Towards Unified System Modeling with the ModelicaML UML Profile

Adrian Pop, David Akhvlediani, Peter Fritzson

Programming Environments Laboratory,
Department of Computer and Information Science
Linköping University
adrpo@ida.liu.se

EOLT'2007, 2007-07-30

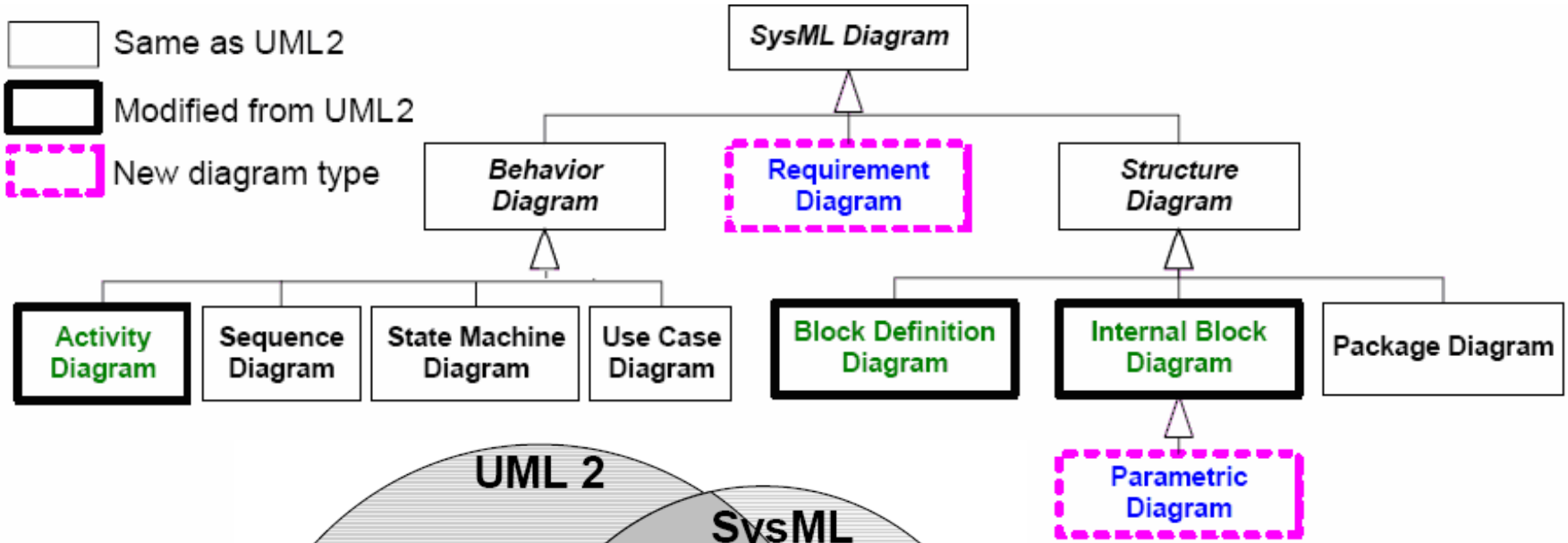
Outline

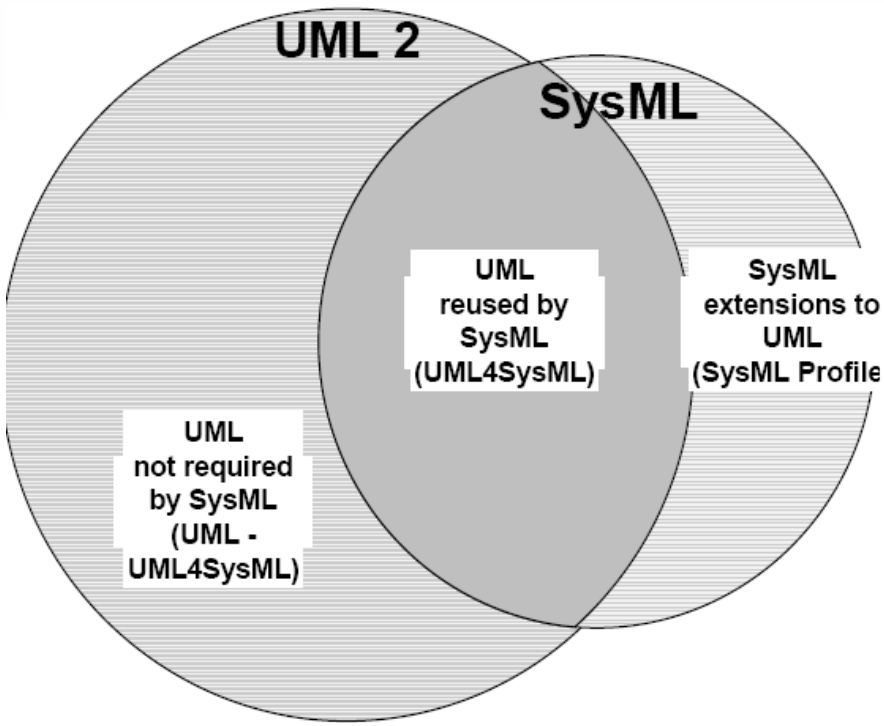
- Introduction
  - System Modeling Language (SysML™)
  - Modelica
- ModelicaML: a UML profile for Modelica
  - Overview and Purpose
  - Diagrams
    - Package Diagram
    - Class Diagram and Internal Class Diagram
    - Equation Diagram
    - Simulation Diagram
- Conclusions and Future Work

System Modeling Language (SysML™)

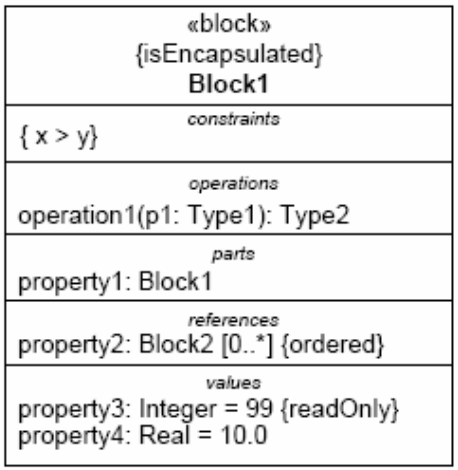
- Graphical modeling language for Systems Engineering constructed as a UML2 Profile
- Designed to provide simple but powerful constructs for modeling a wide range of systems engineering problems
- Effective in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis
- Intended to support multiple processes and methods such as structured, object-oriented, etc.

