Composition of XML Dialects

A ModelicaXML case study

Adrian Pop, Ilie Savga, Uwe Aßmann, Peter Fritzson
Programming Environments Laboratory
Linköping University

Outline

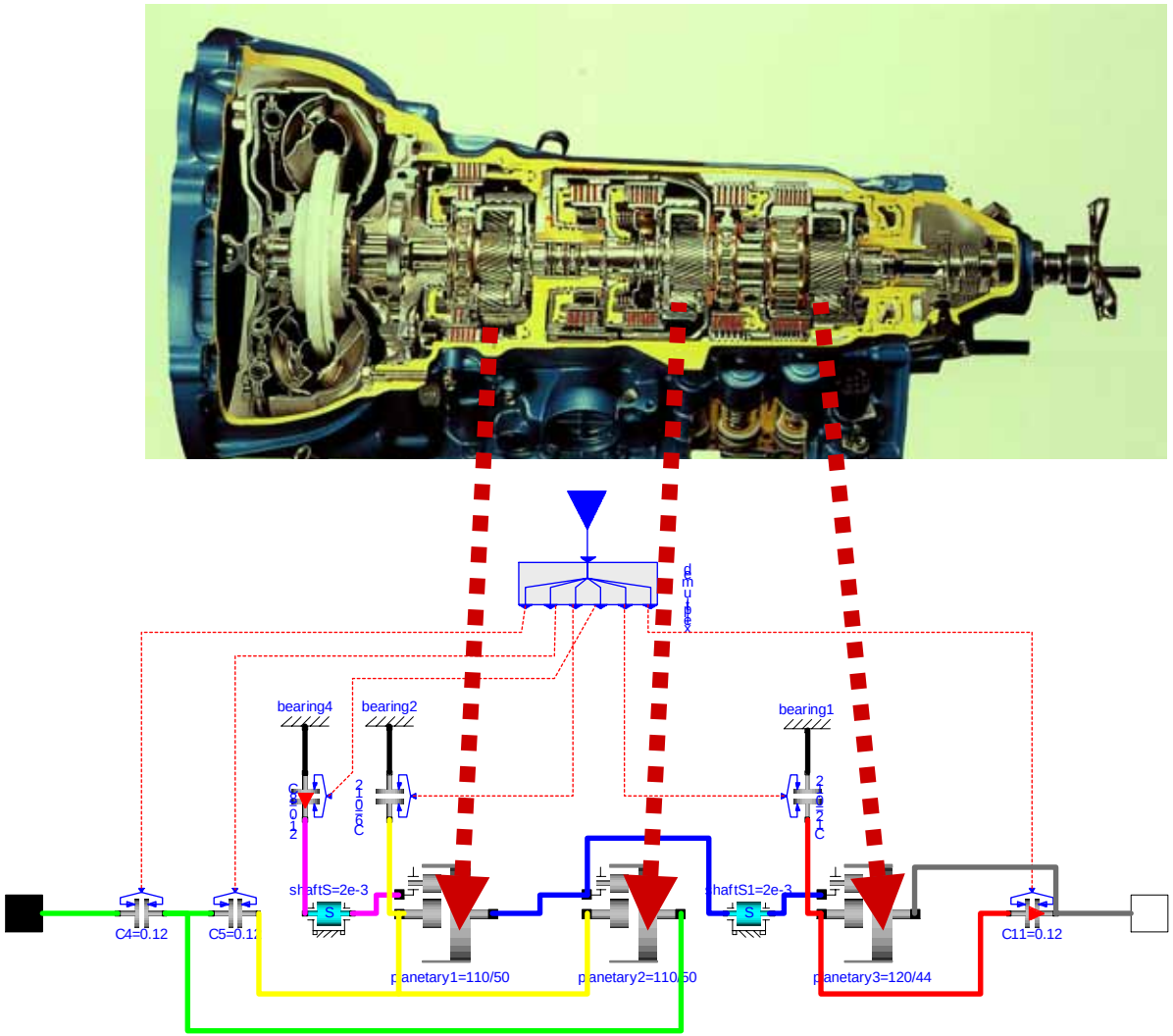
- Introduction
- Modelica
- ModelicaXML
- Compost
- ModelicaXML extension of Compost
- Modelica Component Model
- Composition programs
- Conclusions and Future Work

