The Modelica Language and Technology for Model-Based Development

Overview Talk

March 28, 2012

Peter Fritzson

Professor at Linköping University, Sweden
Vice Chairman of Modelica Association
Director of Open Source Modelica Consortium

peter.fritzson@liu.se
Part I

Introduction to Modelica
Modelica Background: Stored Knowledge

Model knowledge is stored in books and human minds which computers cannot access

“The change of motion is proportional to the motive force impressed”
– Newton
Modelica Background: The Form – Equations

- Equations were used in the third millennium B.C.
- Equality sign was introduced by Robert Recorde in 1557

\[
14.3e -1 - 15.9 = -71.9
\]

Newton still wrote text (Principia, vol. 1, 1686)
“*The change of motion is proportional to the motive force impressed*”

CSSL (1967) introduced a special form of “equation”:
\[
\text{variable} = \text{expression} \\
v = \text{INTEG}(F)/m
\]

**Programming languages usually do not allow equations!**
What is Modelica?

A language for modeling of complex cyber-physical systems

- Robotics
- Control
- Automotive
- Aircraft
- Satellites
- Power plants
- Systems biology
What is Modelica?

A language for **modeling** of complex cyber-physical systems

Primary designed for **simulation**, but there are also other usages of models, e.g. optimization.
What is Modelica?

A language for modeling of complex cyber physical systems
i.e., Modelica is not a tool

Free, open language specification:

There exist several free and commercial tools, for example:

- OpenModelica from OSMC
- MathModelica from MathCore
- Dymola from Dassault systems
- SimulationX from ITI
- MapleSim from MapleSoft

Available at: www.modelica.org
Modelica – The Next Generation Modeling Language

Declarative language
Equations and mathematical functions allow acausal modeling, high level specification, increased correctness

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

Everything is a class
Strongly typed object-oriented language with a general class concept, Java & MATLAB-like syntax

Visual component programming
Hierarchical system architecture capabilities

Efficient, non-proprietary
Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations, ~150 000 lines on standard PC
What is *acausal* modeling/design?

Why does it increase *reuse*?

The acausality makes Modelica library classes *more reusable* than traditional classes containing assignment statements where the input-output causality is fixed.

Example: a resistor *equation*:

\[
R \cdot i = v;
\]

can be used in three ways:

\[
i := v/R;
\]

\[
v := R \cdot i;
\]

\[
R := v/i;
\]
What is Special about Modelica?

- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation
What is Special about Modelica?

Multi-Domain Modeling

Cyber-Physical Modeling

Physical

3 domains
- electric
- mechanics
- control

Cyber

Control System

Reference

Electric

Mechanics

Bearing

Angle-Sensor

PID
What is Special about Modelica?

Multi-Domain Modeling

Keeps the physical structure

Acausal model (Modelica)

Causal block-based model (Simulink)

Visual Acausal Hierarchical Component Modeling
What is Special about Modelica?

- Multi-Domain Modeling
- Hierarchical system modeling
- Visual Acausal Hierarchical Component Modeling

<Diagram>

\[
S_{rel} = n^\top \text{transpose}(n) + \{\text{identity}(3) - n^\top \text{transpose}(n)\} \cdot \cos(q) - \text{skew}(n) \cdot \sin(q);
\]
\[
w_{rela} = n^\top q_d;
\]
\[
z_{rela} = n^\top q_{dd};
\]
\[
S_{b} = Sa^\top \text{transpose}(S_{rel});
\]
\[
r_{0b} = r_{0a};
\]
\[
v_{b} = S_{rel}^\top v_{a};
\]
\[
w_{b} = S_{rel}^\top (w_{a} + w_{rela});
\]
\[
ab = S_{rel}^\top aa;
\]
\[
z_{b} = S_{rel}^\top (za + z_{rela} + \text{cross}(wa, w_{rela}));
\]
What is Special about Modelica?

A textual class-based language
Object-Orientation mainly used as structuring concept

Behaviour described declaratively using
- Differential algebraic equations (DAE) (continuous-time)
- Event triggers (discrete-time)

```
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1) "Descriptive string for x";
  Real y(start = 1) "y coordinate";
  parameter Real lambda = 0.3;
  equation
    der(x) = y;
    der(y) = -x + lambda*(1 - x*x)*y;
  end VanDerPol;
```
What is Special about Modelica?