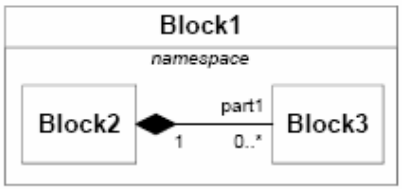
SysML™ - Diagrams

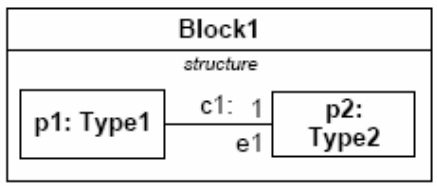


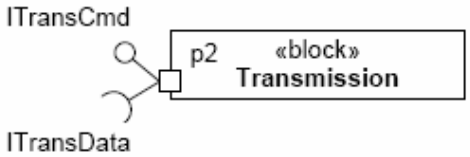


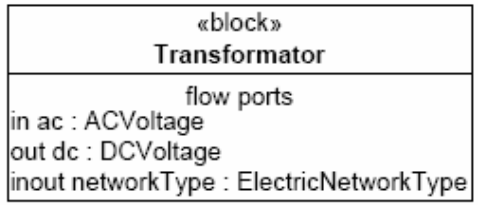
SysML™ - Block Definitions

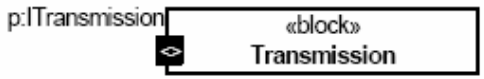






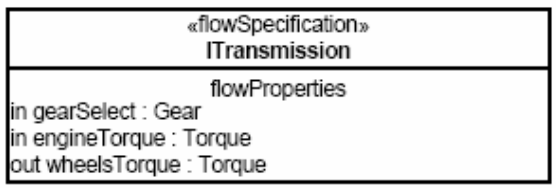


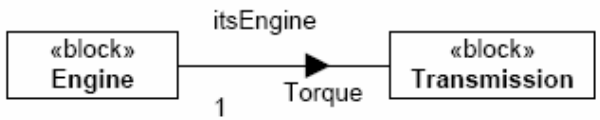




Conjugated Flow port

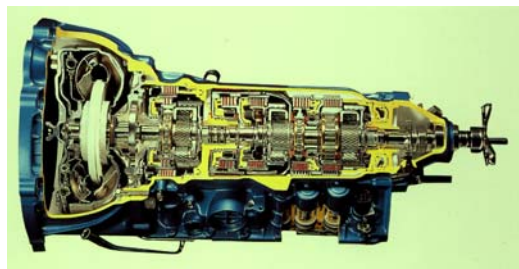
Atomic Flow Ports

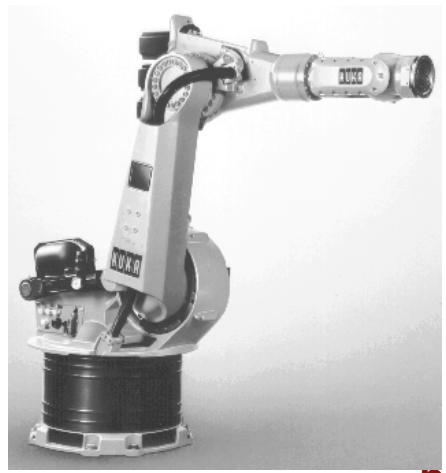




Modelica – General Formalism to Model Complex Systems

- Robotics
- Automotive
- Aircrafts
- Satellites
- Biomechanics
- Power plants
- Hardware-in-the-loop, real-time simulation
- etc





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, nonproprietary*
  - Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations

Modelica Language Properties

- *Declarative* and *Object-Oriented*
- *Equation-based*; continuous and discrete equations
- *Parallel process* modeling of concurrent applications, according to synchronous data flow principle
- *Functions* with algorithms without global side-effects
  (but local data updates allowed)
- *Type system* inspired by
  Abadi/Cardelli (Theory of Objects)
- *Everything is a class* – Real, Integer, models, functions, packages, parameterized classes….

Modelica Acausal Modeling Semantics

- What is *acausal* modeling/design?
- Why does it increase *reuse*?
  - The acausality makes Modelica classes *more reusable* than traditional classes containing assignment statements where the input-output causality is fixed.
- Example: a resistor *equation*:

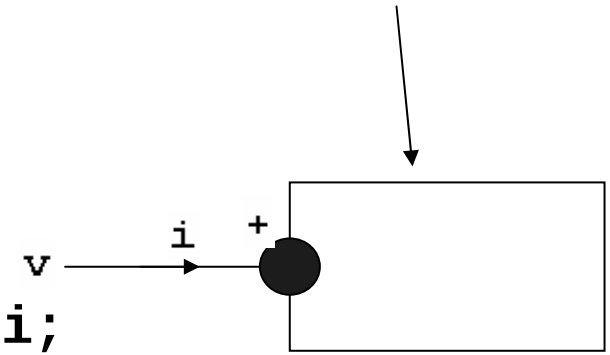
```
R*i = v;
```

- can be used in three ways:

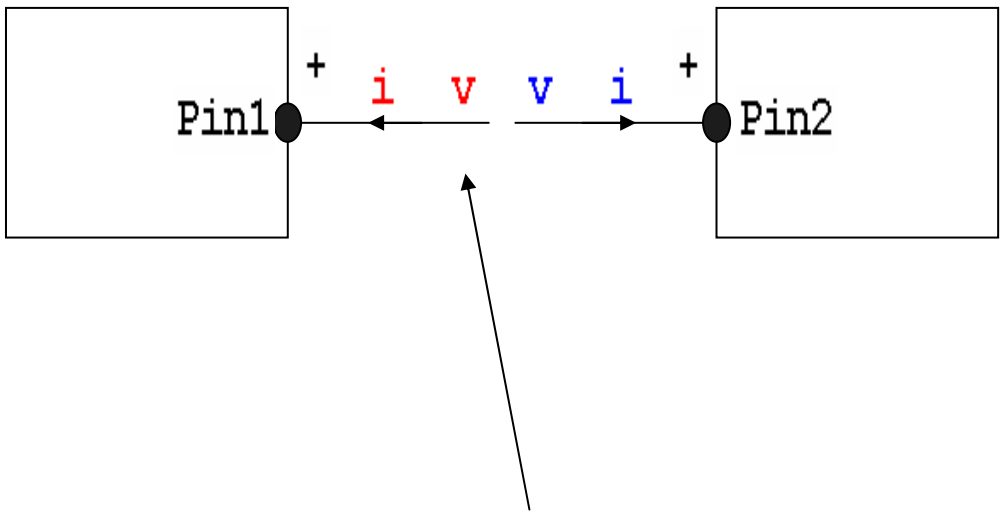
```
i := v/R;
v := R*i;
R := v/i;
```

Connector Classes, Components and Connections

```
connector Pin
   Voltage v;
   flow Current i;
end Pin;
```



Keyword **flow** indicates that currents of connected pins sums to zero.