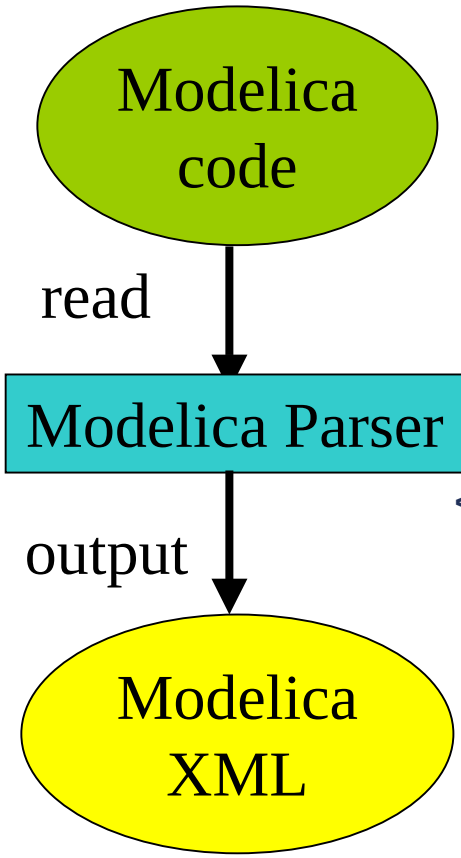
- **Why the need for Modelica composition and transformation?**
  - Interoperability between existing modeling languages or CAD tools and Modelica
  - Automatic generation of different version of models from product specifications. Choosing best design based on automatic simulation.
  - Automatic configuration of models using external sources (XML, databases, files)
  - Protection of intellectual property through obfuscation
  - Fine grain support for library developers

Modelica

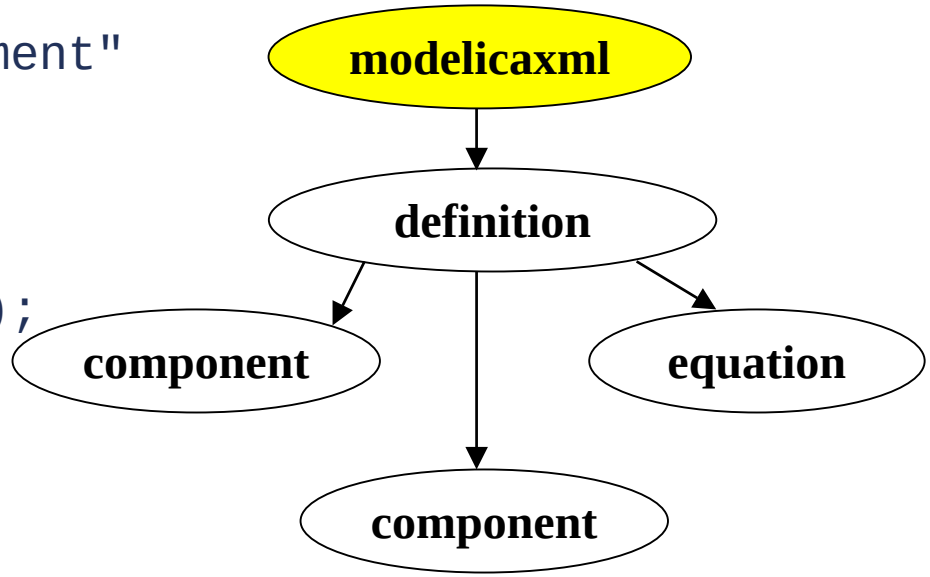
- *Declarative language*
  - Equations and mathematical functions allow acausal modeling, high level specification, increased correctness
- *Multi-domain modeling*
  - Combine electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc…
- *Everything is a class*
  - Strongly 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