- Visual Acausal Component Modeling
- Multi-Domain Modeling
- Hybrid modeling = continuous-time + discrete-time modeling
- Typed Declarative Equation-based Textual Language

Hybrid modeling =
continuous-time + discrete-time modeling
Graphical Modeling - Using Drag and Drop Composition
A DC motor can be thought of as an electrical circuit which also contains an electromechanical component.

```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```

Multi-Domain (Electro-Mechanical) Modelica Model
## Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 = \text{DC.p.i} + \text{R.n.i} )</td>
<td>( \text{EM.u} = \text{EM.p.v} - \text{EM.n.v} )</td>
</tr>
<tr>
<td>( \text{DC.p.v} = \text{R.n.v} )</td>
<td>( 0 = \text{EM.p.i} + \text{EM.n.i} )</td>
</tr>
<tr>
<td>( \text{EM.i} = \text{EM.p.i} )</td>
<td>( \text{R.i} = \text{R.p.i} )</td>
</tr>
<tr>
<td>( 0 = \text{R.p.i} + \text{L.n.i} )</td>
<td>( \text{EM.u} = \text{EM.k*EM.}\omega )</td>
</tr>
<tr>
<td>( \text{R.p.v} = \text{L.n.v} )</td>
<td>( \text{EM.i} = \text{EM.M}/\text{EM.k} )</td>
</tr>
<tr>
<td>( \text{EM.J<em>EM.}\omega = \text{EM.M-EM.b</em>EM.}\omega )</td>
<td>( \text{L.u} = \text{L.p.v} - \text{L.n.v} )</td>
</tr>
<tr>
<td>( 0 = \text{L.p.i} + \text{EM.n.i} )</td>
<td>( \text{DC.u} = \text{DC.p.v} - \text{DC.n.v} )</td>
</tr>
<tr>
<td>( \text{L.p.v} = \text{EM.n.v} )</td>
<td>( 0 = \text{DC.p.i} + \text{DC.n.i} )</td>
</tr>
<tr>
<td>( \text{EM.p.v} = \text{DC.n.v} )</td>
<td>( \text{DC.i} = \text{DC.p.i} )</td>
</tr>
<tr>
<td>( 0 = \text{DC.n.i} + \text{G.p.i} )</td>
<td>( \text{DC.u} = \text{DC.Amp<em>Sin}[2\pi\text{DC.f</em>t}] )</td>
</tr>
<tr>
<td>( \text{DC.n.v} = \text{G.p.v} )</td>
<td>( \text{(load component not included)} )</td>
</tr>
</tbody>
</table>

Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f[x, u, t] \\
g\left[\frac{dx}{dt}, x, u, t\right] = 0
\]
Model Translation Process to Hybrid DAE to Code

Modeling Environment

Modelica Graphical Editor → Modelica Model
Modelica Textual Editor → Modelica Model

Frontend

Translator → Modelica Source code

Analyzer → Flat model Hybrid DAE

Optimizer → Sorted equations

Code generator → Optimized sorted equations

C Compiler → C Code

Executable

Backend

"Middle-end"
Modelica in Power Generation

GTX Gas Turbine Power Cutoff Mechanism

Developed by MathCore for Siemens

Courtesy of Siemens Industrial Turbomachinery AB, Finspång, Sweden
Modelica in Avionics
Modelica in Automotive Industry
Modelica in Biomechanics
Application of Modelica in Robotics Models
Real-time Training Simulator for Flight, Driving

- Using Modelica models generating real-time code
- Different simulation environments (e.g. Flight, Car Driving, Helicopter)
- Developed at DLR Munich, Germany
- Dymola Modelica tool

Courtesy of Martin Otter, DLR, Oberpfaffenhofen, Germany
Combined-Cycle Power Plant
Plant model – system level

- GT unit, ST unit, Drum boilers unit and HRSG units, connected by thermo-fluid ports and by signal buses.

- Low-temperature parts (condenser, feedwater system, LP circuits) are represented by trivial boundary conditions.

- GT model: simple law relating the electrical load request with the exhaust gas temperature and flow rate.

Courtesy Francesco Casella, Politecnico di Milano – Italy and Francesco Pretolani, CESI SpA - Italy
Formation flying on elliptical orbits

Control the relative motion of two or more spacecraft

Attitude control for satellites using magnetic coils as actuators

Torque generation mechanism: interaction between coils and geomagnetic field

Courtesy of Francesco Casella, Politecnico di Milano, Italy
The Modelica Standard Library contains components from various application areas, including the following sublibraries:

- **Blocks**  
  Library for basic input/output control blocks
- **Constants**  
  Mathematical constants and constants of nature
- **Electrical**  
  Library for electrical models
- **Icons**  
  Icon definitions
- **Fluid**  
  1-dim Flow in networks of vessels, pipes, fluid machines, valves, etc.
- **Math**  
  Mathematical functions
- **Magnetic**  
  Magnetic.Fluxtubes – for magnetic applications
- **Mechanics**  
  Library for mechanical systems
- **Media**  
  Media models for liquids and gases
- **Slunits**  
  Type definitions based on SI units according to ISO 31-1992
- **Stategraph**  
  Hierarchical state machines (analogous to Statecharts)
- **Thermal**  
  Components for thermal systems
- **Utilities**  
  Utility functions especially for scripting
Brief Modelica History

- First Modelica design group meeting in fall 1996
  - International group of people with expert knowledge in both language design and physical modeling
  - Industry and academia

- Modelica Versions
  - 1.0 released September 1997
  - 2.0 released March 2002
  - 2.2 released March 2005
  - 3.0 released September 2007
  - 3.1 released May 2009
  - 3.2 released May 2010
  - 3.3 planned May 2012

- Modelica Association established 2000
  - Open, non-profit organization
Modelica Conferences