A connect statement in Modelica

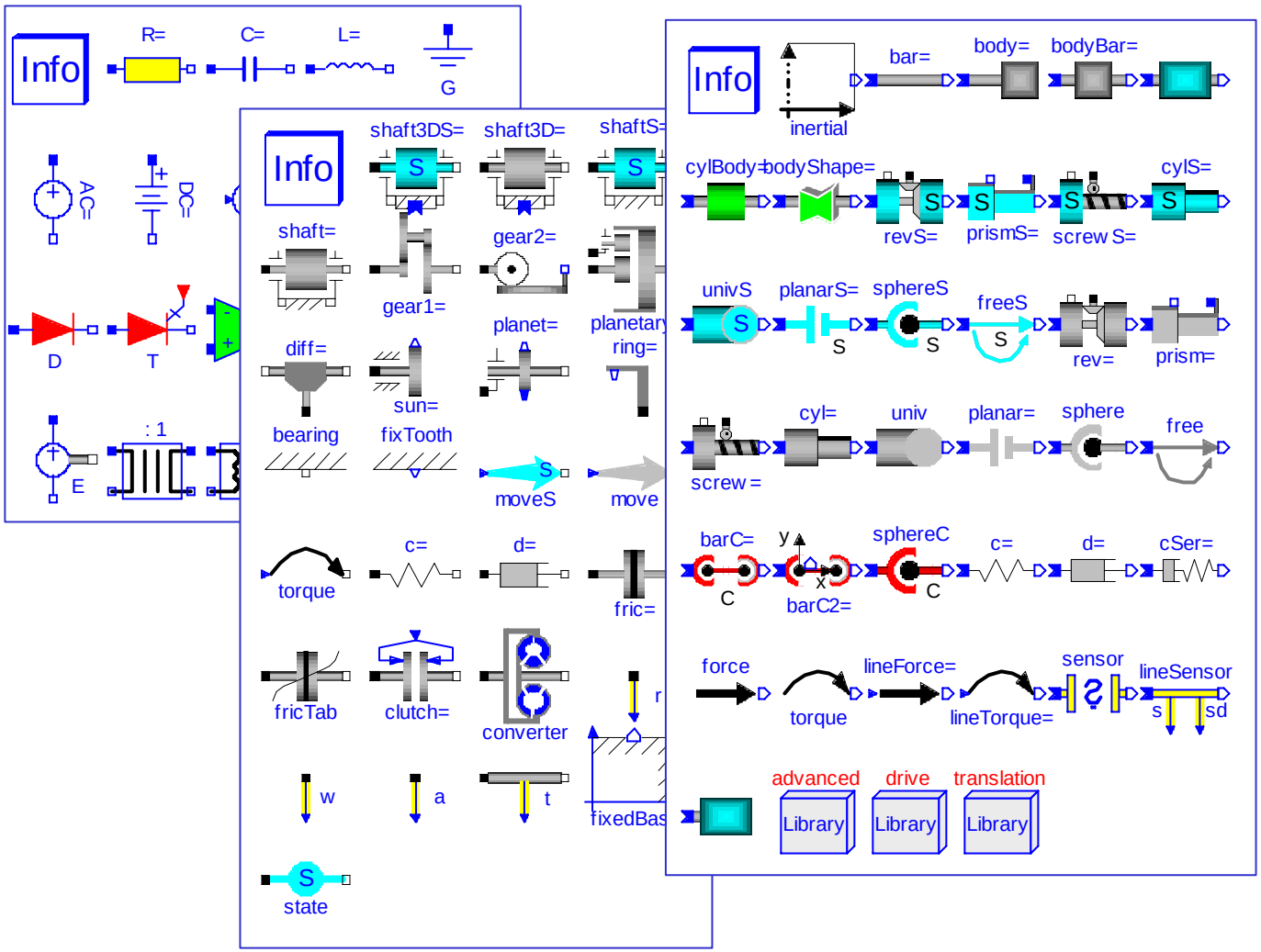
```
connect(Pin1,Pin2)
```

corresponds to equations:

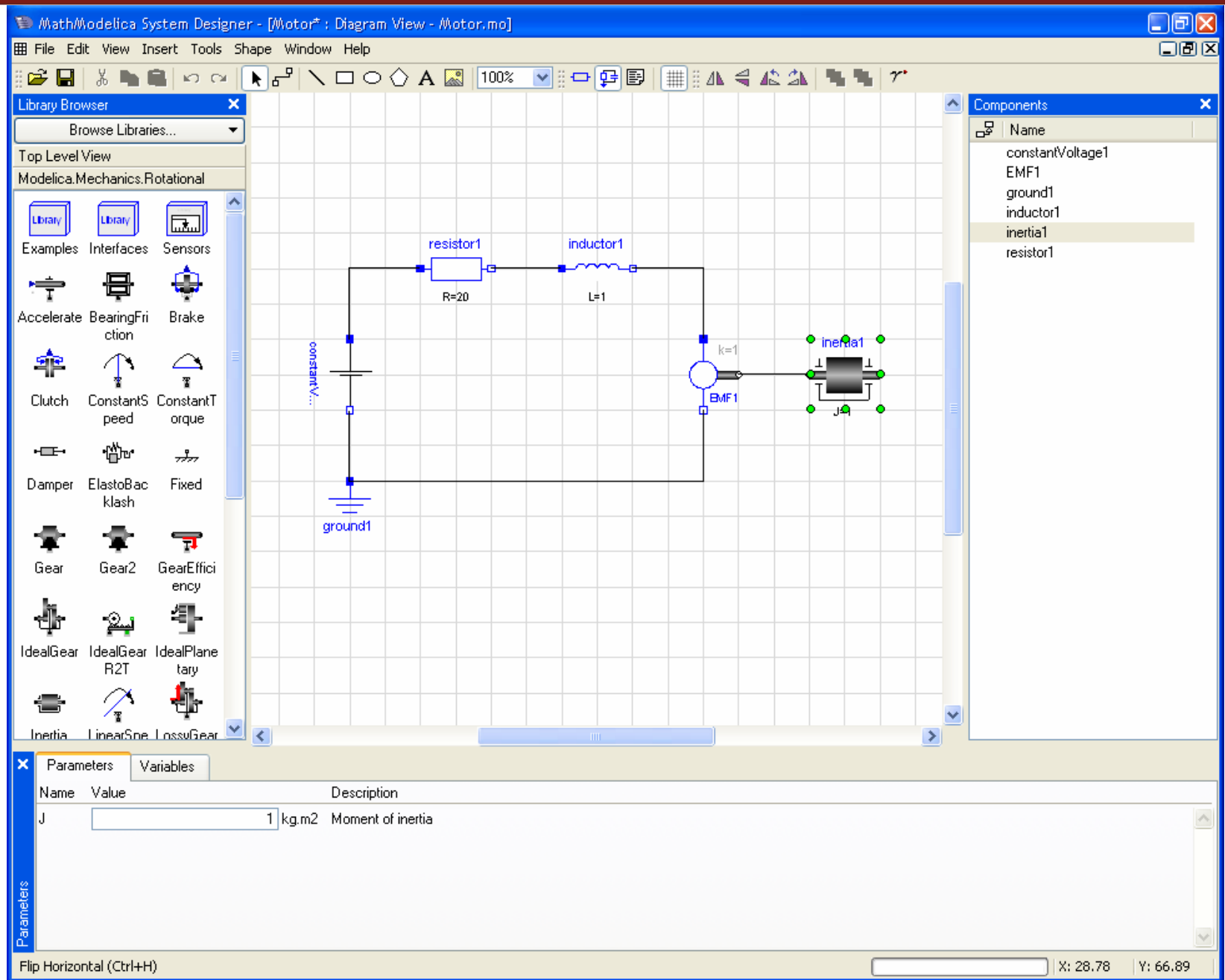
```
Pin1.v = Pin2.v
Pin1.i + Pin2.i = 0
```

