



National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

SysML-Modelica: JPL Implementation Overview & Specification-related Issues

Alek Kerzhner and Nicolas Rouquette
Jet Propulsion Laboratory, Caltech

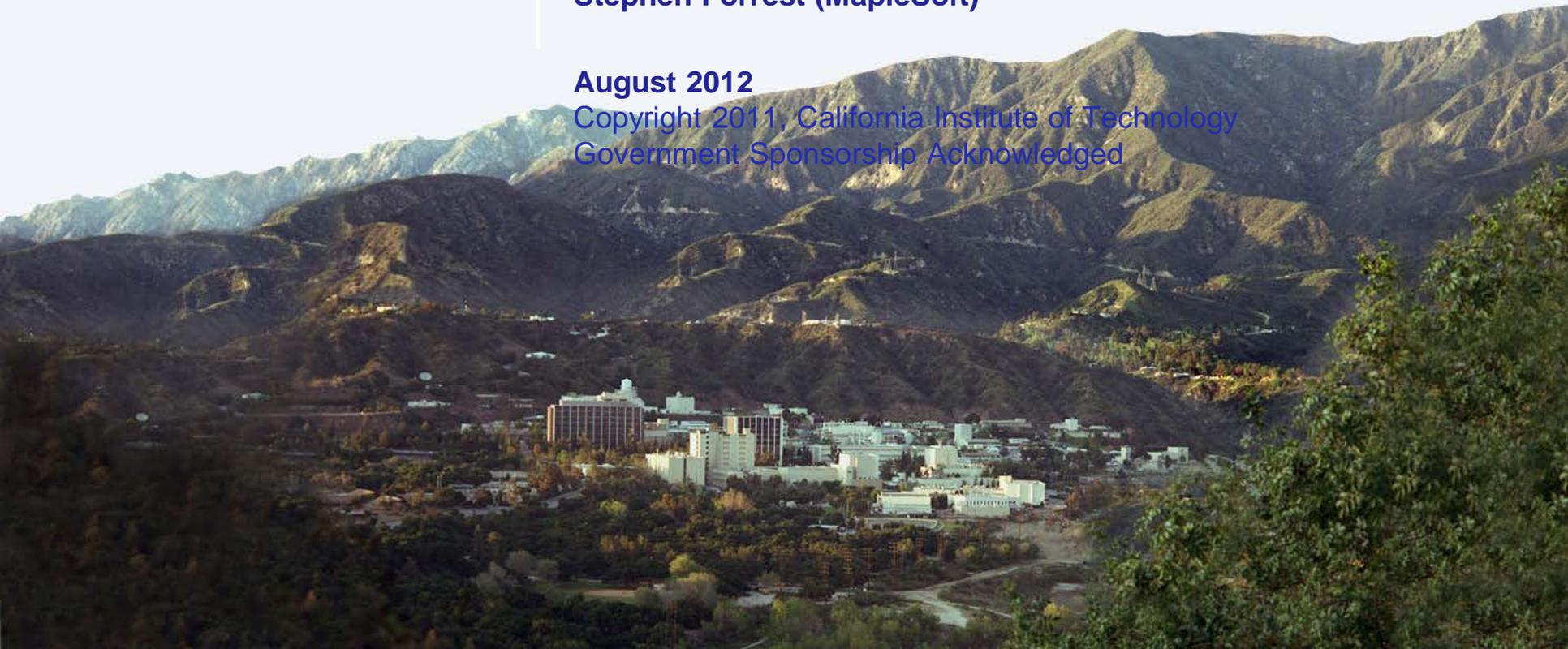
With the help from:

Sebastian Herzig (Georgia Inst. Of Technology)

Stephen Forrest (MapleSoft)