Decomposition and Abstraction of an Automatic Gearbox



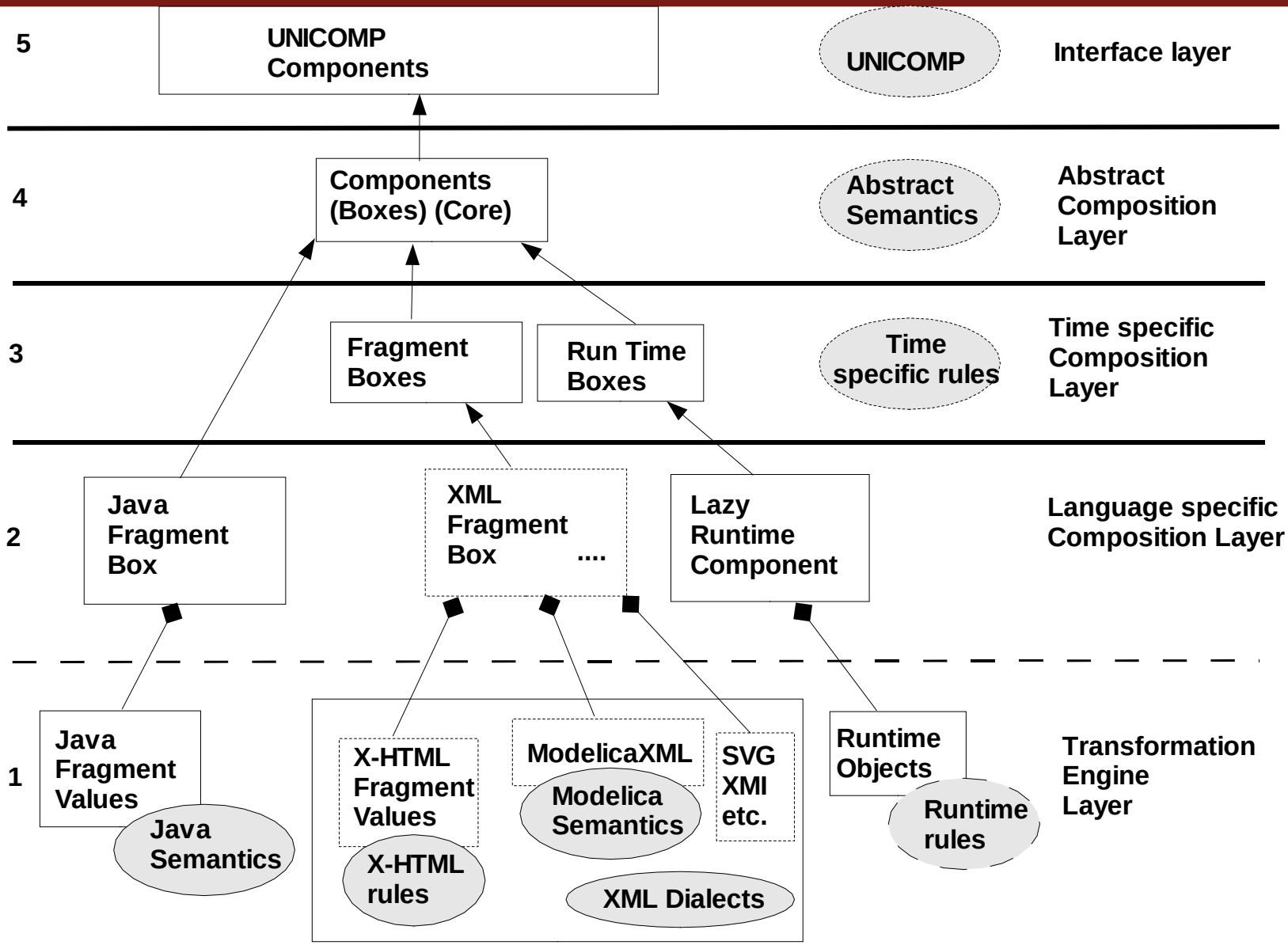



```
class Test "comment"
   Real x;
   Real xdot;
 equation
   xdot = der(x);
end Test;
```

