Modeling with Partial Differential Equations – Modelica Language Extension Proposal

Jan Šilar\textsuperscript{1} \quad Kristian Stavåker\textsuperscript{2} \quad Marek Mateják\textsuperscript{1} \quad Pavol Privitzer\textsuperscript{1} \quad Jozef Nagy\textsuperscript{3}

\textsuperscript{1}First Faculty of Medicine \quad \textsuperscript{2}Department of Computer and Information Science
Charles University \quad Linköping University
Prague, Czech Republic \quad Linköping, Sweden

\textsuperscript{3}Prague, Czech Republic

OpenModelica Workshop 2014
## ODE and PDE

<table>
<thead>
<tr>
<th>examples</th>
<th>ODE</th>
<th>PDE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vibrating mass</td>
<td>vibrating string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unknown (solution)</th>
<th>function of 1 variable (here time)</th>
<th>multi-variable function (here time, space coordinate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>derivatives</td>
<td>the only variable $dt$</td>
<td>at least two variables $\partial t$, $\partial x$, ...</td>
</tr>
<tr>
<td>conditions</td>
<td>initial</td>
<td>initial and boundary</td>
</tr>
</tbody>
</table>

\[
\frac{d^2 x}{dt^2} + kx = 0
\]

\[
\frac{\partial^2 u}{\partial t^2} - q \frac{\partial^2 u}{\partial x^2} = 0
\]
PDE handling in the context of Modelica

- PDEs are not supported in the Modelica standard
- Approaches to unifying PDE modeling with Modelica exist
- Chapter in Peter Fritzson’s Modelica book [1]
- Levon Saldamli proposed in his PhD thesis [2]
  - support for PDE on language level
  - language extension not complete
  - not supported in OM and other tools

We continue this work and propose some modifications and further extensions.
Example – 3D heat transfer with source and PID controller

- heat transfer in a room
- in middle of right wall temperature sensor
- left wall – heating
- PID controller
Mathematical model of heat transfer

PDE:

\[ \frac{\partial T}{\partial t} - \alpha \cdot \left( \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) = 0 \]

IC:

\[ T(0, x, y, z) = a - kx \]

BC for left wall (heating)

\[ \frac{\partial T}{\partial \vec{n}}(t, 0, y, z) = -\beta \frac{P}{l_x l_y} \]

BC for other walls (insulated)

\[ \frac{\partial T}{\partial \vec{n}} = 0 \quad \text{on remaining walls} \]

\( T \) .. temperature – field
\( t, x, y, z \) .. time and space coordinates
\( l_x, l_y, l_z \) .. dimensions of the room
\( P \) .. power of heating
\( \vec{n} \) .. normal vector of the boundary
Mathematical model of PID controller and heating

heat sensor

\[ T_s = T \left( l_x, \frac{l_y}{2}, \frac{l_z}{2} \right) \]

PID controller and heating – ODE

\[ e = T_d - T_s \]

\[ P = k_p e + k_i \int_0^t e(\tau)d\tau + k_d \frac{d}{dt}e \]

\( T_s \) .. temperature of the sensor placed in middle of the right wall
model heatPID
  class Room
    extends DomainBlock3D;
    Region0D sensorPos(shape=shapeFunc, range=\{{1,1},
                      \{0.5,0.5\},\{0.5,0.5\}\});
  end Room
  parameter Real lx = 4, ly = 6, lz = 3;
  Room room(Lx=lx, Ly=ly, Lz=lz);
  parameter Real T_0 = \{a-k*room.x in room.interior\};
  field Real T(domain = room, start = T_0);
  Real Ts, P, eInt;
  parameter Real kp, ki, kd, Td, a, k, alpha, beta;

  equation
    pder(T,time) - alpha*(pder(T,x,x) + pder(T,y,y))
    + pder(T,z,z)) = 0 in room.interior;
    pder(T,region.n) = -beta*P/(lx*ly) in room.left;
    pder(T,region.n) = 0 in room.right room.front,
    room.rare, room.top, room.bottom;
    Ts = T in room.sensorPos;
    e = Td - Ts;
    der(eInt) = e;
    P = kp*e + ki*eInt + kd*der(e);
  end heatPID;
Domain