Connection between Pin1 and Pin2

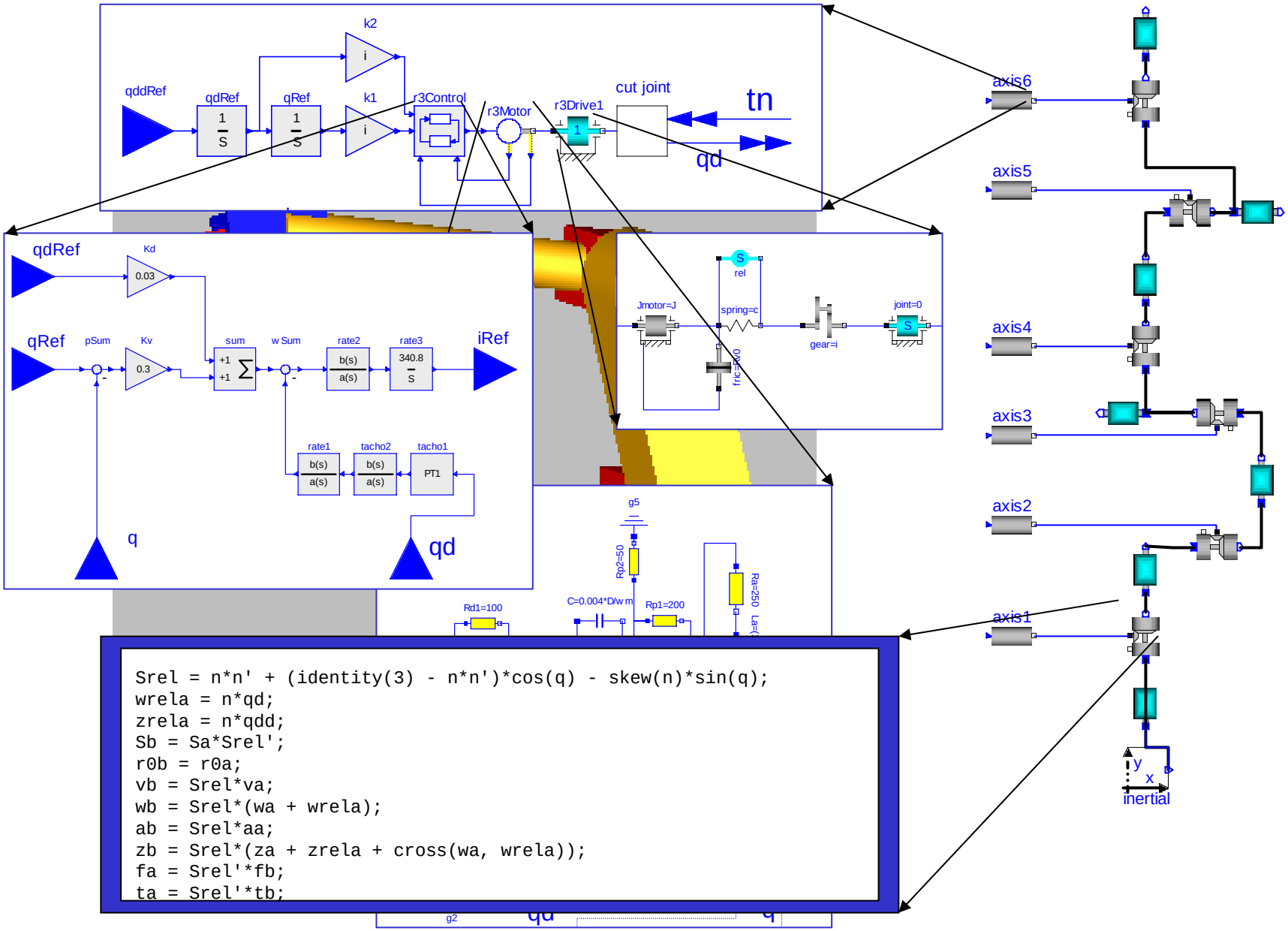
Modelica - Reusable Class Libraries



Graphical Modeling - Drag and Drop Composition



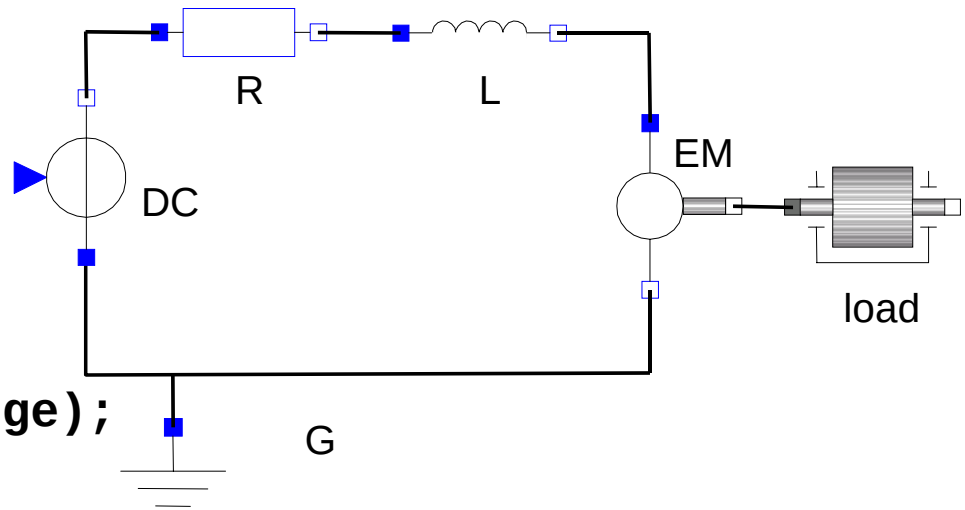
Hierarchical Composition Diagram for a Model of a Robot