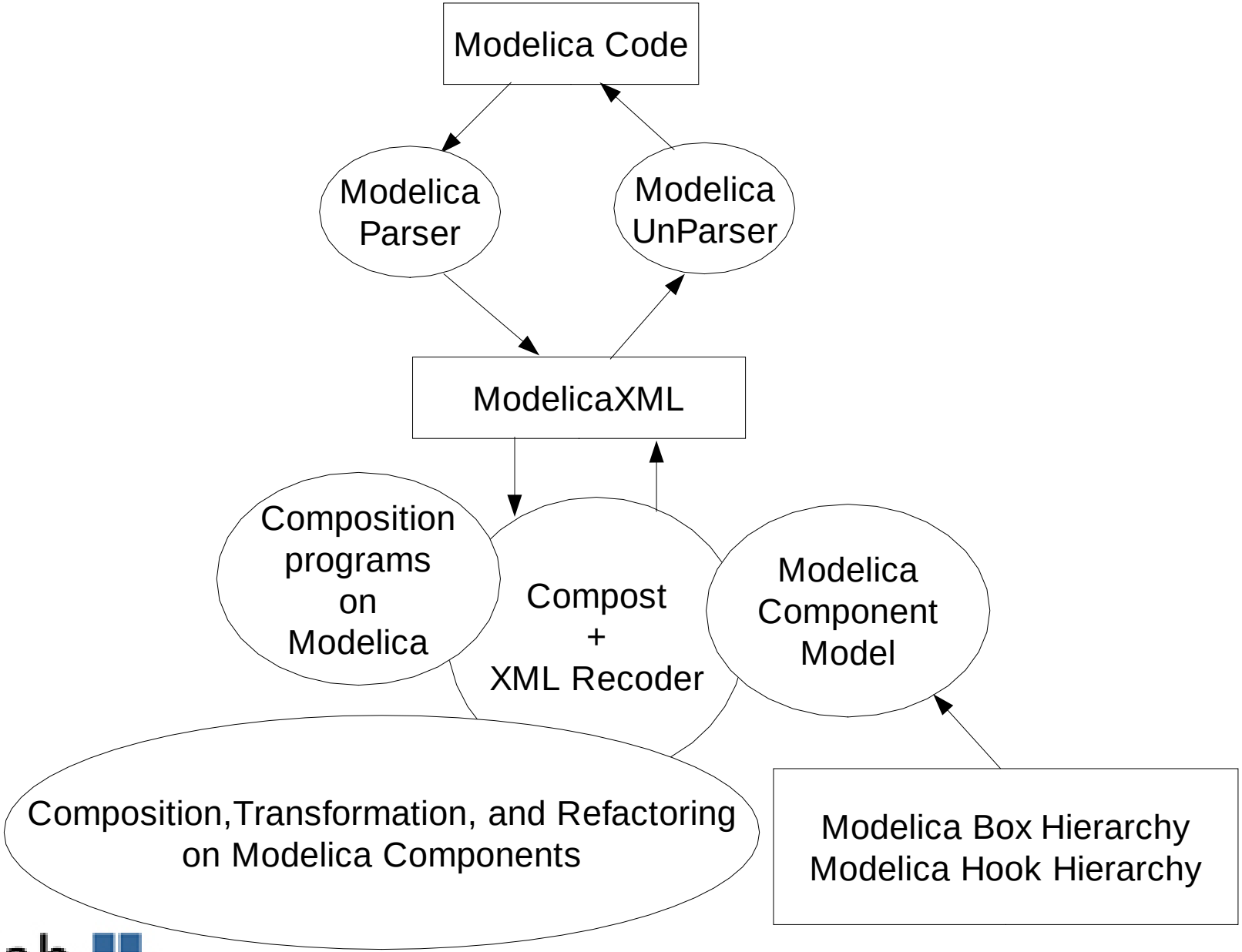



```
<modelicaxml>
  <definition ident= "Test"
              comment="comment">
    <component ident="x" type="Real"
               visibility="public" />
    <component ident="xdot" type="Real"
               visibility="public" />
    <equation>...</equation>
  </definition>
</modelicaxml>
```