- The 1st International Modelica conference October, 2000
- The 2nd International Modelica conference March 18-19, 2002
- The 3rd International Modelica conference November 5-6, 2003 in Linköping, Sweden
- The 4th International Modelica conference March 6-7, 2005 in Hamburg, Germany
- The 5th International Modelica conference September 4-5, 2006 in Vienna, Austria
- The 6th International Modelica conference March 3-4, 2008 in Bielefeld, Germany
- The 7th International Modelica conference Sept 21-22, 2009, Como, Italy
- The 8th International Modelica conference March 20-22, 2011 in Dresden, Germany
- Coming: the 9th International Modelica conference Sept 3-5, 2012 in Munich, Germany
Part II

Modelica Environments and OpenModelica
Wolfram System Designer – Wolfram Research

- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- www.wolfram.com
- www.mathcore.com

Car model graphical view

Mathematica

Simulation and analysis
MWORKS – Suzhou Tongyuan Software & Control

MWORKS Studio

- Tongyuan Software & Control
- China
- Recent Modelica tool on the market
- CAD interface
MapleSim – MapleSoft

- Maplesoft
- Canada
- Recent Modelica tool on the market
- Integration with Maple
- www.maplesoft.com

Courtesy of MapleSoft
Dymola – Dassault Systémès

- Dassault Systémès
- France, Sweden
- First Modelica tool on the market
- Main focus on automotive industry
- www.3ds.com
Simulation X – ITI

- ITI
- Germany
- Mechatronic systems
- www.simulationx.com

Courtesy of ITI
The OpenModelica Environment

www.OpenModelica.org
The OpenModelica Open Source Environment
www.openmodelica.org

- Advanced Interactive Modelica compiler (OMC)
  - Supports most of the Modelica Language
  - Modelica and Python scripting
- Basic environment for creating models
  - OMSHELL – an interactive command handler
  - OMNOTEBOOK – a literate programming notebook
  - MDT – an advanced textual environment in Eclipse
- OMEdit graphic Editor
- OMOptim optimization tool
- ModelicaML UML Profile
- MetaModelica extension
- ParModelica extension

The OpenModelica Open Source Environment
www.openmodelica.org
OSMC – International Consortium for Open Source Model-based Development Tools, 40 members Feb 2012

Founded Dec 4, 2007

Open-source community services

• Website and Support Forum
• Version-controlled source base
• Bug database
• Development courses
• www.openmodelica.org

Industrial members (22)

• ABB Corp Research, Sweden
• Bosch Rexroth AG, Germany
• Siemens PLM, California, USA
• Siemens Turbo, Sweden
• CDAC Centre, Kerala, India
• Creative Connections, Prague
• DHI, Aarhus, Denmark
• Evonik, Dehli, India
• Equa Simulation AB, Sweden
• Fraunhofer FIRST, Berlin
• Frontway AB, Sweden
• IFP, Paris, France

• InterCAX, Atlanta, USA
• ISID Dentsu, Tokyo, Japan
• MathCore Engineering/Wolfram, Sweden
• Maplesoft, Canada
• TLK Thermo, Germany
• Sozhou Tongyuan, China
• VI-grade, Italy
• VTI, Linköping, Sweden
• VTT, Finland
• XRG Simulation, Germany

Code Statistics

[Graph showing Lines of Code]

University members (18)

• Linköping University, Sweden
• TU Berlin, Institute of UEBB, Germany
• FH Bielefeld, Bielefeld, Germany
• TU Braunschweig, Institute of Thermodynamics, Germany
• TU Dortmund, Proc. Dynamics, Germany
• Technical University Dresden, Germany
• Université Laval, modelEAU, Canada
• Georgia Institute of Technology, USA
• Ghent University, Belgium
• Griffith University, Australia
• Hamburg Univ. Technology/TuTech, Institute of Thermo-Fluid, Germany
• University of Ljubljana, Slovenia
• University of Maryland, Inst. Systems Engineering, USA
• University of Maryland, CEEE, USA
• Politecnico di Milano, Italy

• Ecoles des Mines, ParisTech, CEP, France
• Mälardalen University, Sweden
• Telemark University College, Norway
1 Kalman Filter

Often we don’t have access to the internal states of a system and have to reconstruct the state of the system based on the observed outputs. The idea with an observer is that we feedback the measured quantity and estimate the estimation is correct then the difference should become zero.

Another difficulty is that the measured quantities of the states
\[
\begin{align*}
\dot{\hat{x}} &= A\hat{x} + Bu(t) + K(y(t) - C\hat{x}) \\
K(y(t) - C\hat{x})
\end{align*}
\]

Here are \( e \) denoting a disturbance in the input signal that can be evaluated by the difference
\[
K(y(t) - C\hat{x})
\]

By using this quantity as feedback we obtain the observer state differential
\[
\dot{\hat{x}} = A\hat{x} + Bu(t) + K(y(t) - C\hat{x})
\]

Now form the error as
\[
\dot{\hat{x}} = A\hat{x} + Bu(t) + K(y(t) - C\hat{x})
\]

The differential error is
OpenModelica Demo
OpenModelica MDT Eclipse Plug-in

- Code Outline for easy navigation within Modelica files
- Identifier Info on Hovering
OMOptim – Optimization

Solved problems

Result plot

Export result data .csv

Pareto Front
Modelica Language Interoperability
External Functions – C, FORTRAN 77

It is possible to call functions defined outside the Modelica language, implemented in C or FORTRAN 77