August 2012

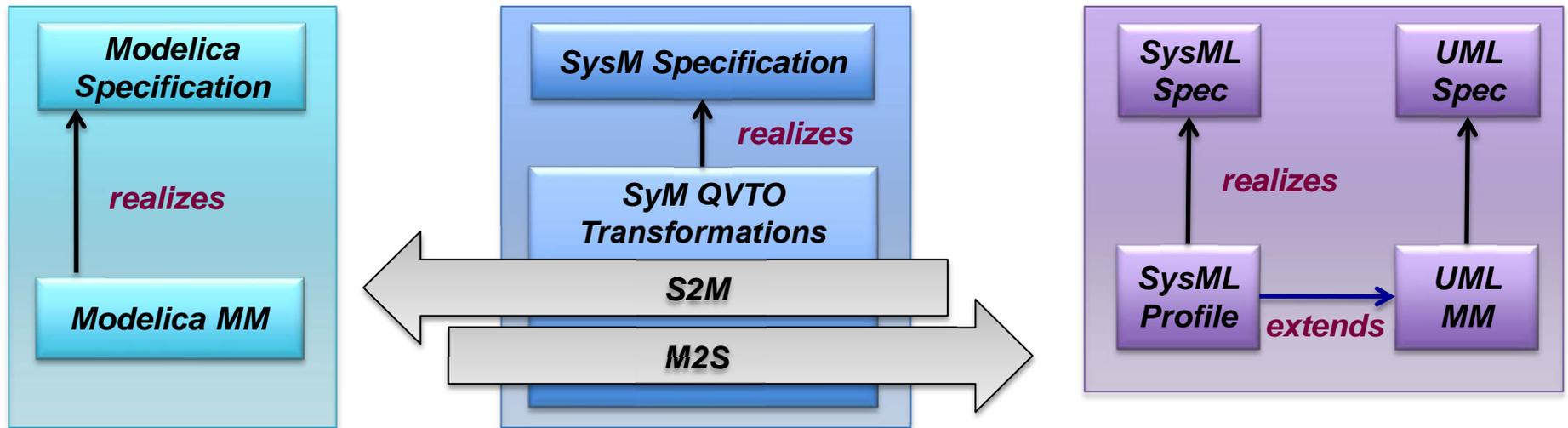
Copyright 2011, California Institute of Technology
Government Sponsorship Acknowledged