Multi-Domain Modelica Model - DCMotor

- 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
```



SysML vs. Modelica

- SysML
  - Pros
    - Can model all aspects of complex system design
  - Cons
    - Precise behavior can be described *but not simulated (executed)*
- Modelica
  - Pros
    - Precise behavior *can be described and simulated*
  - Cons
    - Cannot model all aspects of complex system design, i.e. requirements, inheritance diagrams, etc

Outline so far

- Introduction
  - System Modeling Language (SysML™)
  - Modelica
- ModelicaML: a UML profile for Modelica
  - Overview and Purpose
  - Diagrams
    - Package Diagram
    - Class Diagram and Internal Class Diagram
    - Equation Diagram
    - Simulation Diagram
- Conclusions and Future Work

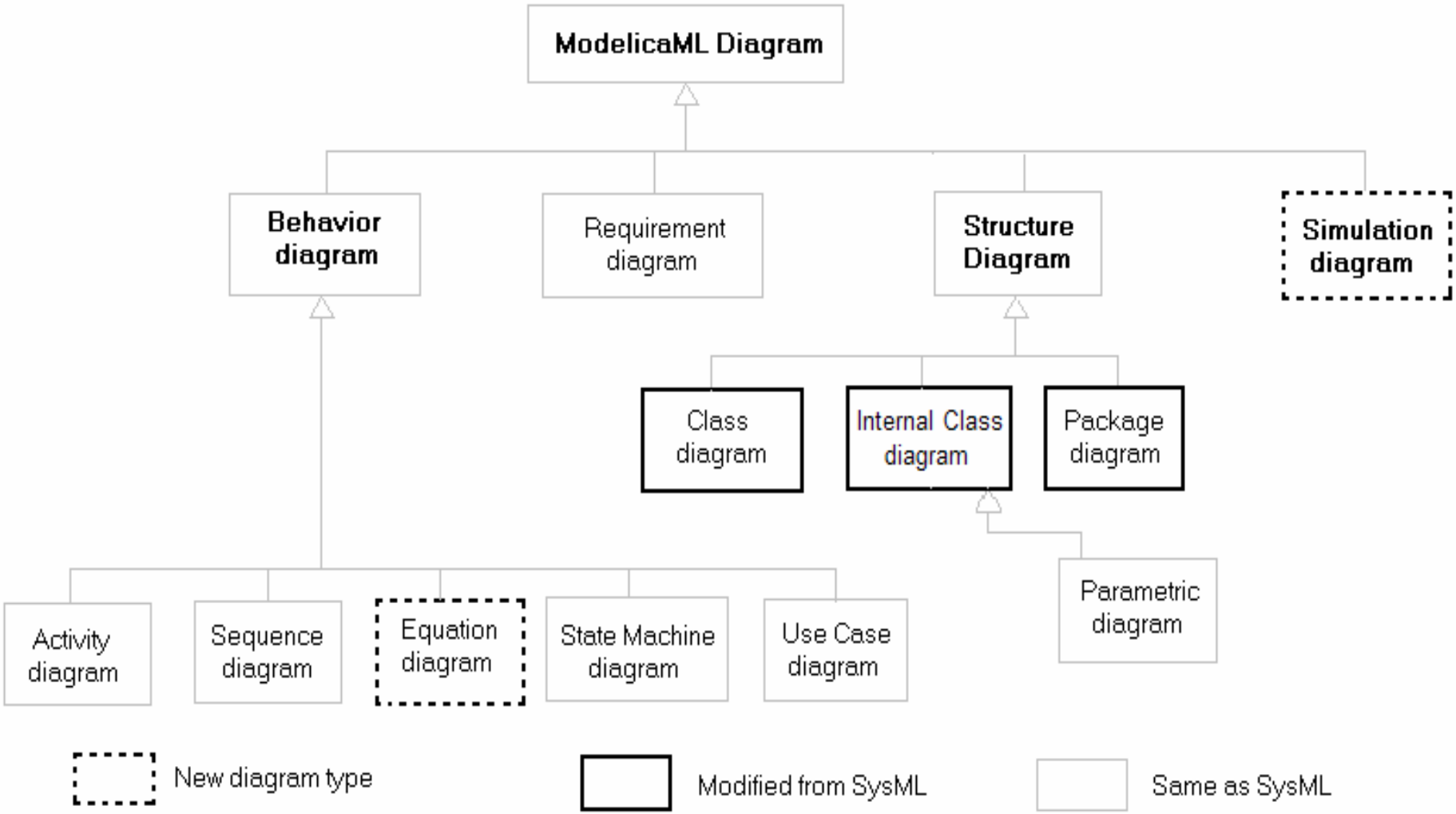
ModelicaML - a UML profile for Modelica

- **Supports modeling with all Modelica constructs i.e. restricted classes, equations, generics, discrete variables, etc.**
- **Multiple aspects of a system being designed are supported**
  - **system development process phases such as requirements analysis, design, implementation, verification, validation and integration.**
- **Supports mathematical modeling with equations (to specify system behavior). Algorithm sections are also supported.**
- **Simulation diagrams are introduced to configure, model and document simulation parameters and results in a consistent and usable way.**
- **The ModelicaML meta-model is consistent with SysML in order to provide SysML-to-ModelicaML conversion and back.**

ModelicaML - Purpose

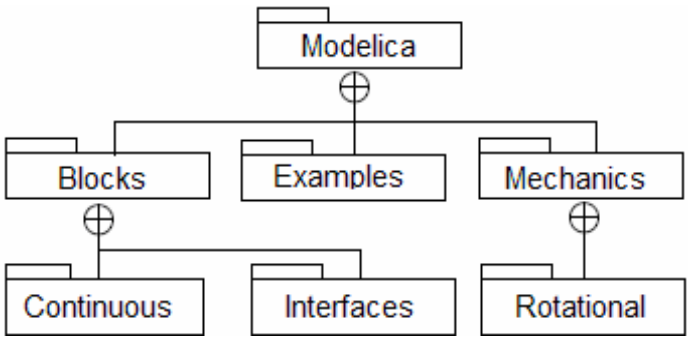
- *Targeted to Modelica and SysML users*
