Introduction to MODELICA

P. Moreaux
LISTIC – Polytech Annecy-Chambéry
Université de Savoie – France
April 10th, 2012
Introduction to MODELICA

P. Moreaux
LISTIC – Polytech Annecy-Chambéry
Université de Savoie – France
April 10th, 2012

SIMUREX 2012
Cargèse, France - April 10-14, 2012
Overview

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
Overview

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
More information…

- Modelica Association: www.modelica.org
- OpenModelica: www.openmodelica.org
- “The” book:
Credits for this tutorial – Thanks to ...

- A lot of very good, detailed, topic specific tutorials are available, see: [http://modelica.org/](http://modelica.org/)

- Among them:
  - Peter Fritzson, *Introduction to Object-Oriented Modeling and Simulation with OpenModelica*, (2006)
  - Mohsen Torabzadeh-Tari (Linköping University), *Introduction to Object-Oriented Modeling, Simulation and Control with Modelica*, (May 2011).
  - Martin Otter (DLR-RM and Chairman of Modelica Association), *Modelica Overview*, 2009
  - Sébastien FURIC, R&D Engineer, LMS (INSA Lyon, 2007)
Modelica - status

- Free, open language
- Specification available at: www.modelica.org
- Developed since 1997 (1.0), V 3.2 released March 2010
- Modelica Association established 2000 in Linköping
  Open, non-profit organization
  Industry and Academia
Overview

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
Modelica – main goals

- Modeling the dynamic behavior of technical systems consisting of components from, e.g., mechanical, electrical, thermal, hydraulic, pneumatic, fluid, control and other domains in a convenient way.

- Models are described by differential, algebraic, and discrete equations.

- No description by partial differential equations, i.e., no FEM (finite element method) and no CFD (computational fluid dynamics), but using results of, e.g., FEM programs.
Modelica – main features (1)

- **Declarative language (acausal)**
  - Equations and mathematical functions allow acausal modelling (high level specification, increased correctness)

- A single modelling framework

- **Multi-domain modeling**
  - electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc... domains

- Hybrid modeling: continuous-time + discrete-time

- Modular programming! Components; connections
  - Hierarchical system architecture capabilities
Modelica – main features (2)

- Several basic constructs (model, block, record, …) but technically,

- Everything is a class (warning: not a OO language!)
  - Strongly typed object-oriented language with a general class concept, Java & MATLAB-like syntax

- Efficient, non-proprietary
  - Efficiency comparable to C; advanced equation compilation,
  - e.g. 300 000 equations, ~150 000 lines on standard PC

- Several environments for Visual component programming
Acausal versus causal modeling

- Acausal model (Modelica)

- Causal block-based model (Simulink)
Modelica vs Simulink Block Oriented Modeling

- Modelica: Physical system
- Simulink: Signal-flow model
Equations versus assignment

Equation clauses are not sequences of statements! (in particular, there is no notion of assignment, nor evaluation order)

It is however possible to describe how to compute a result by means of sequences of assignments, loops, etc. in Modelica, but not using equations!
The semantics of the Modelica language is specified by means of a set of rules for translating any class described in the Modelica language to a flat Modelica structure. A class must have additional properties in order that its flat Modelica structure can be further transformed into a set of differential, algebraic and discrete equations (= flat hybrid DAE). Such classes are called simulation models.
Overview

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
Assembling models

To get a model of a system:
- Pick Basic (or complex!) (sub)models
- Connect these models

The DAE is derived from the sub-models and from the equations defined by the connections
Several elementary constructs

- Model: elementary model or assembly of such models
- Block: construct with defined causality (inputs / outputs)
- Record: data only
- Connector: to define interactions between models
- Function: to define specific computations, behaviors
A DC motor model

A DC motor can be thought of as an electrical circuit which also contains an electromechanical component (Mohsen Torabzadeh-Tari)

```model DCMotor
    Resistor R(R=100);
    Inductor L(L=100);
    VsourceDC DC(f=10);
    Ground G;
    ElectroMechanicalElement EM(k=10, J=10, b=2);
    Inertia load;

    equation
        connect(DC.p,R.n);
        connect(R.p,L.n);
        connect(L.p, EM.n);
        connect(EM.p, DC.n);
        connect(DC.n,G.p);
        connect(EM.flange,load.flange);
end DCMotor
```
Derived system of equations