SysML / Modelica @ JPL

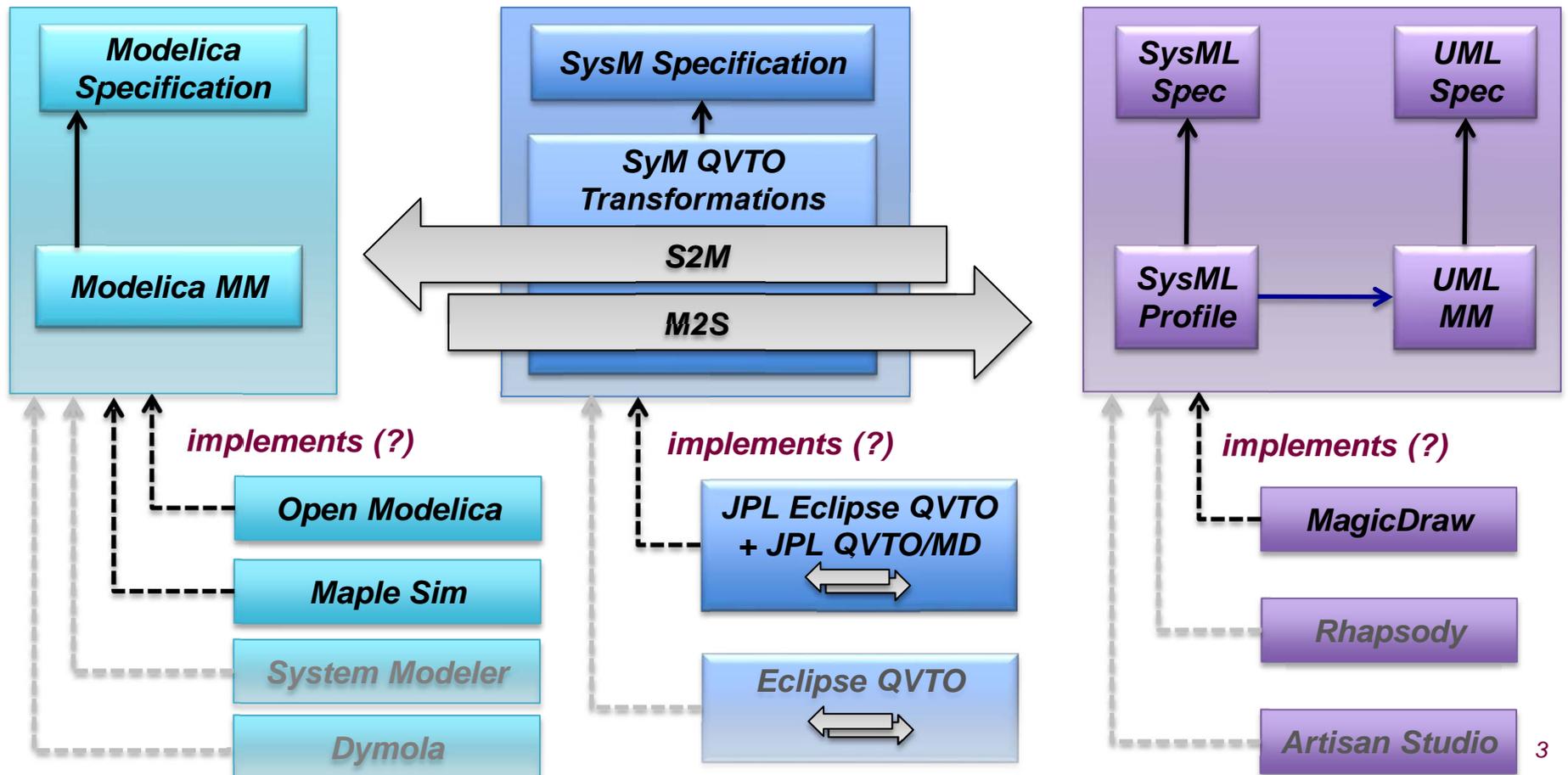
- Why?
 - Promote model-based systems engineering practices combining:
 - analytical modeling in Modelica
 - descriptive modeling in SysML
 - The OMG SyM specification is existence proof it can be done
 - Long term: specifications are cheaper than custom technology development
 - Short term: specification have bugs (just like any other kind of technology)