```plaintext
function polynomialMultiply
  input Real a[:], b[:];
  output Real c[:] := zeros(size(a, 1)+size(b, 1) - 1);
end polynomialMultiply;
```

The body of an external function is marked with the keyword `external`

If no language is specified, the implementation language for the external function is assumed to be C. The external function `polynomialMultiply` can also be specified, e.g. via a mapping to a FORTRAN 77 function:

```plaintext
function polynomialMultiply
  input Real a[:], b[:];
  output Real c[:] := zeros(size(a, 1)+size(b, 1) - 1);
end polynomialMultiply;
```
General Tool Interoperability & Model Exchange
Functional Mock-up Interface (FMI)

The FMI development is part of the MODELISAR 29-partner project
• FMI development initiated by Daimler
• Improved Software/Model/Hardware-in-the-Loop Simulation, of physical
  models and of AUTOSAR controller models from different vendors for
  automotive applications with different levels of detail.
• Open Standard
• 14 automotive use cases for evaluation
• > 10 tool vendors are supporting it

functional mockup interface for model exchange and tool coupling

etc.

Engine with ECU  Gearbox with ECU  Thermal systems  Automated cargo door  Chassis components, roadway, ECU (e.g. ESP)

etc.
OPENPROD – Large 28-partner European Project, 2009-2012

Vision of Cyber-Physical Model-Based Product Development

Unified Modeling: Meta-modeling & Modelica & UML & OWL

OPENPROD Vision of unified modeling framework for model-driven product development from platform independent models (PIM) to platform specific models (PSM)

Current work based on Eclipse, UML/SysML, OpenModelica
OpenModelica – ModelicaML UML Profile
SysML/UML to Modelica OMG Standardization

• ModelicaML is a UML Profile for SW/HW modeling
  • Applicable to “pure” UML or to other UML profiles, e.g. SysML
• Standardized Mapping UML/SysML to Modelica
  • Defines transformation/mapping for **executable** models
  • Being **standardized** by OMG

• ModelicaML
  • Defines graphical concrete syntax (graphical notation for diagram) for representing Modelica constructs integrated with UML
  • Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
    • Which do not exist in Modelica language
    • Which are translated into executable Modelica code
  • Is defined towards generation of executable Modelica code
  • Current implementation based on the Papyrus UML tool + OpenModelica
Example: Simulation and Requirements Evaluation

Req. 001 is instantiated 2 times (there are 2 tanks in the system)

tank-height is 0.6m

Req. 001 for the tank 2 is violated

Req. 001 for the tank 1 is not violated
Faster Simulation – Compiling Modelica to Multi-Core

• **Automatic Parallelization of Mathematical Models**
  • Parallelism over the numeric solver method.
  • Parallelism over time.
  • **Parallelism over the model equation system**
    • ... with fine-grained task scheduling

• **Coarse-Grained Explicit Parallelization Using Components**
  • The programmer partitions the application into computational components using strongly-typed communication interfaces.
    • Co-Simulation, Transmission-Line Modeling (TLM)

• **Explicit Parallel Programming**
  • Providing general, easy-to-use explicit parallel programming constructs within the *algorithmic* part of the modeling language.
    • OpenCL, CUDA, ...
Some Speedup Measurements on NVIDIA Modelica Model, Generated Code, Function of Problem Size

![Graph showing simulation time as a function of problem size for different GPUs.](image)
Part III

Future Modelica and Modelica 4
Motivation for Modelica 4

1) Modelica is getting **large and complicated**, with many special cases
   - Increasingly **harder to extend** the language
   - Increasingly **harder to implement** the language in a correct way

2) Current Modelica 3.2 problems with the synchronous (event-handling) features in embedded systems contexts
   - Improvements needed for a fast and correct Embedded Systems Library
   - Goal is good support for embedded systems development
Improved Synchronous Features in Modelica

- More precise synchronous features, clock, subsampling, etc.
- Can be separated from the continuous-time parts
- Usage in generating fast code for embedded systems
- New version will be developed of Embedded Systems Library
- First version scheduled for release in the Modelica 3.3 Language; Planned for May 2012
**Approach to Modelica 4 – Future Modelica**

- **Simplify and generalize** language features
- **Partly Library-based definition**
  - Define many Modelica language features by modeling the language features in a Core Library
- This needs **core language extensions**
  - symbolic recursive data structures (trees, lists)
  - pattern matching for transformations
  - some higher-order language features.
- **Ongoing design work in Modelica Association**

- **Advantages**
  - improved correctness
  - language modularity and extensibility
MetaModelica Language Extension – Modelica 4 style
Meta-Level Operations on Models

• Model Operations
  • Creation, Query, Manipulation,
  • Composition, Management

• Manipulation of model equations for
  • Optimization purposes
  • Parallelization, Model checking
  • Simulation with different solvers

• MetaModelica language features
  • Lists, trees, pattern matching, symboli transformations
  • Garbage collection

• Very Large MetaModelica Application
  • OpenModelica compiler, implemented in 150 000 lines of MetaModelica, compiles itself efficiently.
Part IV

Modelica language concepts and textual modeling
Acausal Modeling

The order of computations is not decided at modeling time

**Acausal**

**Causal**

### Visual Component Level

- **Acausal**: ![Acausal Diagram](image1)
- **Causal**: ![Causal Diagram](image2)

### Equation Level

- **Acausal**:
  
  A resistor *equation*: 
  \[ R \cdot i = v; \]

- **Causal possibilities**:
  
  \[ i := v/R; \]
  \[ v := R \cdot i; \]
  \[ R := v/i; \]
Typical Simulation Process

“Static” semantics / compile time

Modelica model → Hybrid DAE

Elaboration

Equation Transformation & Code generation

“Dynamic” semantics / run time

Executable → Simulation Result

Simulation
Simple model - Hello World!

Equation: \( x' = -x \)
Initial condition: \( x(0) = 1 \)