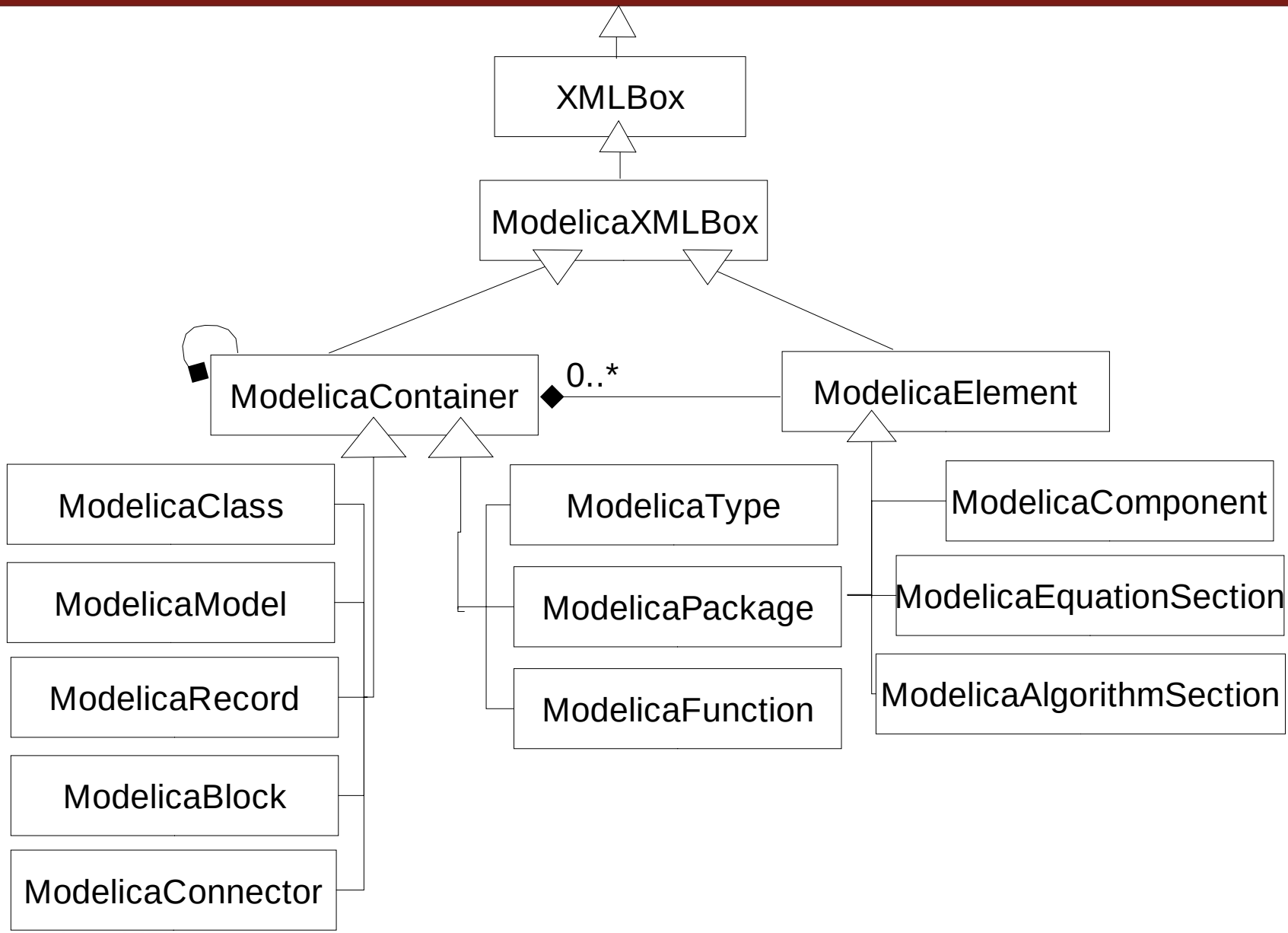
Compost Overview



ModelicaXML extension of Compost

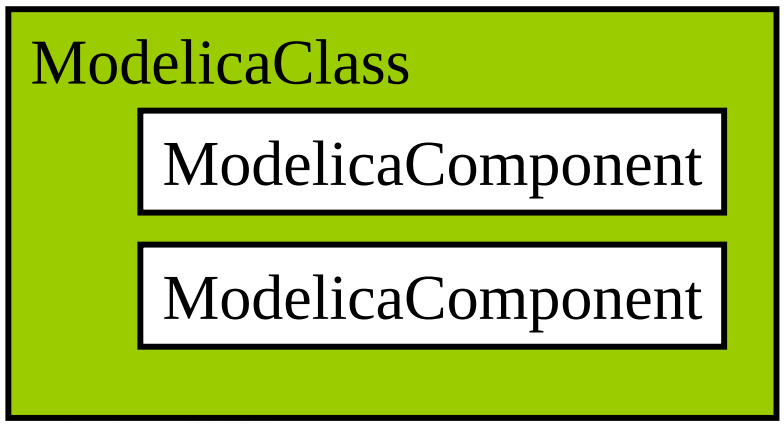


Modelica Component Model – Box hierarchy



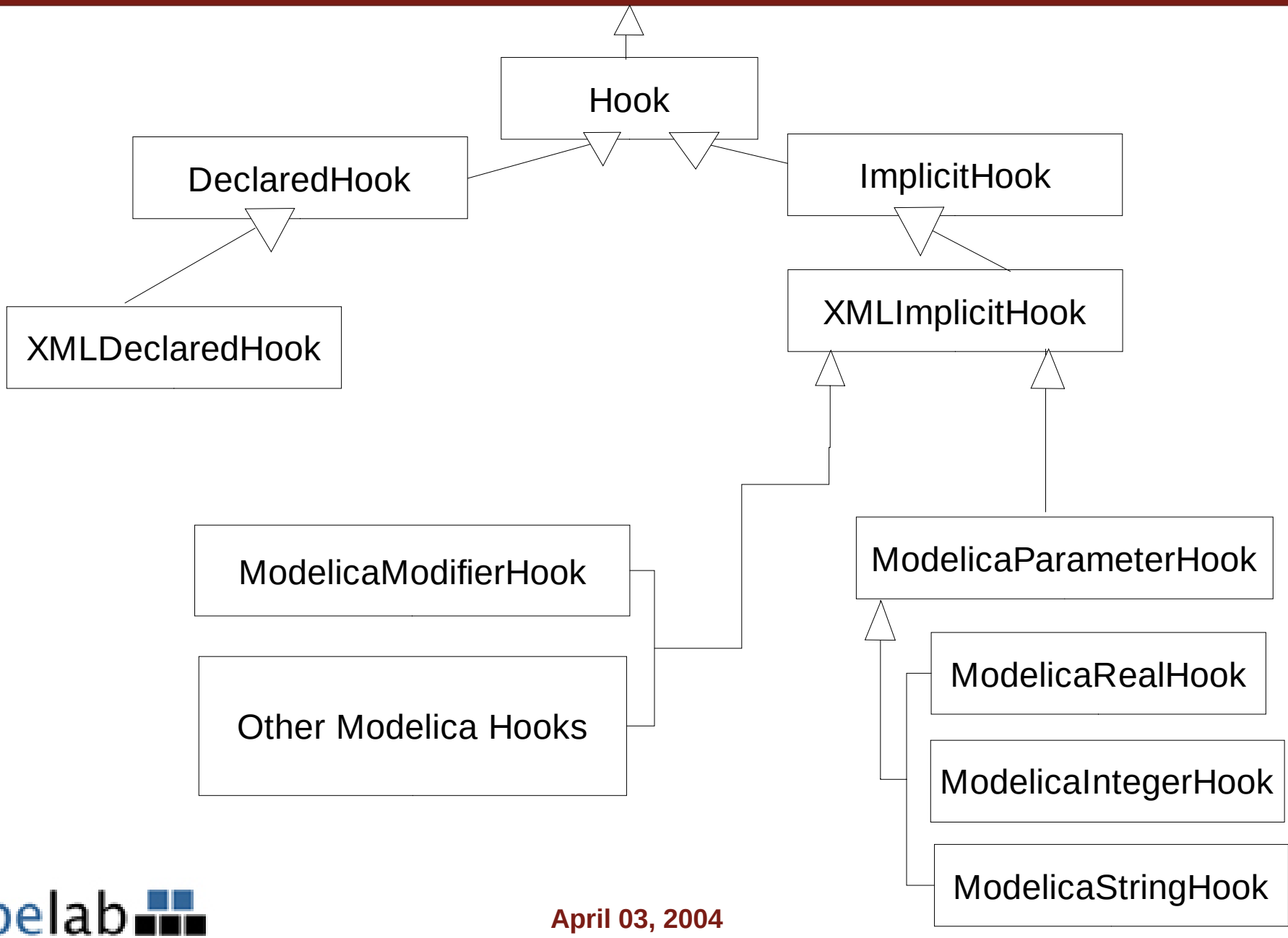
Example Box Hierarchy

```
<definition ident="Engine" restriction="class">
   <component visibility="public" variability="parameter"
              type="Integer" ident="cylinders">
      <modification_equals>
         <integer_literal value="4"/>
      </modification_equals>
   </component>
   <component visibility="public" type="Cylinder" ident="c">
      <array_subscripts>
         <component_reference ident="cylinders"/>
      </array_subscripts>
   </component>
</definition>
```