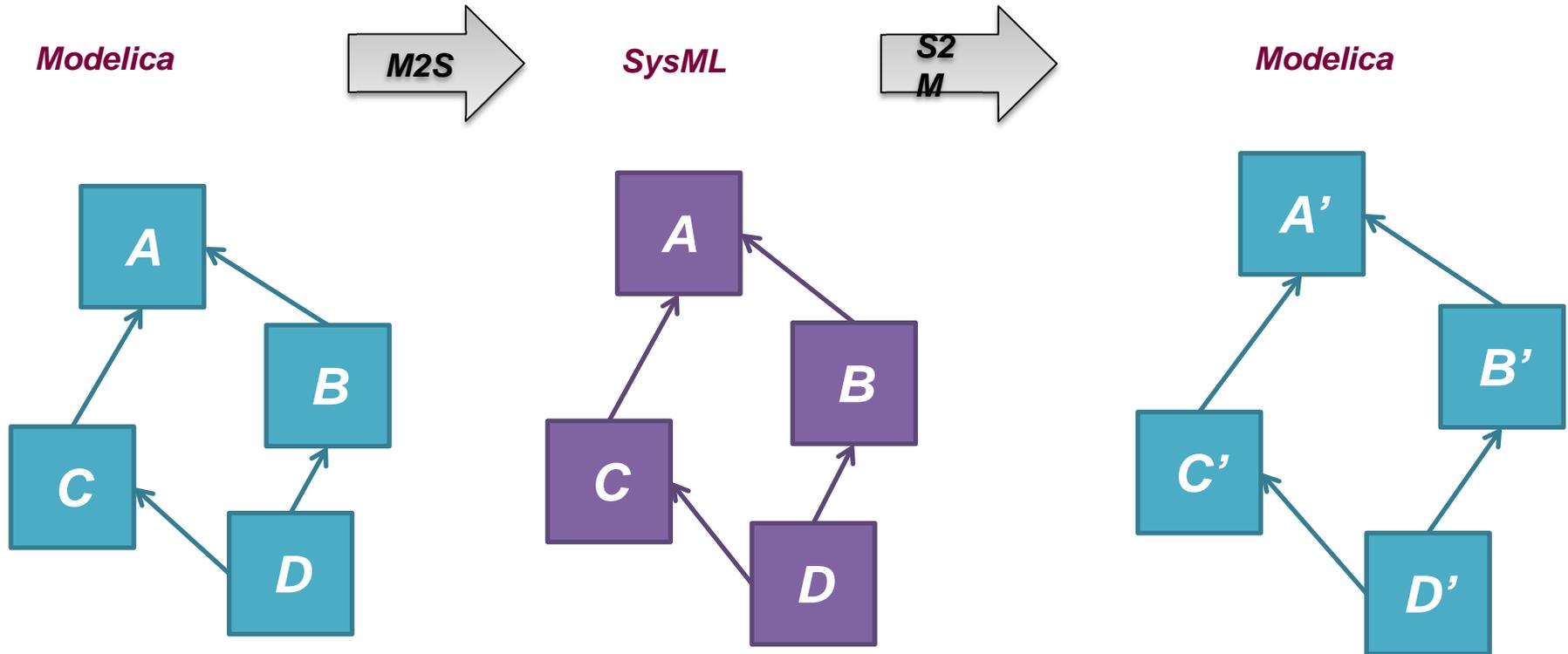
Specifications are practically useful

- Specification record the history of technology evolution & understanding
 - Long term: specifications help guide future technology development
 - Short term: specification help identify gaps in selected technologies (w.r.t some criteria)