- *Provide a SysML/UML view of Modelica for*
  - *Documentation purposes*
  - *Language understanding*
- *To extend Modelica with additional design capabilities (requirements modeling, inheritance diagrams, etc)*
- *To support translation between Modelica and SysML models via XMI*

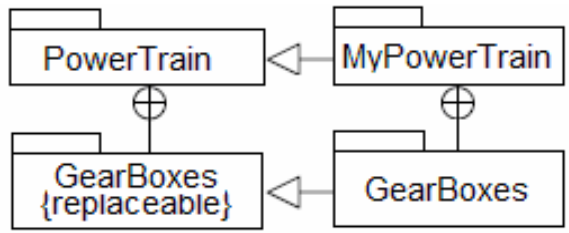
ModelicaML - Overview

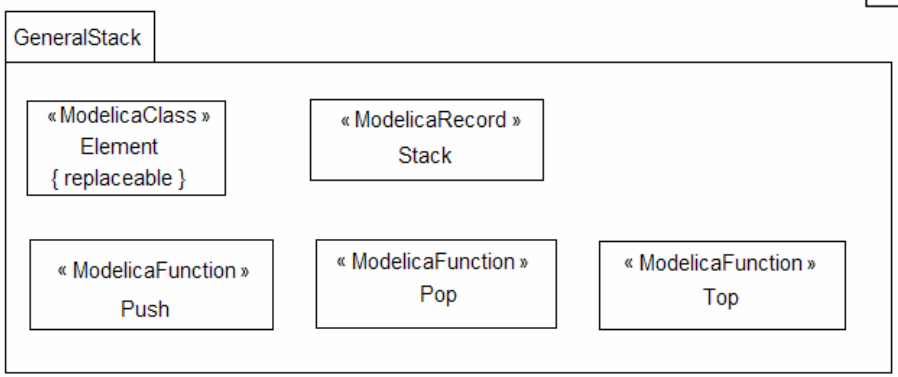


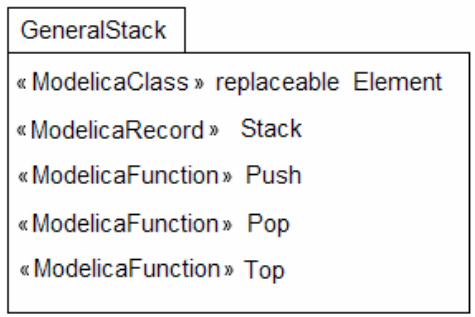
ModelicaML – Package Diagram

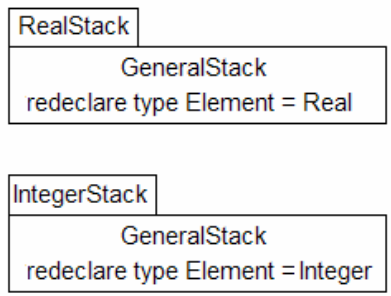
- The Package Diagram groups logically connected user defined elements into packages.
- The primarily purpose of this diagram is to support the specifics of the Modelica packages.





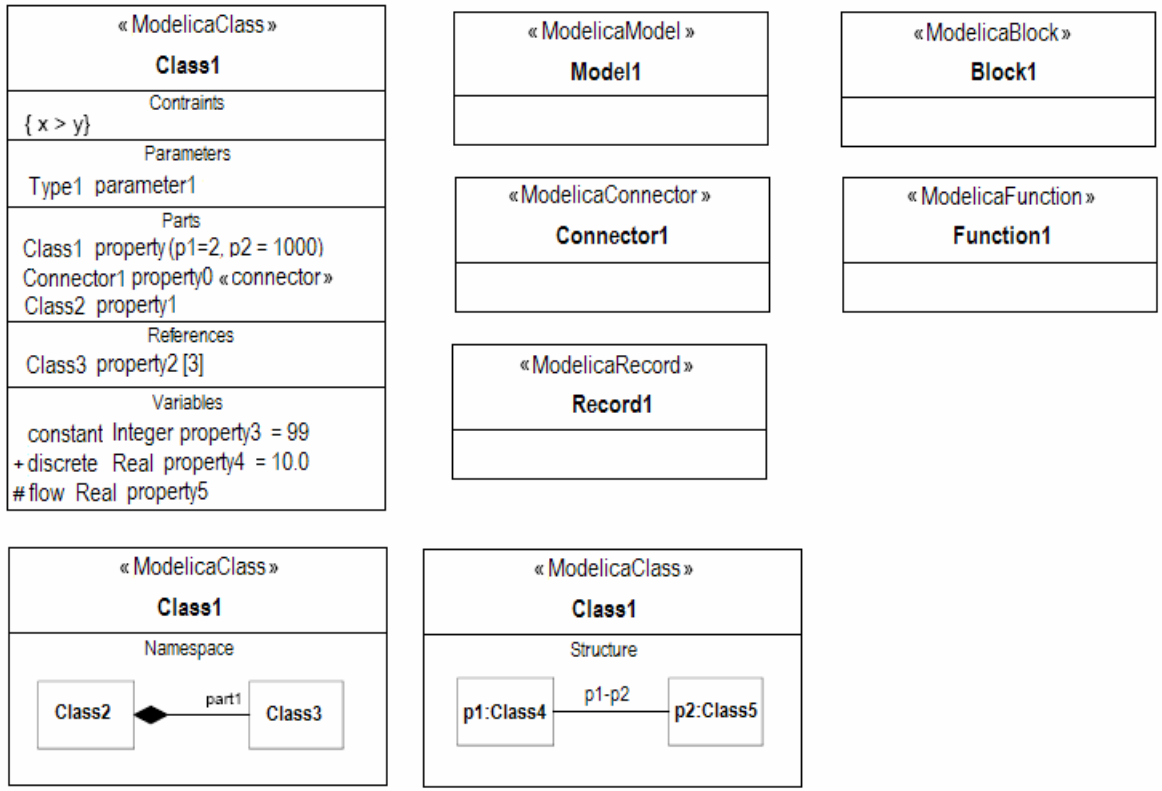






ModelicaML – Class Diagram

- ModelicaML provides extensions to SysML in order to support the full set of Modelica constructs.
- ModelicaML defines unique class definition types ModelicaClass, ModelicaModel, ModelicaBlock, ModelicaConnector, ModelicaFunction and ModelicaRecord that correspond to `class`, `model`, `block`, `connector`, `function` and `record` restricted Modelica classes.
- Modelica specific restricted classes are included because a modeling tool needs to impose their semantic restrictions (for example a record cannot have equations, etc).



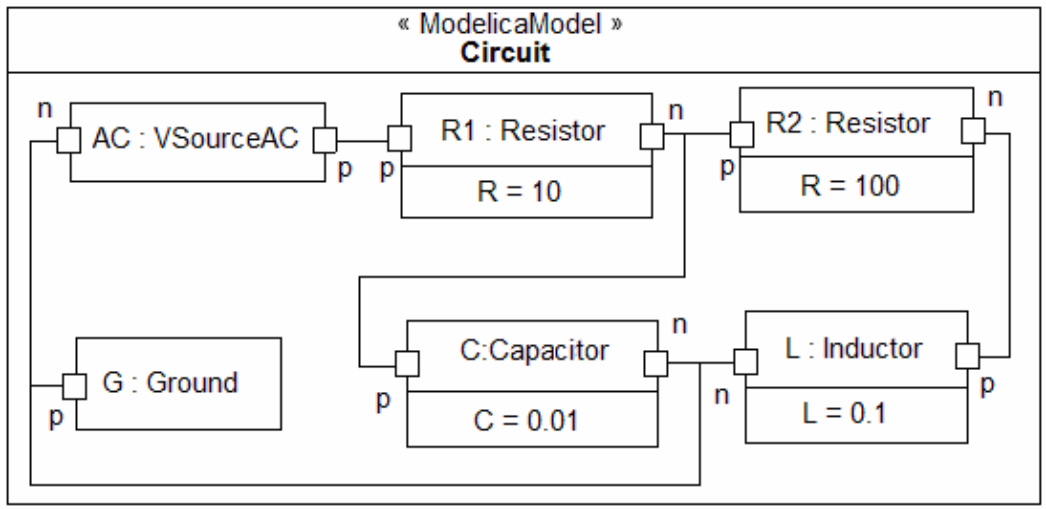
Class Diagram defines Modelica classes and relationships between classes, like generalizations, association and dependencies

ModelicaML - Internal Class Diagram

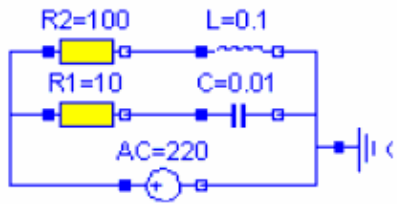
- Internal Class Diagram shows the internal structure of a class in terms of parts and connections

ModelicaML Internal Class Diagram

