  - Use just one domain, transform coordinates from the other domain. Example from 3.3.6
    `v(dom.x,dom.y,dom.z,t)*region.n = pder(u,t)(dom.x - a_x,dom.y - a_y,t) in room.membrane;`
    I dislike usage of arguments in equations.
- addition of regions (operator +)
  - the meaning is unintuitive, it is not clear thet regions are treated as sets
  - the resulting type is doubtable, should it be realy region as well? In 1D it is completly strange. In Rectangle2D e.g the `left` and `top` regions are defined using the same shape-functon, but shape-function of `left + right` is different, and complicated – requires conditions.

- perhaps instead of `Region1D reg3 = reg1 + reg2;`
  write
  `Region1D[] reg3 = {reg1, reg2};`

- Atribut `interval` in region constructor is assigned an interval value or a single constant. The letter is strange. Should be done in different way.

- Initialization.

- Rename *region* to *manifold*[1]?

- unify somehow concept of region and domain?

- How to call divergence operator (standard `div` is is already used for integer division)

- How should the shape, geometrical structure, mesh structure, etc. be described by an external file? Should be the file imported into the Modelica language, or just loaded by the runtime.

- Philosophical problem: What exists first, domain or coordinates? I would say coordinates must exist first as domain shape is defined using shape-function using some coordinates.

- is OK := op in fields?

- Allow higer derivatives? Perhaps allow only higher space derivatives, not time? Why are higher derivatives not allowed in current Modelica?

  - rather allow

- Allow some of this shortcuts to `pder(u,dom.x) = ...  in omega.interior`:
  `pder(u,dom.x) = ...  in omega //if no region specified, interior used implicitly`
  `pder(u,omega.x) = ...  //in omega ommited, information inferd from omega.x`

  - rather not

- Field variables and equations written within domains?

- Normal vector – should it be written rather in function-like way, `normal(omega.region1)` rather than `omega.region1.n`

  - perhaps "." notation is better as the normal vector is not a value but a function of coordinates

- field literal constructor:
`field Real f = field (2*x+y for (omega.x,omega.y));`
or
`field Real f = field (2*dom.x+dom.y in omega.interior );`
or
`field Real f = field (2* x+y for (x, y) in omega ) ;`
?

Solved problems:

- Multiple inheritance of domains – should it be allowd, what is the meaning?
  - multiple inheritance is allowed in general, but resulting equations must not be in conflict. Deffinition of regions using intervals is also som kind of equation. So we cannot inherit two domain calses that both defines e.g. Region interior.
- **How to deal with (name of) coordinate (independent)** variables, so that it doesn't meddle with other variables (ODE)?
  - coordinates are defined within the domain class. This solves the problem. Inside this class they may be addressed directly, outside `className.coordName` as other class members are accesed. In equation may be used shortcut keywords `domain` (or `dom`?) (and `region`) to address domain (and region) specified with `in` operator. `E.g.` `pder(u, domain.x)=0 in omega.left`
  - NO. avoid coordinate variables at all
    - allow writing equations coordinate-free, using only `pder(u,time)`, `grad, div,` ... operators (does it mean, we need no coordinates defined in domain?).
    - use operators `pderx(u)`, `pdert(u) or`
  - NO. Fixed names `x, y, z` used stand-alone. If they meddle with other variable, infere which one is it from tha fact that we differentiate with respect to this variable and from the actual domain (indicated with `in` op.). – Makes model confusing.
  - NO. fixed names and approach ODE variables from PDE in some special way.
  - NO. use longer name for coordinate variables (e.g. `spaceX` ...)
- **Allow writing equations independent on particular domain and also coordinate system?**

  - yes, using `replaceable` and `redeclare` on domain class and using coordinate free differential operators if we even don't know the dimension (`grad, div` etc.)

- Rename ranges to intervals?

  - yes

- Domain description where some parameters are in range and others are fixed: {{1,1}, {0.5,0.5}} or {{1,1}, 0.5}?

  - allow both

- How to deal with vector fields? How to acces its elements – using an index or scalar product with standard base vectors?

  - both

- How to distinguish the main domain (now called `DomainLineSegment1D`, `DomainRectangle2D` ...) and its "subsets" where some equations hold (now called `Domain0D`, `Domain1D` ...). I think only one of them should be called domain.

  - "subsets" renaimed to regions – `(Region0D, Region1D, Region 2D, Region3D)`

- directional derivative

  - `der(u,v)` (u is scalar or vector field in R^n, v is vector in R^n)

- Should it be possible to override initial and boundary conditions given in model with some different values from external configuration file?

  - yes

- How to set initial condition for field derivative in similar way as using `start` atribute (i.e. not using equation in `initial` section)? See 3.2.2

  - `start` atribut is array where index denotes the derivative `start[0]` - actual value, `start[1]` - first derivative

B.2 Generated code

- How to represent on which particular boundary an boundary condition hold in generated code (or even on which interior domain hold which PDE equation system, if we support various interiors)? – Some domain struct could hold both shapeFunction parameter ranges and pointer (or some index) to function with the corresponding equations. Or boundary condition function knows on which elements (indexes) of variable arrays should be applied.

- **Should be generated functions independent on grid? It means either**
  ```
  functionPDEIndependent(u,u_x,t,x)
  u_t = ...
  return u_t
  ```
  or
  ```
  functionPDEDependent(data)
  for (int i ...)
  u_t[i] = ...
  ```

B.3 Numerics and solver

- **Shall we support higher derivatives in solver?**

- **What about space derivatives? – All state and algebraics have corresponding array for its space derivative, not all of them are used. – Or all space derivatives of states and algebraics are stored as different algebraic fields. – Or there is array for space derivatives that is utilised by both states and algebraics that need it.**

- What about multi step mothods (RK, P-K)?