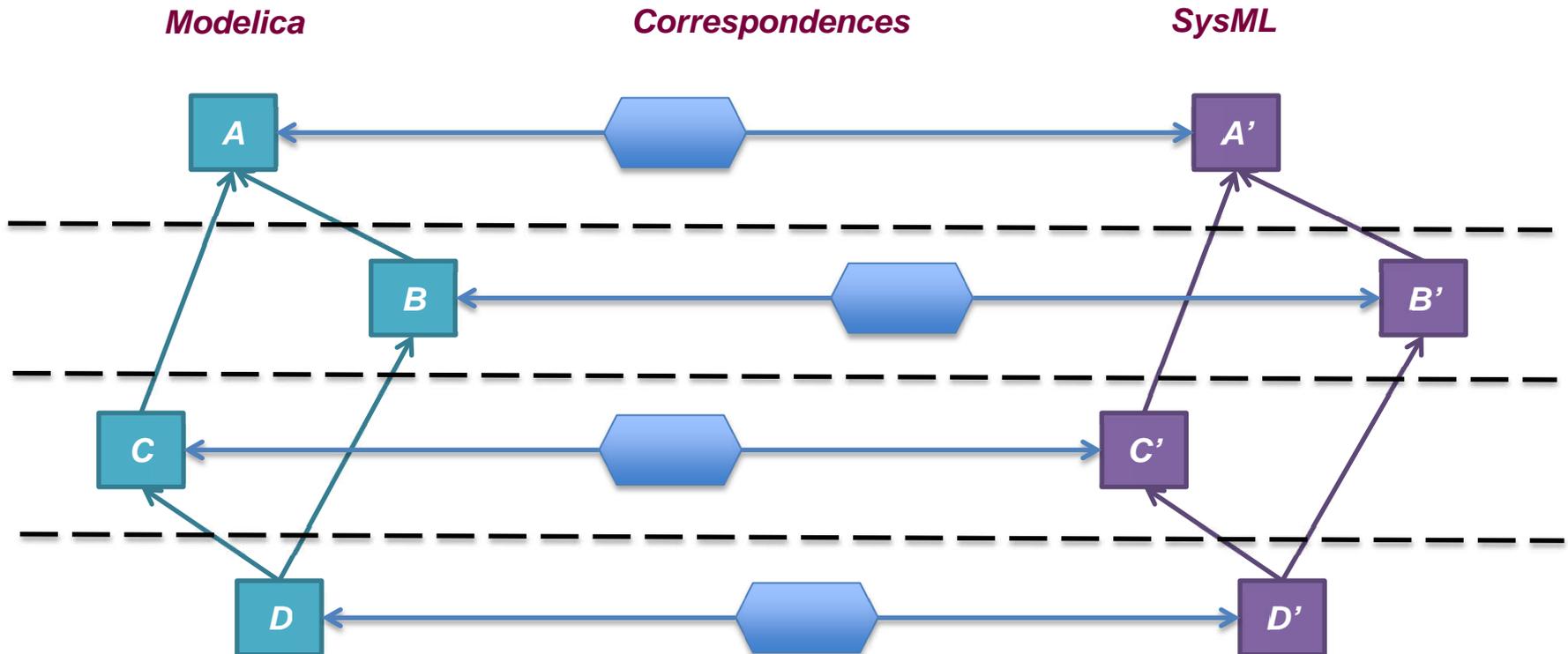
Original Implementation Approach



- If we apply M2S and S2M, do we get the “*same thing*” as the original?
- The original spec is written to map everything
 - In practice, the modelica models are very large; only a subset of the information is relevant and useful to map to SysML;
 - Practical implementations of M2S + S2M will likely lose some information



Incremental Approach with Correspondences

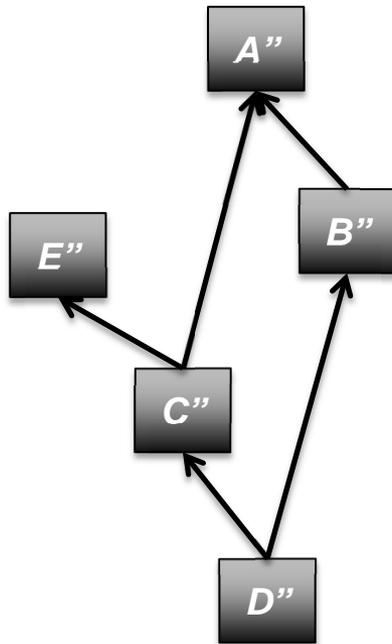


- The Modelica Standard Library is **very** large (and growing!)
- In practice, it is necessary to reuse prior Modelica/SysML mappings
- Correspondence records of Modelica/SysML mappings facilitate reuse



Modelica Semantics Matters

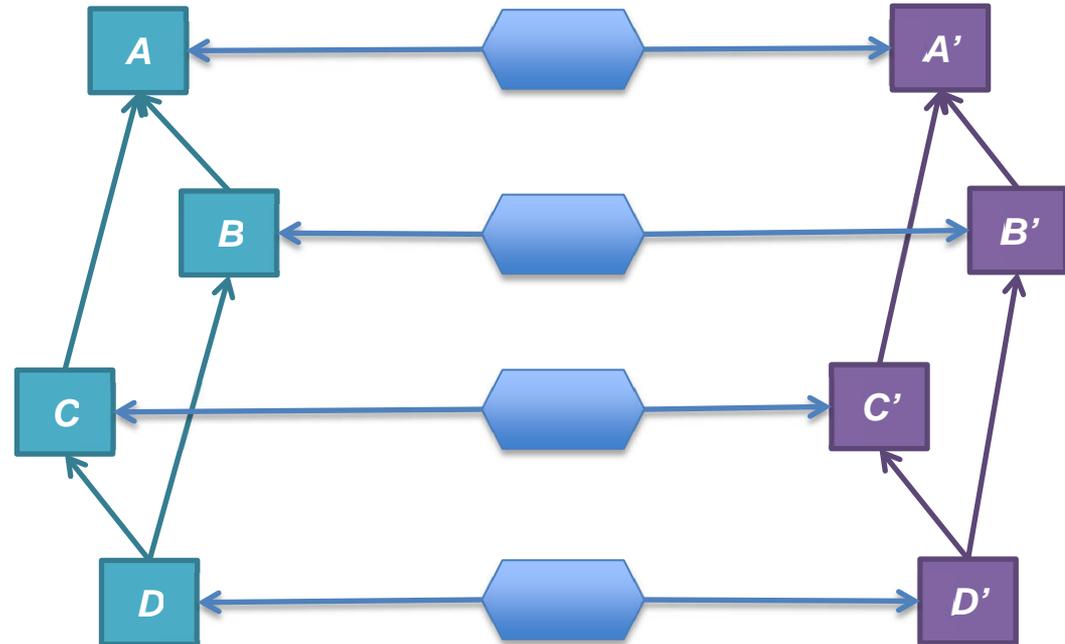
**Compiler, Simulator,
Code Generator, ...**



