Growing an Equation-Based Object-Oriented Modeling Language

4th MODPROD Workshop on Model-Based Product Development

February 10, 2010

David Broman

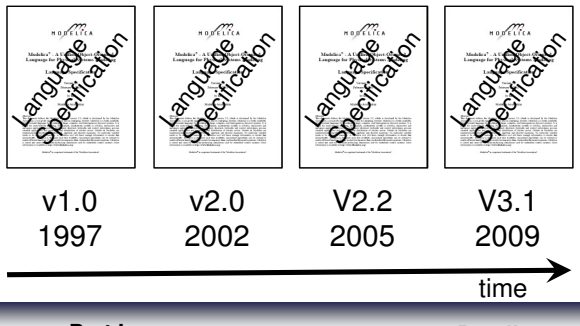
Department of Computer and Information Science
Linköping University, Sweden
davbr@ida.liu.se

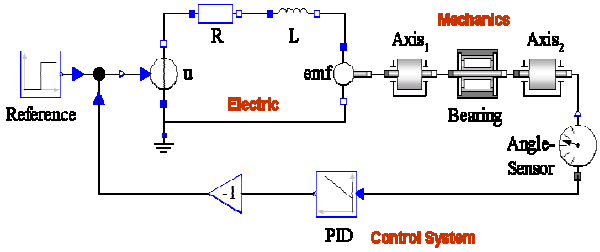
Linköpings universitet

Problem

MODELICA

- Differential Algebraic Equations
- Object-Oriented concepts
- Multi-domain modeling
- Declarative acausal modeling
- Hybrid modeling





Problems

- Large and complex languages
  →hard to implement
- Many constructs -
  →hard to reason about
- Backwards compatibility

How should one design a modeling language over time?

Contribution

How should you grow an equation-based object-oriented language, such as Modelica, in a sound manner?

Main contributions of this work

- A systematic way of categorizing different ways of growth of an equation-based object-oriented language.
- Analyzed tradeoffs from different stakeholder's perspective

Agenda

Part I
Introduction and concepts of syntax and semantics

Part II
Different ways of growing a language

Part III
The right way to grow

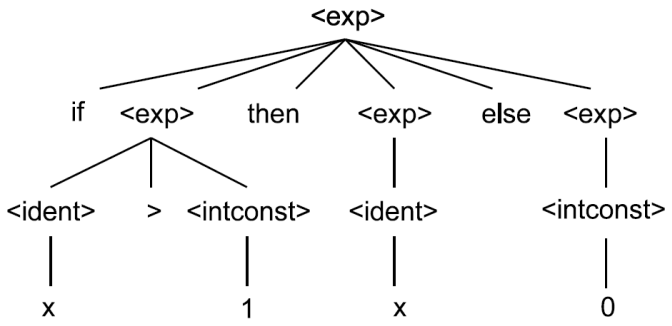
Part I

The concepts of syntax and semantics

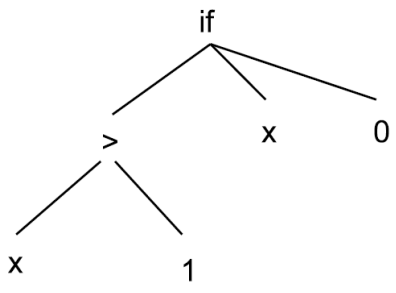
What does *syntax* of a language mean?

The syntax describes the structure of a model

```
if x > 1 then x else 0
```



Parse Tree (Concrete Syntax)



Abstract Syntax Tree (AST)

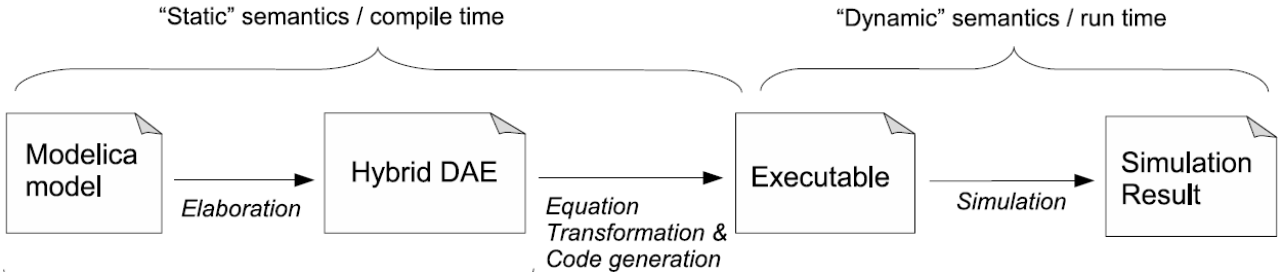
What does *semantics* of a language mean?