```
model HelloWorld "A simple equation"
  Real x(start=1);
  parameter Real a = -1;
  equation
    der(x) = a*x;
end HelloWorld;
```

Simulation in OpenModelica environment

```plaintext
simulate(HelloWorld, stopTime = 2)
plot(x)
```
Modelica Variables and Constants

• Built-in primitive data types
  
  **Boolean**  
  true or false

  **Integer**  
  Integer value, e.g. 42 or –3

  **Real**  
  Floating point value, e.g. 2.4e-6

  **String**  
  String, e.g. “Hello world”

  **Enumeration**  
  Enumeration literal e.g. ShirtSize.Medium

• Parameters are constant during simulation

• Two types of constants in Modelica
  
  • **constant**

  ```
  constant Real PI=3.141592653589793;
  constant String redcolor = "red";
  constant Integer one = 1;
  ```

  • **parameter**

  ```
  parameter Real mass = 22.5;
  ```
A Simple Rocket Model

acceleration = \frac{thrust - mass \cdot gravity}{mass}

mass' = -massLossRate \cdot \text{abs}(thrust)

altitude' = velocity

velocity' = acceleration

```modelica
class Rocket "rocket_class"
  parameter String name;
  Real mass(start=1038.358);
  Real altitude(start=59404);
  Real velocity(start=-2003);
  Real acceleration;
  Real thrust; // Thrust force on rocket
  Real gravity; // Gravity forcefield
  parameter Real massLossRate=0.000277;
  equation
    (thrust-mass\cdot gravity)/mass = acceleration;
    der(mass) = -massLossRate \cdot \text{abs}(thrust);
    der(altitude) = velocity;
    der(velocity) = acceleration;
end Rocket;
```
Celestial Body Class

A class declaration creates a type name in Modelica

```modelica
class CelestialBody
  constant Real g = 6.672e-11;
  parameter Real radius;
  parameter String name;
  parameter Real mass;
end CelestialBody;
```

An instance of the class can be declared by prefixing the type name to a variable name

```modelica
... CelestialBody moon;
...```

The declaration states that `moon` is a variable containing an object of type `CelestialBody`
Moon Landing

```
class MoonLanding

  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  protected parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;

  public Rocket apollo(name="apollo13");
  CelestialBody moon(name="moon",mass=7.382e22,radius=1.738e6);

  equation
    apollo.thrust = if (time < thrustDecreaseTime) then force1
                   else if (time < thrustEndTime) then force2
                   else 0;
    apollo.gravity=moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end MoonLanding;
```
Simulation of Moon Landing

It starts at an altitude of 59404 (not shown in the diagram) at time zero, gradually reducing it until touchdown at the lunar surface when the altitude is zero.