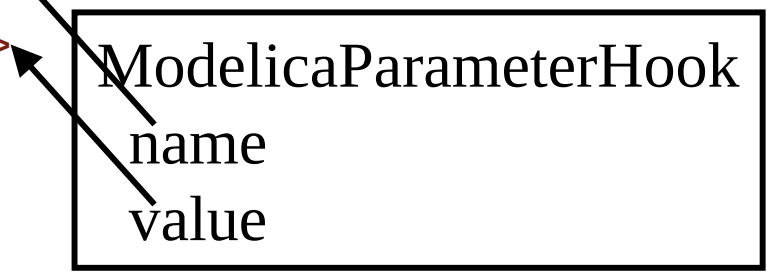

```
class Engine
   parameter Integer
                cylinders = 4;
   Cylinder c[cylinders];
end Engine;
```

Modelica Component Model – Hook hierarchy



Example: Hooks

```
<component visibility="public" variability="parameter"
           type="Integer" ident="cylinders">
   <modification_equals>
      <integer_literal value="4"/>
   </modification_equals>
</component>
```




```
parameter Integer
                cylinders = 4;
```

```
<definition ident="NewEngine" restriction="class">
   <extends type="Engine">
   ....
</definition>
```

```
class NewEngine
  extends Engine;
  ....
end NewEngine;
```

```
<definition ident="Engine" restriction="class">
  <extract>
    <component>..</component> ...
  </extract>
</definition>
```

Composition Programs: Mixin

```
ModelicaCompositionSystem cs =
     new ModelicaCompositionSystem();
ModelicaClass resultBox =
     cs.createModelicaClass(”Class1.mo.xml”);
ModelicaClass firstMixin =
     cs.createModelicaClass(”Class2.mo.xml”);
ModelicaClass secondBox =
     cs.createModelicaClass(”Result.mo.xml”);
   resultBox.mixin(firstMixin);
   resultBox.mixin(secondMixin);
   resultBox.print();
```

Composition Program: Input

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

Composition Program

```
ModelicaCompositionSystem
      cs = new ModelicaCompositionSystem();
ModelicaClass bodyBox =
     cs.createModelicaClass(``Body.mo.xml'');
ModelicaClass celestialBodyBox =
      cs.createModelicaClass(``Celestial.mo.xml'');
ModelicaElement extractedPart =
      celestialBody.findHook(``extract'').getValue();
   celestialBody.findHook(``extract'').bind(null);
   bodyBox.append(extractedPart); bodyBox.print();
   celestialBody.findHook(``superclass'').bind(``Body'');
   celestialBody.print();
```

Composition Programs: Result

```
class Body ``Generic Body''
    Real mass;
    String name;
end Body;
class CelestialBody ``Celestial Body''
   extends Body;
   constant Real g = 6.672e-11;
   parameter Real radius;
end CelestialBody;
```

Conclusions and Future Work

- ModelicaXML+Compost
  - a start in providing better tools for Modelica language
- Future Work
  - refine the Modelica+Compost extension
  - more composition operators
  - validation of composition using OpenModelica compiler
  - automatic generation of Modelica component model from the Modelica ontology

Thank you!

Questions?

ModelicaXML and Compost

- ModelicaXML
  - http://www.ida.liu.se/~adrpo/modelica/xml
- Compost
  - http://www.the-compost-system.org/

Components

Transformation and Composition

Sagrada Familia