The semantics describes the meaning of a model

```
if x > 1 then x else 0
```

Different syntax, but could have the same semantics, e.g. meaning

```
if(x > 1){return x;} else {return 0;}
```



Part II

Different ways of growing a language

Ways of growth matrix

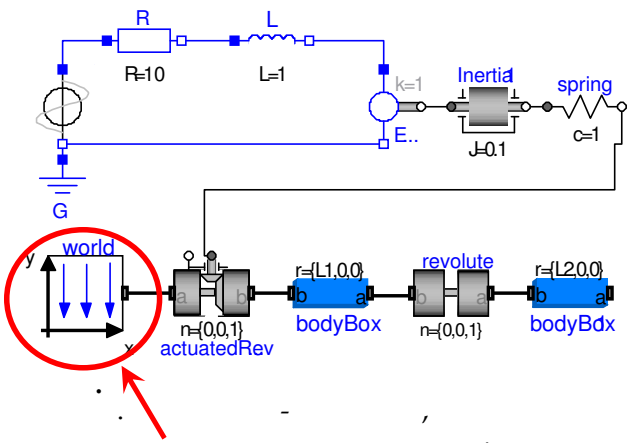
Extending the Syntax \ Extending the Semantics	yes	no
yes	*"growth by adding new language features"*	*"growth by adding syntactic sugar"*
no	*"growth by new meanings of annotations or built-in functions"*	*"growth by new user defined abstractions"*

Extending syntax and semantics

Growth by adding new language features

New syntax
Inner and outer

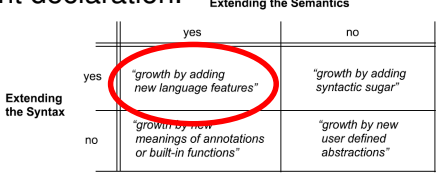
```
model M                 model N
  outer Real x;           inner Real x;
  ...                     M m1, m2;
end M;                    ...
                        end N;
```



Compontents need to have access to other compents in the instance hierarchy

New semantics...

"An element declared with the prefix outer references an element instance with the same name but using the prefix inner which is nearest in the enclosing instance hierarchy of the outer element declaration."



Ways of growth matrix

Extending the Syntax \ Extending the Semantics	yes	no
yes	*"growth by adding new language features"*	*"growth by adding syntactic sugar"*
no	*"growth by new meanings of annotations or built-in functions"*	*"growth by new user defined abstractions"*

Extending only the syntax

Growth by adding syntactic sugar

```
model M1a
  Real x(start=5);
equation
  der(x) = -x + 2;
end M1a;
```

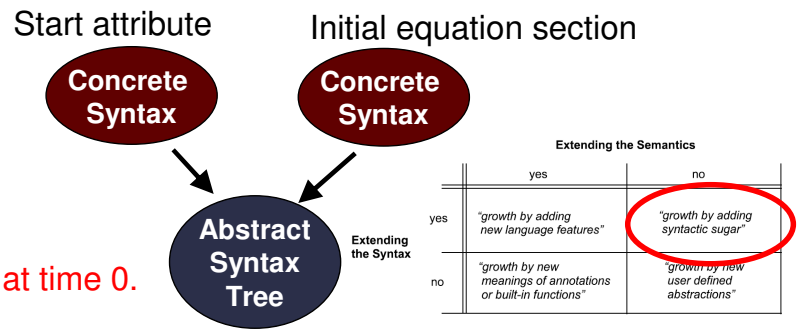
```
model M1b
  Real x;
equation
  der(x) = -x + 2;
initial equation
  x = 5;
end M1b;
```

```
model M1c
  Real x(start = 5,
         fixed=true);
equation
  der(x) = -x + 2;
end M1c;
```

```
model M1d
  Real x(fixed =
           true);
equation
  der(x) = -x + 2;
initial equation
  x = 5;
end M1d;
```

Do not compile. Over-determined x=0 and x=5 at time 0.