The rocket initially has a high negative velocity when approaching the lunar surface. This is reduced to zero at touchdown, giving a smooth landing.
Specialized Class Keywords

- Classes can also be declared with other keywords, e.g.: model, record, block, connector, function, ...
- Classes declared with such keywords have restrictions or enhancements
- These apply to the contents of specialized classes
- After Modelica 3.0 the `class` keyword means the same as `model`

- Example: A `model` is a class that cannot be used as a connector class
- Example: A `record` is a class that only contains data, with no equations
- Example: A `block` is a class with fixed input-output causality

```model CelestialBody
    constant Real    g = 6.672e-11;
    parameter Real   radius;
    parameter String name;
    parameter Real   mass;
end CelestialBody;
```
Modelica Functions

• Modelica Functions can be viewed as a special kind of specialized class

• A function can be called with arguments, and is instantiated dynamically when called

```modelica
function sum
  input Real arg1;
  input Real arg2;
  output Real result;
  algorithm
    result := arg1+arg2;
end sum;
```
Example Modelica function call:

```modelica
...  
  p = polynomialEvaluator({1,2,3,4},21)
```

The function `PolynomialEvaluator` computes the value of a polynomial given two arguments: a coefficient vector `A` and a value of `x`.

```modelica
function PolynomialEvaluator
  // array, size defined
  // at function call time
  // default value 1.0 for x
  input Real A[:];    // array, size defined
  input Real x := 1.0; // default value 1.0 for x
  output Real sum;
protected:
  Real   xpower;          // local variable xpower
algorithm
  sum := 0;
xpower := 1;
  for i in 1:size(A,1) loop
    sum := sum + A[i]*xpower;
xpower := xpower*x;
  end for;
end PolynomialEvaluator;
```

\{1,2,3,4\} becomes the value of the coefficient vector `A`, and `21` becomes the value of the formal parameter `x`. 
Inheritance

Data and behavior: field declarations, equations, and certain other contents are *copied* into the subclass
Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color

```modelica
class Color
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
  equation
    red + blue + green = 1;
end Color;
```

```modelica
class Point
  Real x;
  Real y, z;
end Point;
```

```modelica
class ColoredPointWithoutInheritance
  Real x;
  Real y, z;
  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real green;
  equation
    red + blue + green = 1;
end ColoredPointWithoutInheritance;
```

```modelica
class ColoredPoint extends Point;
  extends Color;
end ColoredPoint;
```

Equivalent to
Simple Class Definition

• Simple Class Definition
  • Shorthand Case of Inheritance

• Example:

   class SameColor = Color;

   class SameColor extends Color;
   end SameColor;

Equivalent to:

• Often used for introducing new names of types:

   type Resistor = Real;

   connector MyPin = Pin;
Inheritance Through Modification

• Modification is a concise way of combining inheritance with declaration of classes or instances

• A modifier modifies a declaration equation in the inherited class

• Example: The class Real is inherited, modified with a different start value equation, and instantiated as an altitude variable:

  ```
  ...
  Real altitude(start= 59404);
  ...
  ```
Discrete Events and Hybrid Systems

Picture: Courtesy Hilding Elmqvist
Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling

- A point in time that is instantaneous, i.e., has zero duration
- An event condition so that the event can take place
- A set of variables that are associated with the event
- Some behavior associated with the event, e.g. conditional equations that become active or are deactivated at the event
**Event creation – if**

*if-equations, if-statements, and if-expressions*

```model Diode "Ideal diode"
  extends TwoPin;
  Real s;
  Boolean off;
  equation
    off = s < 0;
    if off then
      v = s
    else
      v = 0;
    end if;
  i = if off then 0 else s;
end Diode;
```

- False if $s<0$
- If-equation choosing equation for $v$
- If-expression
Event creation – when

**when-equations**

```modelica
definition when-equations =
  when <conditions> then
    <equations>
  end when;
```

#### Time event

```modelica
definition time-event =
  when time >= 10.0 then
    ...
  end when;
```

- Only dependent on time, can be scheduled in advance

#### State event

```modelica
definition state-event =
  when sin(x) > 0.5 then
    ...
  end when;
```

- Related to a state. Check for zero-crossing

Equations only active at event times
Generating Repeated Events

The call \( \text{sample}(t_0, d) \) returns true and triggers events at times \( t_0 + i \cdot d \), where \( i=0,1,... \)

```model SamplingClock
  Integer i;
  discrete Real r;
  equation
    when sample(2, 0.5) then
      i = pre(i)+1;
      r = pre(r)+0.3;
    end when;
  end SamplingClock;
```