```
model Circuit
  Resistor  R1(R=10);
  Capacitor C(C=0.01);
  Resistor  R2(R=100);
  Inductor  L(L=0.1);
  VSourceAC AC;
  Ground    G;
equation
  connect (AC.p, R1.p);
  connect (R1.n, C.p);
  connect (C.n, AC.n);
  connect (R1.n, R2.p);
  connect (R2.n, L.p);
  connect (L.n,  C.n);
  connect (AC.n, G.p);
end Circuit;
```



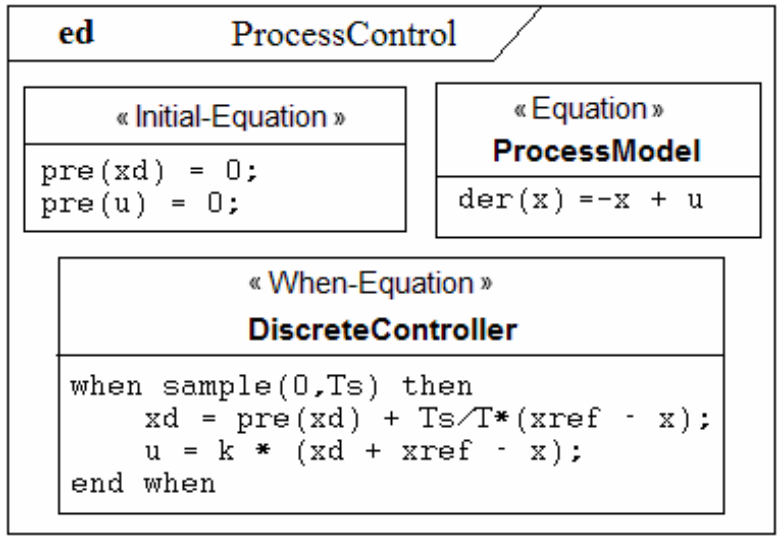
Modelica Connection Diagram

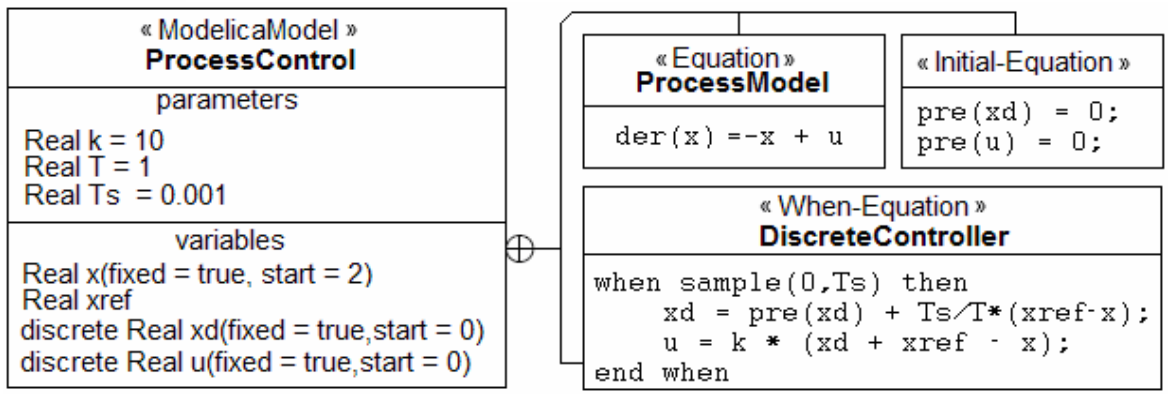


ModelicaML – Equation Diagram

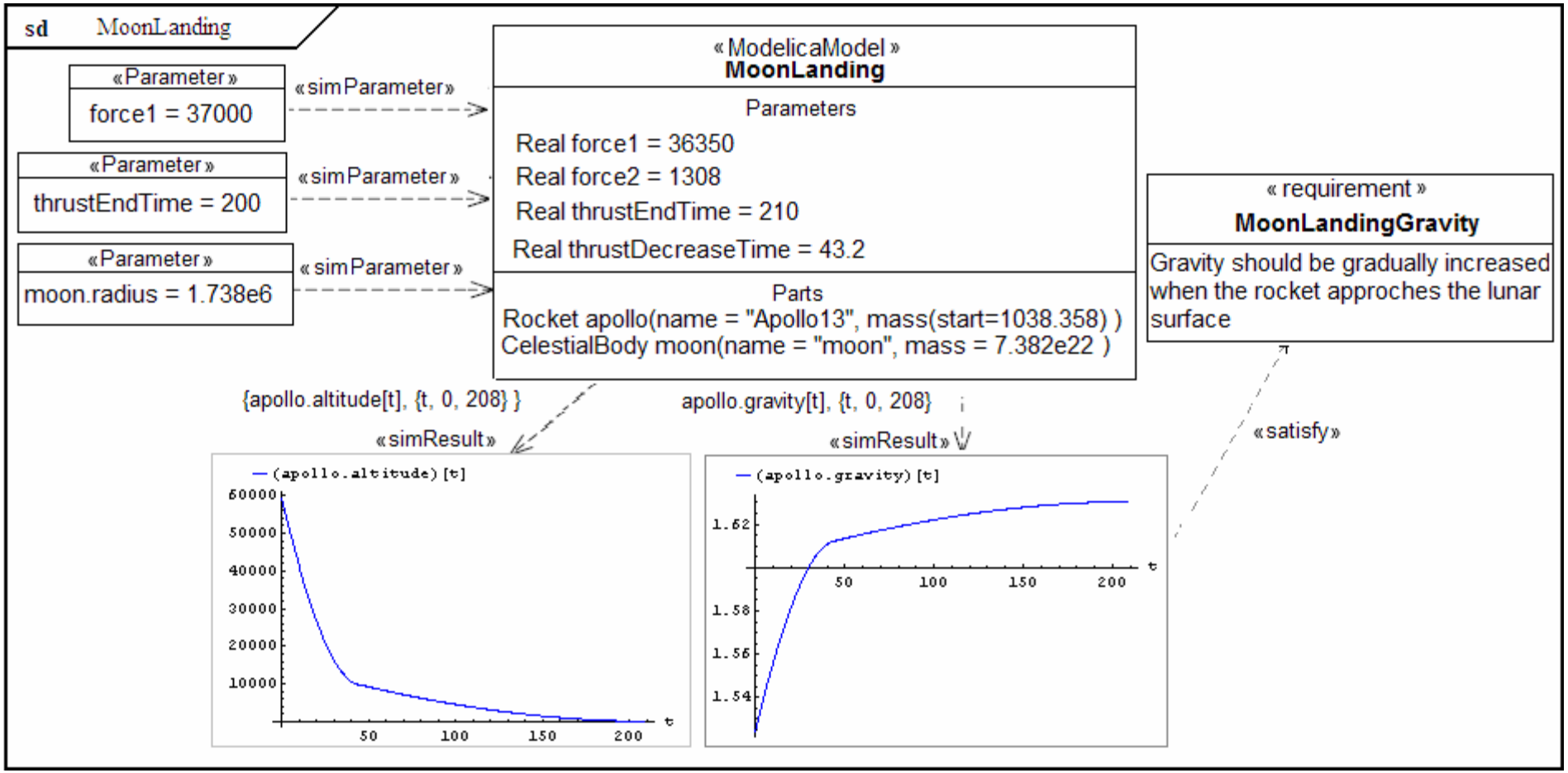
- behavior is specified using Equation Diagrams
- all Modelica equations have their specific diagram:
  - initial, when, for, if equations