Modelica

Correspondences

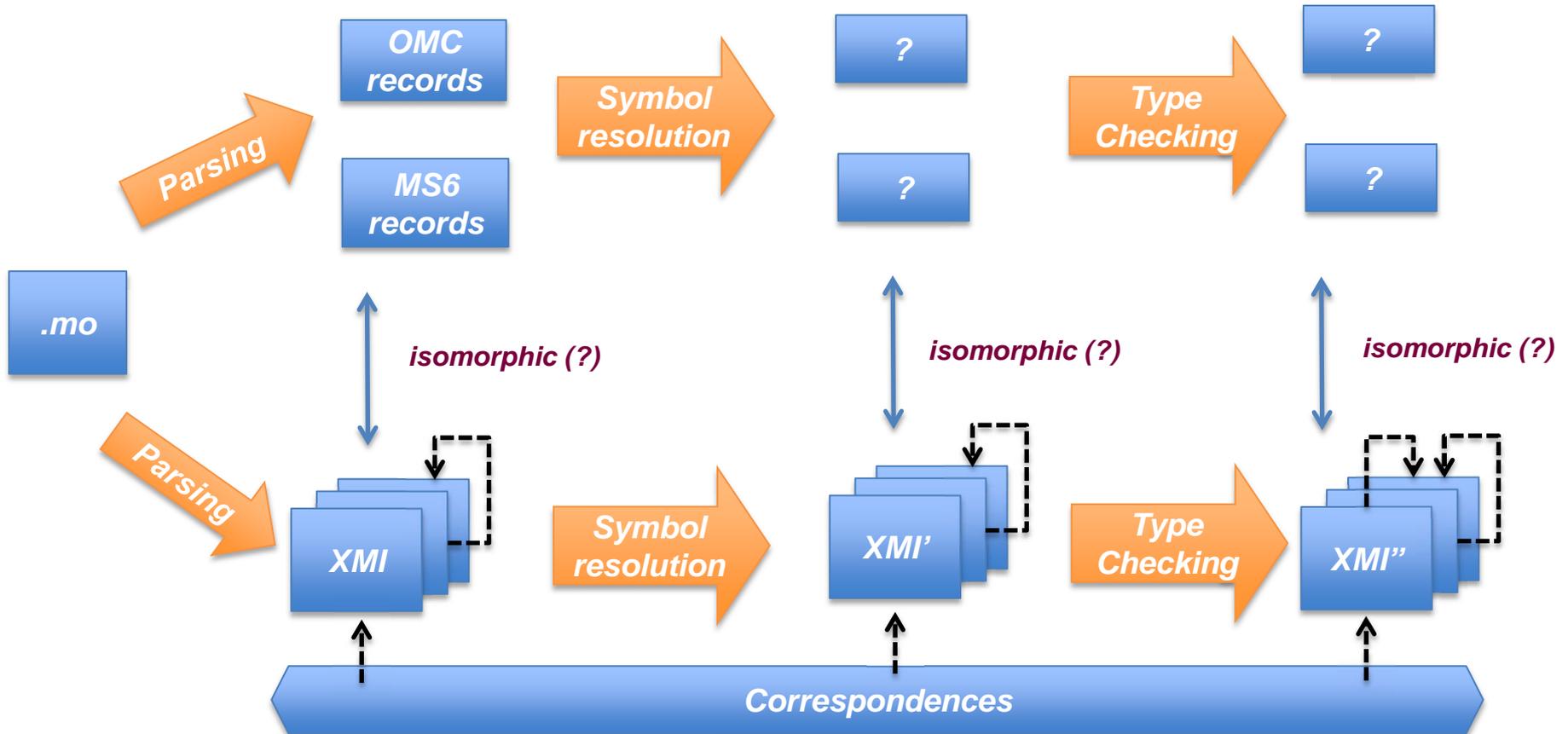
SysML



- In practice, the semantics of the Modelica language becomes “visible” in terms of, e.g., the simulator code that Modelica tools generate
- All modelica-compliant tools must agree on the meaning of