Variables need to be discrete

Creates an event after 2 s, then each 0.5 s

pre(...) takes the previous value before the event.
Reinit - discontinuous changes

The value of a *continuous-time* state variable can be instantaneously changed by a `reinit`-equation within a `when`-equation

```model BouncingBall "the bouncing ball model"
  parameter Real g=9.81;  //gravitational acc.
  parameter Real c=0.90;  //elasticity constant
  Real height(start=10),velocity(start=0);
  equation
    der(height) = velocity;
    der(velocity)=-g;
    when height<0 then
      reinit(velocity, -c*velocity);
    end when;
  end BouncingBall;
```

Reinit "assigns" continuous-time variable `velocity` a new value

Initial conditions
Part V

Components, Connectors and Connections – Modelica Libraries
A component class should be defined *independently of the environment*, very essential for *reusability*

A component may internally consist of other components, i.e. *hierarchical* modeling

Complex systems usually consist of large numbers of *connected* components
Connectors and Connector Classes

Connectors are instances of *connector classes*

**electrical connector**

**connector class**

**keyword** flow indicates that currents of connected pins sum to zero.

**an instance pin** of class Pin

**mechanical connector**

**connector class**

**an instance flange** of class Flange
The flow prefix

Two kinds of variables in connectors:

- Non-flow variables potential or energy level
- Flow variables represent some kind of flow

Coupling

- Equality coupling, for non-flow variables
- Sum-to-zero coupling, for flow variables

The value of a flow variable is positive when the current or the flow is into the component.
### Physical Connector

#### Classes Based on Energy Flow

<table>
<thead>
<tr>
<th>Domain Type</th>
<th>Potential</th>
<th>Flow</th>
<th>Carrier</th>
<th>Modelica Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical</td>
<td>Voltage</td>
<td>Current</td>
<td>Charge</td>
<td>Electrical. Analog</td>
</tr>
<tr>
<td>Translational</td>
<td>Position</td>
<td>Force</td>
<td>Linear momentum</td>
<td>Mechanical. Translational</td>
</tr>
<tr>
<td>Rotational</td>
<td>Angle</td>
<td>Torque</td>
<td>Angular momentum</td>
<td>Mechanical. Rotational</td>
</tr>
<tr>
<td>Magnetic</td>
<td>Magnetic potential</td>
<td>Magnetic flux rate</td>
<td>Magnetic flux</td>
<td></td>
</tr>
<tr>
<td>Hydraulic</td>
<td>Pressure</td>
<td>Volume flow</td>
<td>Volume</td>
<td>HyLibLight</td>
</tr>
<tr>
<td>Heat</td>
<td>Temperature</td>
<td>Heat flow</td>
<td>Heat</td>
<td>HeatFlow1D</td>
</tr>
<tr>
<td>Chemical</td>
<td>Chemical potential</td>
<td>Particle flow</td>
<td>Particles</td>
<td>Under construction</td>
</tr>
<tr>
<td>Pneumatic</td>
<td>Pressure</td>
<td>Mass flow</td>
<td>Air</td>
<td>PneuLibLight</td>
</tr>
</tbody>
</table>
Connections between connectors are realized as equations in Modelica:

\[ \text{connect}(\text{connector1, connector2}) \]

The two arguments of a connect-equation must be references to connectors, either to be declared directly within the same class or be members of one of the declared variables in that class.

\[ \text{Pin pin1, pin2; // A connect equation} \]

// in Modelica:
\[ \text{connect(pin1, pin2);} \]

Using Kirschoff’s Laws:

\[ \text{Connect to} \]
\[ \text{Corresponds to} \]

\[ \text{pin1.v} = \text{pin2.v}; \]
\[ \text{pin1.i + pin2.i} = 0; \]
Common Component Structure

The base class `TwoPin` has two connectors `p` and `n` for positive and negative pins respectively.

```modelica
partial model TwoPin
  Pin p; Pin n;
  equation
    v = p.v - n.v;
    0 = p.i + n.i;
    i = p.i;
end TwoPin;

// TwoPin is same as OnePort in Modelica.Electrical.Analog.Interfaces
```

electrical connector class

- partial class (cannot be instantiated)
- positive pin
- negative pin
Electrical Components

model Resistor "Ideal electrical resistor"
  extends TwoPin;
  parameter Real R;
  equation
  R*i = v;
end Resistor;

model Inductor "Ideal electrical inductor"
  extends TwoPin;
  parameter Real L "Inductance";
  equation
  L*der(i) = v;
end Inductor;

model Capacitor "Ideal electrical capacitor"
  extends TwoPin;
  parameter Real C;
  equation
  i=C*der(v);
end Capacitor;
model Source
  extends TwoPin;
  parameter Real A, w;
  equation
    v = A*sin(w*time);
  end Resistor;

model Ground
  Pin p;
  equation
    p.v = 0;
  end Ground;
Resistor Circuit – Very Simple Example

```model ResistorCircuit
Resistor R1(R=100);
Resistor R2(R=200);
Resistor R3(R=300);
equation
connect(R1.p, R2.p);
connect(R1.p, R3.p);
end ResistorCircuit;

R1.p.v = R2.p.v;
R1.p.v = R3.p.v;
R1.p.i + R2.p.i + R3.p.i = 0;
```
Modelica Standard Library - Graphical Modeling

- *Modelica Standard Library* (called Modelica) is a standardized predefined package developed by Modelica Association.