```
model ProcessControl
  parameter Real k=10,T=1;
  parameter Real Ts=0.001;
  Real x(fixed=true,start=2);
  Real xref;
  discrete Real xd(fixed=true,start=0);
  discrete Real u(fixed=true,start=0);
equation
  der(x) = =-x + u; // Process model
  // Discrete PI Controller
  when sample(0,Ts) then
    xd=pre(xd)+Ts/T*(xref-x);
    u=k*(xd + xref - x);
  end when;
initial equation
  pre(xd) = 0; pre(u) = 0;
end ProcessControl;
```





ModelicaML – Simulation Diagram



- Used to model, configure and document simulation parameters and results
- Simulation diagrams can be integrated with any Modelica modeling and simulation environment (OpenModelica)

Outline so far

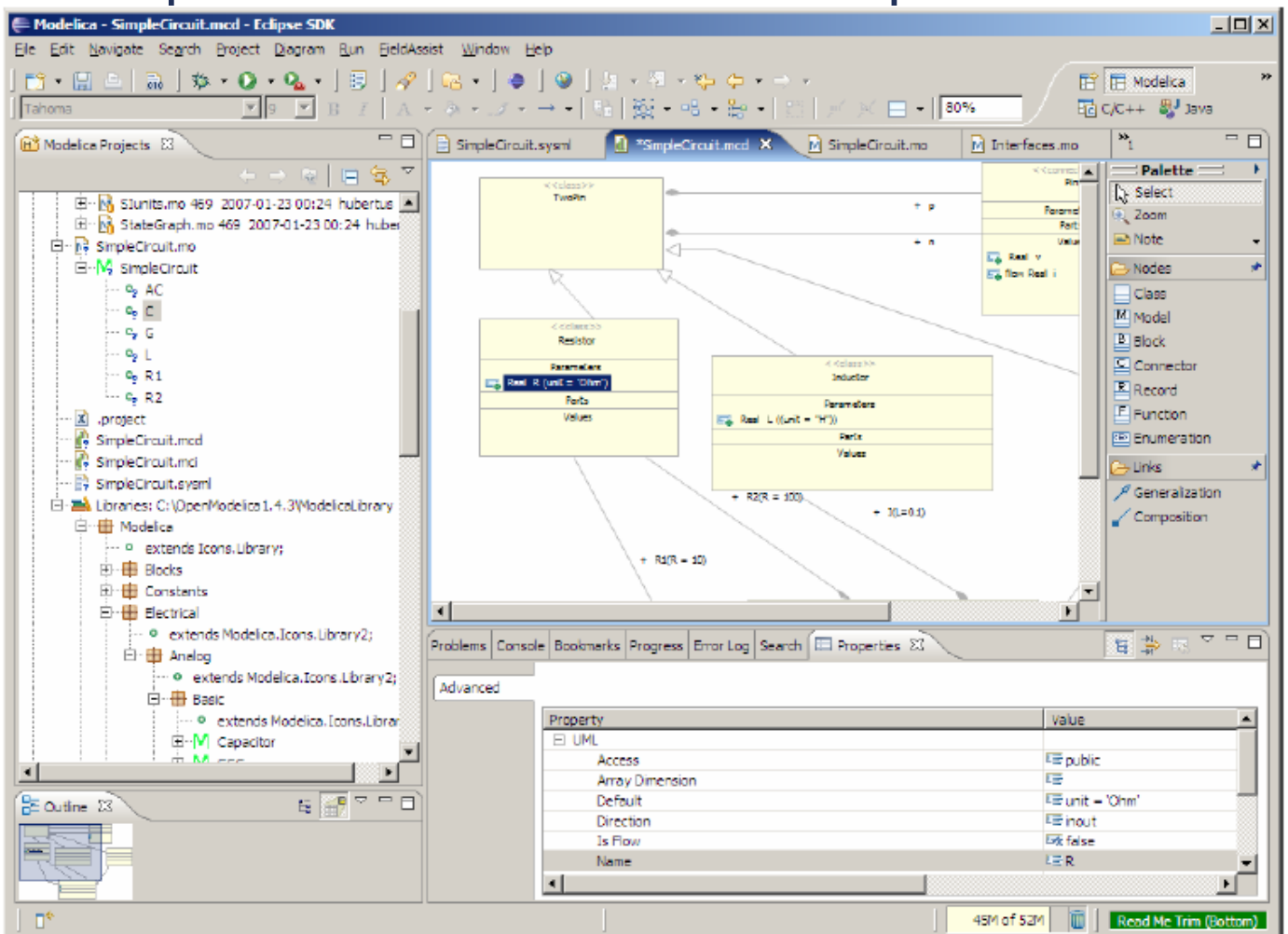
- Introduction
  - System Modeling Language (SysML™)
  - Modelica
- ModelicaML: a UML profile for Modelica
  - Overview and Purpose
  - Diagrams
    - Package Diagram
    - Class Diagram and Internal Class Diagram
    - Equation Diagram
    - Simulation Diagram
- Conclusions and Future Work

Conclusions

- ModelicaML – UML profile for Modelica
  - *Targeted to Modelica and SysML users*
  - *Provides a SysML/UML view of Modelica for*
    - *Documentation purposes*
    - *Language understanding*
  - *extends Modelica with additional design capabilities (requirements modeling, inheritance diagrams, etc)*
  - *supports translation between Modelica and SysML models via XMI*

Future Work

- integration with Modelica Development Tooling (MDT) and the OpenModelica Compiler
- Translation between Modelica, ModelicaML and SysML
- Further improvements to ModelicaML specification



Thank You!
Questions?

Modelica Development Tooling (MDT)
http://www.ida.liu.se/~pelab/modelica/OpenModelica/MDT/

OpenModelica Project
http://www.ida.liu.se/~pelab/modelica/OpenModelica.html