source models (e.g., A, B, C, D) – e.g., avoid the situation where we have partial equivalence for some classes (A, B) but not others (C, D)



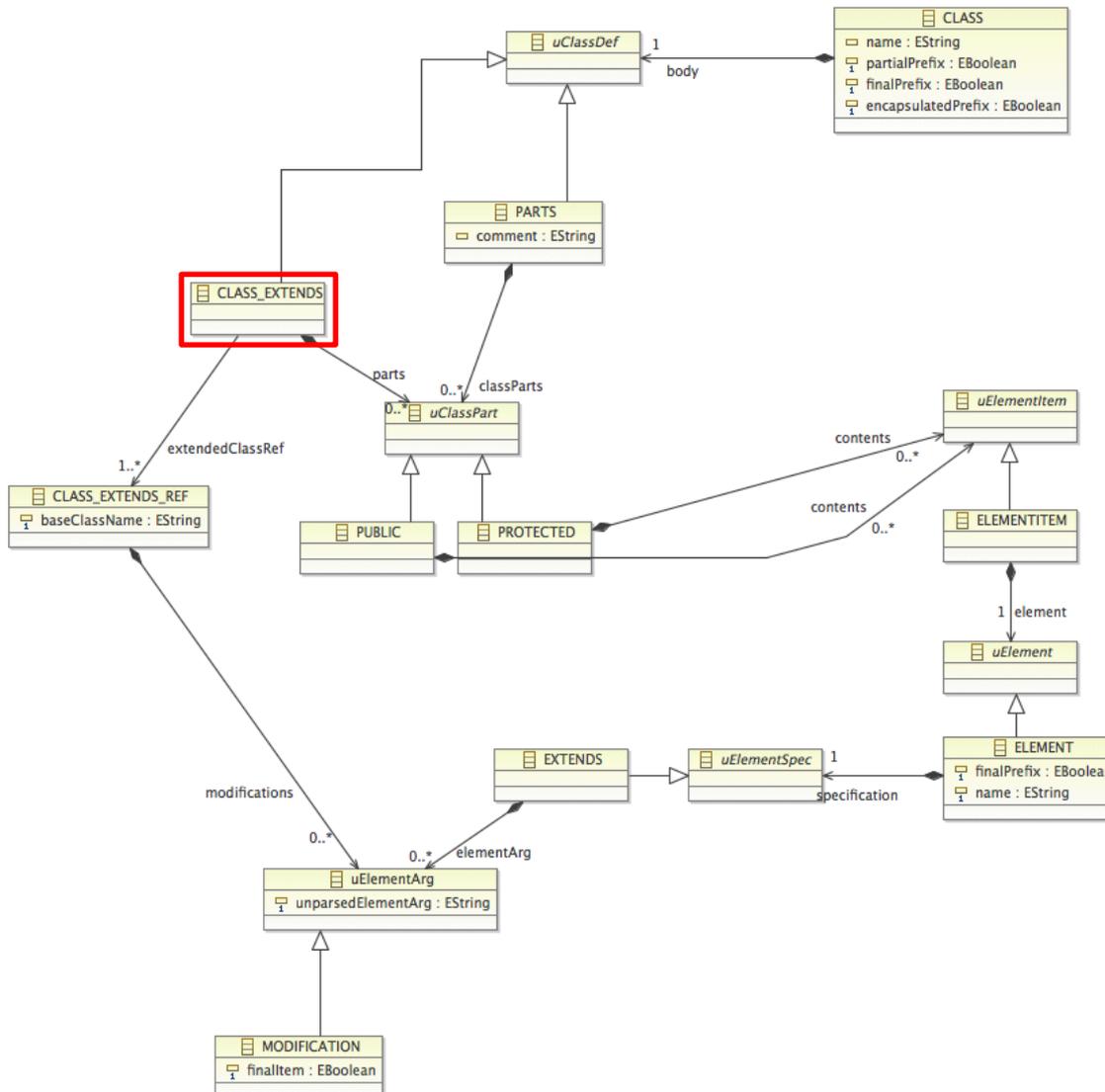
JPL's Compiler-style Approach: Simplifying a complex problem into separate, modular concerns