- How to generate even (or arbitrary) meshes with nonlinea shape functions?

- How to generate mesh points just on the boundary? 1D – simple – just two points. 2D – We can go through the boundary curve and detect crossings of grid lines. 3D – who knows?!

- How to plugin an already existing solver?

- How to determine causality of boundary conditions and other equations that hold on less dimensional manifolds.

- Build whole solver in some PDE framework, perhaps Overture (http://www.overtureframework.org/)

B.4 TODO

- Write a library for vector fields defining scalar and vector product, divergence, gradient, rotation...
- Write model in coordinates different from cartesian

Bibliography

[1] Krister Ahlander, Magne Haveraaen, and HansZ. Munthe-Kaas. On the role of mathematical abstractions for scientific computing. In RonaldF. Boisvert and PingTakPeter Tang, editors, *The Architecture of Scientific Software*, volume 60 of *IFIP – The International Federation for Information Processing*, pages 145–158. Springer US, 2001.

[2] Jordi Alastruey, Kim H Parker, and Spencer J Sherwin. Arterial pulse wave haemodynamics.

[3] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Press, 2004.

[4] Feng Gao, Masahiro Watanabe, Teruo Matsuzawa, et al. Stress analysis in a layered aortic arch model under pulsatile blood flow. *Biomed Eng Online*, 5(25):1–11, 2006.

[5] Raymond G Gosling and Marc M Budge. Terminology for describing the elastic behavior of arteries. *Hypertension*, 41(6):1180–1182, 2003.

[6] J. Horacek, P. Sidlof, and J.G. Svec. Numerical simulation of self-oscillations of human vocal folds with hertz model of impact forces. *Journal of Fluids and Structures*, 20(6):853 – 869, 2005. Axial-Flow Fluid-Structure Interactions Axial-Flow Fluid-Structure Interactions.

[7] Roberto M Lang, Bernard P Cholley, Claudia Korcarz, Richard H Marcus, and Sanjeev G Shroff. Measurement of regional elastic properties of the human aorta. a new application of transesophageal echocardiography with automated border detection and calibrated subclavian pulse tracings. *Circulation*, 90(4):1875–1882, 1994.

[8] Randall LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems.* Society for Industrial and Applied Mathematics, 2007.

[9] Zhihua Li, Ling Zheng, and Huili Zhang. Solving pde models in modelica. In *Proceedings of the 2008 International Symposium on Information Science and Engieering-Volume 01*, pages 53–57. IEEE Computer Society, 2008.

[10] Wade S Martinson and Paul I Barton. A differentiation index for partial differential-algebraic equations. *SIAM Journal on Scientific Computing*, 21(6):2295–2315, 2000.

[11] Wade S Martinson and Paul I Barton. Index and characteristic analysis of linear pdae systems. *SIAM Journal on Scientific Computing*, 24(3):905–923, 2003.

[12] Levon Saldamli. *A High-Level Language for Modeling with Partial Differential Equations*. PhD thesis, Department of Computer and Information Science, Linköping University, 2006.

[13] Levon Saldamli, Bernhard Bachmann, Hansjürg Wiesmann, and Peter Fritzson. A framework for describing and solving pde models in modelica. In *Paper presented at the 4th International Modelica Conference*, 2005.

[14] SJ Sherwin, V Franke, J Peiró, and K Parker. One-dimensional modelling of a vascular network in space-time variables. *Journal of Engineering Mathematics*, 47(3-4):217–250, 2003.

[15] Kristian Stavaker. Demonstration: Using hiflow3 together with modelica. Slides, March 26 2013.

[16] Kristian Stavåker, Staffan Ronnås, Martin Wlotzka, Vincent Heuveline, and Peter Fritzson. Pde modeling with modelica via fmi import of hiflow3 c++ components. Accepted.

[17] Inga Voges, Michael Jerosch-Herold, Jürgen Hedderich, Eileen Pardun, Christopher Hart, Dominik Daniel Gabbert, Jan Hinnerk Hansen, Colin Petko, Hans-Heiner Kramer, Carsten Rickers, et al. Normal values of aortic dimensions, distensibility, and pulse wave velocity in children and young adults: a cross-sectional study. *Journal of Cardiovascular Magnetic Resonance*, 14(1):77, 2012.

[18] Wikipedia. Acoustic theory. http://en.wikipedia.org/wiki/Acoustic_theory.

[19] Wikipedia. Advection equation. http://en.wikipedia.org/wiki/Advection.

[20] Wikipedia. Blood viscosity. http://en.wikipedia.org/wiki/Blood_viscosity.

[21] Wikipedia. Hagen–poiseuille equation. http://en.wikipedia.org/wiki/Hagen

[22] Wikipedia. Heat equation. http://en.wikipedia.org/wiki/Heat_equation.

[23] Wikipedia. PÅŹestup tepla. http://cs.wikipedia.org/wiki/PÅŹestup_tepla.

[24] Wikipedia. Solid modeling. http://en.wikipedia.org/wiki/Solid_modeling.

[25] Wikipedia. Vibrating membrane. http://en.wikipedia.org/wiki/Vibrations_of_a_circular_membrane.

[26] Wikipedia. Vibrating string. http://en.wikipedia.org/wiki/Vibrating_string.