<table>
<thead>
<tr>
<th>Equation 1</th>
<th>Equation 2</th>
<th>Equation 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 == DC.p.i + R.n.i</td>
<td>EM.u == EM.p.v - EM.n.v</td>
<td>R.u == R.p.v - R.n.v</td>
</tr>
<tr>
<td>DC.p.v == R.n.v</td>
<td>0 == EM.p.i + EM.n.i</td>
<td>0 == R.p.i + R.n.i</td>
</tr>
<tr>
<td>0 == R.p.i + L.n.i</td>
<td>EM.i == EM.p.i</td>
<td>R.i == R.p.i</td>
</tr>
<tr>
<td>R.p.v == L.n.v</td>
<td>EM.i == EM.M / EM.k</td>
<td>R.u == R.R * R.i</td>
</tr>
<tr>
<td>0 == L.p.i + EM.n.i</td>
<td>EM.J * EM.ω == EM.M - EM.b * EM.ω</td>
<td>L.u == L.p.v - L.n.v</td>
</tr>
<tr>
<td>L.p.v == EM.n.v</td>
<td>DC.u == DC.p.v - DC.n.v</td>
<td>0 == L.p.i + L.n.i</td>
</tr>
<tr>
<td>0 == EM.p.i + DC.n.i</td>
<td>0 == DC.p.i + DC.n.i</td>
<td>L.i == L.p.i</td>
</tr>
<tr>
<td>EM.p.v == DC.n.v</td>
<td>DC.i == DC.p.i</td>
<td>L.u == L.L * L.i'</td>
</tr>
<tr>
<td>0 == DC.n.i + G.p.i</td>
<td>DC.u == DC.Amp * Sin[2 π DC.f * t]</td>
<td>(load component not included)</td>
</tr>
<tr>
<td>DC.n.v == G.p.v</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Connectors

- Connect(A.x, B.y):
  - Acausal coupling: \( x = y \)
  - Causal coupling: \( x \) and \( y \) are declared flow: \( A.x + B.y = 0 \)
Connectors are specific classes with two kinds of variables:

- Standard: potential or energy level => Equality
- Flow: some kind of flow => Sum to zero

**Domain**
- Hydraulic Pressure
- Heat Temp.

**Potential**
- Volume flow
- Heat Flow

**Flow**
- Volume
- Heat

**Carrier**
- HyLibLight
- HeatFlowID

**Modelica Lib.**
Overview

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
Algorithms

- Sequence of statements

- Classical programming constructs:
  - Assignments
  - If – then - elseif – else
  - Loops : for, while

- Event driven statements (see “events”):
  - When
Event based modeling

- Events are distinguished time instants

- Time instants defined through:
  - Explicit time
  - Change of variable value: False to True or change of boolean: \((x \geq 2.4)\) (edge, change)

- Initial and terminal events

- Repeated events (sample)

- Event based behavior:
  
  \[
  \text{when cond then equation end when}
  \]
Functions (1)

Definition of a function:
- Eventually, some structural parameters used to denote array dimensions
- Formal parameters
- Eventually, some internal variables
- Executable statements (or call to foreign function, written in C for instance)
- Executable statements include assignments, "while" loops, "for" loops and return

```
function myFunction
    input Real u1, u2;
    output Real y1, y2;
    protected
        Real x1, x2;
    algorithm
        x1 := u1 + u2;
        x2 := u1 - u2;
        y1 := x1 * x2;
        y2 := x1 / x2;
end myFunction;
```
Functions (2)

- Declarative Mathematical semantics: always returns the same result given the same input argument values
- Positional and named arguments
- Multiple results
- External functions (C or FORTRAN code)
Overview

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
Managing complex systems modeling

- Models: Composition and Hierarchy
- Packages: set of Modelica entities
- Libraries: structured sets of packages
- REUSE!
Packages

A package is
- A name space (model definition, programming)
- A hierarchical set of Modelica classes and constant components

Packages may be stored:
- As nested Modelica classes, in a single file
- In the host file system, as a tree of directories and files
Libraries

- A library is a hierarchy of packages, structured by the application domain or by the provided programming or technical functionalities.

- A library usually provides several subpackages containing:
  - The public types used in the library
  - Eventually, some useful functions
  - The connectors used to build classes
  - Interfaces of classes
  - Instantiable classes
  - Some test models
And now ..

- Modelica – where to get Information
- Main features
- Building Modelica models
- Some programming details
- Structuring – reuse
- Run the simulation !!!
Modelica - Modeling and simulation

Global workflow for modeling and simulation
Free and commercial Modelica tools

- Usually provide Modeling AND simulations facilities
- Should provide tool-independent models
- Open source:
  - OpenModelica from OSMC
- Commercial
  - MathModelica by MathCore
  - Dymola by Dassault systems / Dynasim
  - SimulationX by ITI
  - MapleSim by MapleSoft
- ...
Thank you

Enjoy modeling/simulation with Modelica!

Questions?