- We have to represent the room and its boundaries.
- Subsets of space (1D, 2D or 3D) where fields are defined, equations hold and calculations are performed we call domain.
- There is new built-in type Domain:

```plaintext
type Domain
    parameter Integer ndim;
    replaceable Region interior;
    replaceable function shape
        input Real u[ndim-1];
        output Real coord[ndim];
    end shape;
end Domain;
```

- All domains extend this built-in type
- Domain contains regions representing its interior and boundaries. `region` is also new built-in type.
- Shape-function maps real intervals onto regions and thus defines them (idea from Peter's Book [1]).
class DomainBlock3D
extends Domain(ndim=3);
parameter Real Lx, Ly, Lz, ax, ay, az;
Coordinate x, y, z;
coord = {x,y,z};
redeclare function shape //impure - not supported
   input Real vx, vy, vz;
   output Real x=ax+Lx*vx, y=ay+Ly*vy, z=az+Lz*vz;
end shape;
Region3D interior(shape=shape, interval=\{\{0,1\},\{0,1\},\{0,1\}\});
Region2D left(shape=shape, interval=\{0,\{0,1\},\{0,1\}\});
Region2D bottom(shape=shape, interval=\{\{0,1\},\{0,1\},0\});
...  //right, top, front, rear
end DomainBlock3D;
other options to define domains exist
Coordinates

We used coordinate variables to determine position of points in the room.

Coordinates

- are independent variables (such as time in current Modelica)
- fields are functions of coordinates
- fields may be differentiated with respect to coordinates
- need spacial handling in compiler and runtime

New modifier coordinate – to define coordinates. Usage e.g.

```modelica
coordinate Real x(name='''cartesian''');
```

The type should be always Real.
Temperature in the room is a field variable

- its value depends on time and space position (multi-variable function)

new modifier field to define fields, usage e.g.

```plaintext
field Real T(domain=room);
```

- modelica built-in types or types derived from them allowed only
- `domain` – mandatory attribute – to determine definition domain
Field – literal constructor, accessing value

- to set initial value of the temperature, field literal constructor is needed

```plaintext
field parameter Real T_0 = {a-k*room.x in room};
```

- $T_0$ is a field constant in time
- its domain is room
- its value is given by the expression depending on the $x$ coordinate

- We have to access value of temperature in the point where the temperature sensor is placed.

```plaintext
T_sensor = T in room.sensorPosition;
```

- $T$ is field, $T_{\text{sensor}}$ not
- sensorPosition is of type Region0D – represents the point
in – domain and region specification keyword

- in keyword is used also in PDEs and BC to determine on which region (boundary) they hold.
- default region is interior, if it is not specified

field Real T, W;
Real T_sensor;
parameter Real lambda, T_out;
Room room; //3D domain, has 0D region sensorPosition

 equation
  W = -lambda*grad(T) in room; //= room.interior (PDE)
  pder(T,room.right.n) = 0 in room.right; //BC
  T_sensor = T in room.sensorPosition;
  //(nonfield-field)
We need partial differential operators that are used in PDEs and BCs.

- **New pder operator:**
  
  \[
  \text{pder}(u, \text{time}) \sim \frac{\partial u}{\partial t}, \quad \text{pder}(u, \text{room.x}) \sim \frac{\partial u}{\partial x}, \\
  \text{pder}(u, \text{room.x}, \text{room.y}) \sim \frac{\partial^2 u}{\partial x \partial y}
  \]

- **normal derivative (often in BC)**

  \[
  \text{pder}(u, \text{room.n}) \sim \frac{\partial u}{\partial n}
  \]

- \(n\) (normal vector) – implicit member of all regions of dimension \(n - 1\) in \(n\)-dimensional domain