- It can be used freely for both commercial and noncommercial purposes under the conditions of *The Modelica License*.

- Modelica libraries are available online including documentation and source code from [http://www.modelica.org/library/library.html](http://www.modelica.org/library/library.html)
The Modelica Standard Library contains components from various application areas, including the following sublibraries:

- **Blocks**  
  Library for basic input/output control blocks
- **Constants**  
  Mathematical constants and constants of nature
- **Electrical**  
  Library for electrical models
- **Icons**  
  Icon definitions
- **Fluid**  
  1-dim Flow in networks of vessels, pipes, fluid machines, valves, etc.
- **Math**  
  Mathematical functions
- **Magnetic**  
  Magnetic.Fluctubes — for magnetic applications
- **Mechanics**  
  Library for mechanical systems
- **Media**  
  Media models for liquids and gases
- **Slunits**  
  Type definitions based on SI units according to ISO 31-1992
- **Stategraph**  
  Hierarchical state machines (analogous to Statecharts)
- **Thermal**  
  Components for thermal systems
- **Utilities**  
  Utility functions especially for scripting
Modelica.Blocks

Continuous, discrete, and logical input/output blocks to build block diagrams.

Examples:
Modelica.Electrical

Electrical components for building analog, digital, and multiphase circuits

Examples:
Modelica.Mechanics

Package containing components for mechanical systems

Subpackages:

- Rotational 1-dimensional rotational mechanical components
- Translational 1-dimensional translational mechanical components
- MultiBody 3-dimensional mechanical components
Hierarchical state machines (similar to Statecharts)
Other Free Libraries

- **WasteWater**  Wastewater treatment plants, 2003
- **ATPlus**  Building simulation and control (fuzzy control included), 2005
- **MotorCycleDynamics**  Dynamics and control of motorcycles, 2009
- **NeuralNetwork**  Neural network mathematical models, 2006
- **VehicleDynamics**  Dynamics of vehicle chassis (obsolete), 2003
- **SPICElib**  Some capabilities of electric circuit simulator PSPICE, 2003
- **SystemDynamics**  System dynamics modeling a la J. Forrester, 2007
- **BondLib**  Bond graph modeling of physical systems, 2007
- **MultiBondLib**  Multi bond graph modeling of physical systems, 2007
- **ModelicaDEVS**  DEVS discrete event modeling, 2006
- **ExtendedPetriNets**  Petri net modeling, 2002
- **External.Media Library**  External fluid property computation, 2008
- **VirtualLabBuilder**  Implementation of virtual labs, 2007
- **SPOT**  Power systems in transient and steady-state mode, 2007
- **...**
Some Commercial Libraries

- Powertrain
- SmartElectricDrives
- VehicleDynamics
- AirConditioning
- HyLib
- PneuLib
- CombiPlant
- HydroPlant
- ...

[Logo] Modelica
Connect Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain

```model Generator
  Modelica.Mechanics.Rotational.Inertia iner;
  Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
  Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
  Modelica.Electrical.Analog.Basic.Resistor R1,R2;
  Modelica.Blocks.Sources.Exponentials ex(riseTime=[2],riseTimeConst=[1]);

  equation
    connect(ac.flange_b, iner.flange_a); connect(iner.flange_b, emf.flange_b);
    connect(emf.p, ind.p); connect(ind.n, R1.p); connect(emf.n, G.p);
    connect(emf.n, R2.n); connect(R1.n, R2.p); connect(R2.p, vsens.n);
    connect(R2.n, vsens.p); connect(ex.outPort, ac.inPort);
  end Generator;
```
Get More Information, Download Software

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


- OpenModelica
  - www.openmodelica.org

- Modelica Association
  - www.modelica.org
New Introductory Book
September 2011
232 pages

Wiley
IEEE Press

For Introductory Short Courses on Object Oriented Mathematical Modeling
Summary

Multi-Domain Modeling

Visual Acausal Component Modeling

www.modelica.org – Language, Standard Library
www.openmodelica.org – Open Source Tool

Typed Declarative Textual Language

Thanks for listening!

Modelica® is a registered trademark of the Modelica Association