- Parsing: constructing the abstract syntax (graph) from the concrete syntax (text)
- Symbol resolution: mapping symbols to locations in the abstract syntax
- Type checking: verifying well-formedness of the abstract syntax



Parsing: Metamodel Issues



- The SyM specification currently refers to the CLASS_EXTENDS construct
 - TODO: add a reference
- The EXTENDS construct is more generic because it can also be used to capture the PUBLIC/PROTECTED aspect of an extends statement



Parsing: Ambiguities about which Abstract Syntax Representation is “correct”

```
package TestCasesReplaceable
```

```
model BaseCorrelation
```

```
  Real x;
```

```
  Real y;
```

```
end BaseCorrelation;
```

```
model SpecialCorrelation2
```

```
extends BaseCorrelation;
```

```
equation
```

```
   $y = x / 3;$ 
```

```
end SpecialCorrelation2;
```

```
model UseCorrelation
```

```
  replaceable model Correlation = SpecialCorrelation;
```

```
  Correlation correlation;
```

```
equation
```

```
  correlation.y = time;
```

```
  y = correlation.x;
```

```
end UseCorrelation;
```

```
model UseCorrelation2
```

```
  extends UseCorrelation(redeclare model Correlation = SpecialCorrelation2);
```

```
end UseCorrelation2;
```

```
end TestCasesReplaceable;
```



Symbol Resolution: What are the rules for the Modelica language?

```
package TestCasesSpc72
```

```
model C
```

```
parameter Real x = 2;
```

```
replaceable package Medium = Modelica.Media.IdealGases.SingleGases.H2O;
```

```
B b(x = x, redeclare package Medium = Medium);
```

```
end C;
```

```
model D
```

```
parameter Real x = 3;
```

```
package Medium = Modelica.Media.IdealGases.SingleGases.O2;
```

```
C c(b(x = x, redeclare package Medium = Medium));
```

```
end D;
```

```
model B
```

```
parameter Real x;
```

```
Medium.ThermodynamicState state(p = 200000, T = 500);
```

```
replaceable package Medium = Modelica.Media.IdealGases.SingleGases.H2O;
```

```
Medium.SpecificHeatCapacity cp = Medium.specificHeatCapacityCp(state);
```

```
Modelica.Blocks.Interfaces.RealOutput y = cp;
```

```
end B;
```

```
end TestCasesSpc72;
```

- Currently, the symbol resolution rules for Modelica are specified in English prose across the Modelica Language Reference
- Practical problems: How many rules are there? What do they mean?



Type Checking: Does the SyM transformation mapping make sense?

- Where does the modelica type checking take place?
 - With .mo development, it's the .mo compiler – OK
 - With SysML to Modelica transformation, it may be impractical and difficult for end users to understand errors found in the .mo code that's been converted from .xmi that's been generated by transformation...
 - Obviously, it's not necessary to do full type checking on SysML4Modelica-profiled models but clearly there is a minimum of type checking that will be practically very useful for end users (and expected!)
- Where are the rules for type checking Modelica specified?
 - Same problem as previously described for symbol resolution rules