Are they all describing the same model?



Ways of growth matrix

Extending the Syntax \ Extending the Semantics	yes	no
yes	*"growth by adding new language features"*	*"growth by adding syntactic sugar"*
no	*"growth by new meanings of annotations or built-in functions"*	*"growth by new user defined abstractions"*

Extending only the semantics

Growth by new meanings of annotations and built-in functions

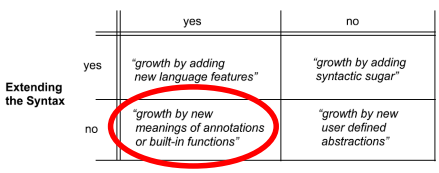
New built-in functions

E.g. cos(x), floor(x), delay(expr, delaytime) etc.

Annotations

Store extra inforamation about models, e.g., graphics, version info, documentation, etc.

```
annotation (
  Icon(coordinateSystem(extent={{-100,-100}, {100,100}}),
       graphics={__NameOfVendor(Circle(center={0,0}, radius=10))})
);
```



Ways of growth matrix

Extending the Syntax \ Extending the Semantics	yes	no
yes	*"growth by adding new language features"*	*"growth by adding syntactic sugar"*
no	*"growth by new meanings of annotations or built-in functions"*	*"growth by new user defined abstractions"*

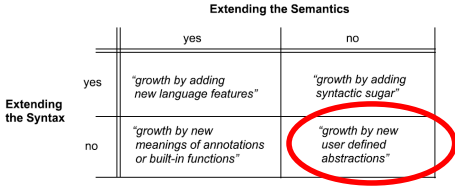
Extending neither syntax nor semantics

Growth by new user defined abstractions

User defined abstractions
Functions, models, blocks etc. Encapsulated into reusable **libraries**

Neither changing the *language* definition nor the *tool* implementation.

Important!
New abstractions should look like primitive constructs in the langauge



Ways of growth matrix

Extending the Syntax	Extending the Semantics: yes	Extending the Semantics: no
yes	*"growth by adding new language features"*	*"growth by adding syntactic sugar"*
no	*"growth by new meanings of annotations or built-in functions"*	*"growth by new user defined abstractions"*

Restricting the language (syntax, semantics or both)

Part III

The right way to grow

Different stakeholders' perspective

Large group – continuously adding features

Handle everything – what to leave out?

Easy to use and understand

Clear semantics, e.g. Initial equations

Small group – kernel, syntactic sugar, missing user input

Language Designers

End Users

Detect and locate errors. Restrictions necessary.

Expressiveness...

Library Users

Tool Vendor

Ideal, grow without changing syntax or semantics

Interpret the specification unambiguously.

Commercial perspective. Conflict of interest with language design. Not always want to be 100% compatible.

Concluding Summary

Minimize or avoid, especially with mature and widely used languages

Most drastic change

A preferable approach. Clearer semantics for users and easier to implement correctly for tool vendors.

Library developers are pushing, language designers and tool vendors negatively affected

Extending the Semantics

Extending the Syntax	yes	no
yes	*"growth by adding new language features"*	*"growth by adding syntactic sugar"*
no	*"growth by new meanings of annotations or built-in functions"*	*"growth by new user defined abstractions"*

Hard to execute in a large design group

Looks initially attractive. Few changes in specification

The preferable approach.

Dangerous. Incompatible models which are tool dependent.

Very hard for young and new kind of languages, e.g. EOO language Modelica

More details…

David Broman. **Growing an Equation-Based Object-Oriented Modeling Language**. *In Proceedings of MATHMOD 09 Vienna*, pages 1316-1324, Vienna, Austria, 2009.

Available from: www.ida.liu.se/~davbr

Thanks for listening!