- **vector differential operators** gradient, divergence, curl are defined
Accessing coordinate variables

- Fields and equations are located outside domain class. Coordinates (and normal vector) inside – in equations must be accessed using "."

```plaintext
pder(u,omega.x) = 0 in omega.interior;
v*omega.left.n = 0 in omega.left;
```

- One possible shortcut would be to use new keywords `dom` and `reg` to refer the domain, resp. region specified with `in`. E.g.

```plaintext
pder(u,dom.x) = 0 in omega.interior;
v*reg.n = 0 in omega.left;
```

- Other option – allow access coordinates directly without specification of the domain:

```plaintext
pder(u,x) = 0 in omega.interior;
v*n = 0 in omega.left;
```

- If a variable of a same name exists, it has preference and the coordinate must be accessed with "." anyway
Changes over previous extensions – domains

• Domains
  • previously described by defining its boundaries using shape-function. It works well in 2D, but doesn’t work in 3D. E.g. to build-up the domain on the picture below, it is hard to find shape-functions generating incomplete intersecting square sides and sphere.

  ![Diagram](image)

• we suggest to use one shape-function to parametrize interior and all boundaries. Complex geometries define in some other way (CSG, import from CAD).
Changes – coordinates, in operator, accessing fields

- Coordinates
  - previously no special syntax to define them. There are two predefined arrays for coordinate systems in built-in type domain.
  - we suggest new modifier coordinate to define coordinates so that user can define any number of coordinate systems.

- in operator
  - previously used only in BCs and filed literal constructor
  - we suggest to use it also in PDEs and equations relating field and non-field variables (explained in next point)

- accessing field values in particular point
  - originally in function-like style, e.g. $T(1.3, 9.1, 4.7)$
  - we suggest to disable this, as it is also not possible to access regular variable in particular time in this way. Use in operator with region of type Region0D to represent the point instead.
Changes – field literal constructor, start values of derivatives

• field literal constructor
  - originally e.g. \( u = \{2a+b \text{ for } (a,b) \in \omega \} \), where iterator variables \((a,b)\) exist only in constructor expression and represent coordinates in \( \omega \). But which coordinate do iterator variables represent if there are more coordinate systems in \( \omega \)?
  - we suggest \( u = \{2\omega.x+\omega.y \text{ in } \omega\text{.interior} \} \) instead, where \( \omega.x \) and \( \omega.y \) are already defined in \( \omega \).

• start value for derivatives of a field
  - higher derivatives of fields are allowed, thus we need to assign initial values to derivatives. We suggest new attributes of a field variables \( \text{startPrime} \) and \( \text{startSecond} \) to set start value for first and second derivative. Prospective even higher derivatives must be initialized in initialEquation section.
Changes – differential operators, normal vector, accessing coordinate variables

- differential operators
  - previously $\text{der}(u) \sim \frac{\partial u}{\partial t}$, $\text{der}(u,x) \sim \frac{\partial u}{\partial x}$
  - we suggest $\text{pder}(u, \text{time}) \sim \frac{\partial u}{\partial t}$, $\text{pder}(u,x) \sim \frac{\partial u}{\partial x}$

- normal vector
  - previously member of domain
  - now member of regions of dimension n-1 in n-dimensional domain i.e. plane in 3D, curve in 2D and points in 1D.

- accessing coordinate variables
  - two possible shortcuts to access coordinates in equations
Conclusions and future goals

- we studied previous extensions
- proposed few changes to eliminate some weaknesses
- added some new extensions

Presented extension – merge of previous and new extensions.

What next?

- complete the extension (still several open problems)
- implement support (by compiler, runtime, solver) in OpenModelica (during my PhD implement at least 1D models)
- attempt to standardize it
New ideas welcome.

Thank you.
Bibliography

Peter Fritzson.  
*Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*.  

Levon Saldamli.  
*A High-Level Language for Modeling with Partial Differential Equations*.  