Higher-Order Acausal Models

2nd International Workshop on Equation-Based Object-Oriented Languages and Tools (EOOLT)

Paphos, Cyprus, July 8, 2008

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

Peter Fritzson
Department of Computer and Information Science
Linköping University, Sweden
petfr@ida.liu.se

Linköpings universitet

The idea of Higher-Order Acausal Models...

David Broman
davbr@ida.liu.se

Higher-Order Functions

I.e. first class citizens, can be passed around as any value

+

Acausal Models

Models in EOO languages, composing DAEs and other interconnected models.

=

Higher-Order Acausal Models

I.e., first class acausal models.

Modeling Kernel Language (MKL)

David Broman
davbr@ida.liu.se

Modeling Kernel Language (MKL)

- A research language with similar modeling capabilities as a subset of the Modelica language.
- Primarily aimed at investigating novel language construct.
- An formal operation semantics of the dynamic elaboration process exists. (Broman, 2007, Tech. Report “Flow Lambda Calculus for Declarative Physical Connection Semantics”)

Here we will use MKL to demonstrate the concept of HOAMs, but...

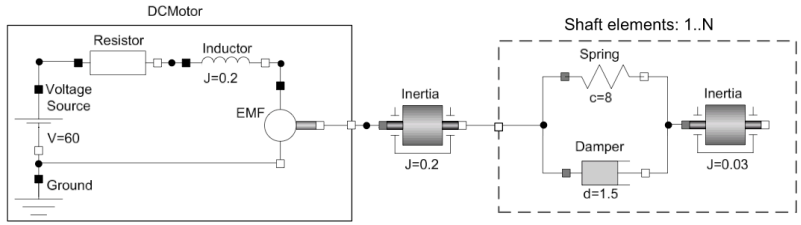
- ...the concept is not limited to this language and can of course be considered in other languages as well...

David Broman
davbr@ida.liu.se

Agenda

Part I
The Basic Idea of Higher-Order

Part II
Higher-Order Modeling in MKL



Part III
Related Work and Future Perspective

Part I
The Basic Idea of Higher-Order

Part II
Higher-Order Modeling in MKL

Part III
Related Work and Future Perspective

Linköpings universitet

David Broman
davbr@ida.liu.se

Part I

The Basic Idea of Higher-Order

What is an Anonymous Function?

David Broman
davbr@ida.liu.se

Anonymous functions (lambda abstractions) exist in ordinary functional languages (e.g. SML, Haskell, LISP etc.)

An anonymous function do not have a defined name.

```
func(x){x*x}
```

Parameters within parathesis. Function body.

An anonymous function can then be applied to an argument.

```
func(x){x*x}(3)
→ 3*3
→ 9
```

Evaluation steps

Anonymous functions are treated as values. It is convenient to give them names.

```
def pi = 3.14
def power2 = func(x){x*x}
```

The named values can then be used in a new expressen.

```
power2(pi)
→ power2(3.14)
→ 3.14 * 3.14
→ 9.8596
```

What is a Higher-Order Function? (1/3)

David Broman
davbr@ida.liu.se

DEFINITION 1 (Higher-Order Function).
A higher-order function is a function that

1. takes another function as argument, and/or
2. returns a function as the result.

Also, a higher-order function is said to be first-class citizens, e.g. the function is treated as a value and can be passed around freely.

What is a Higher-Order Function? (2/3)

David Broman
davbr@ida.liu.se

DEFINITION 1 (Higher-Order Function).
A higher-order function is a function that
1. takes another function as argument, and/or
2. returns a function as the result.

Define a function twice with a function parameter f.

```
def twice = func(f,y){
    f(f(y))
};
```

Apply twice to power2 and constant 3.

```
twice(power2,3)
→ power2(power2(3))
→ power2(3*3)
→ power2(9)
→ 9*9
→ 81
```

We can also have an anonymous function as argument.

```
twice(func(x){2*x-3},5)
→ func(x){2*x-3}(func(x){2*x-3}(5))
→ func(x){2*x-3}(2*5-3)
→ func(x){2*x-3}(7)
→ 2*7-3
→ 11
```

What is a Higher-Order Function? (3/3)

David Broman
davbr@ida.liu.se

DEFINITION 1 (Higher-Order Function).
A higher-order function is a function that

1. *takes another function as argument, and/or*
2. *returns a function as the result.*

In mathematics, functional composition is normally expressed using the infix operator $\circ$. Two functions $f : X \to Y$ and $g : Y \to Z$ can be composed to $g \circ f : X \to Z$, by using the definition $(g \circ f)(x) = g(f(x))$.

The same definition can be given as a higher-order function:

```
def compose = func(g,f){
    func(x){g(f(x))}
};
```

The compose function can then be used as follows:

```
def add7 = func(x){7+x};

def foo = compose(power2,add7);
→ def foo = func(x){power2(add7(x))};

foo(4)
→ func(x){power2(add7(x))}(4)
→ power2(add7(4))
→ power2(7+4)
→ power2(11)
→ 11*11
→ 121
```

Elaboration and Simulation of Acausal Models

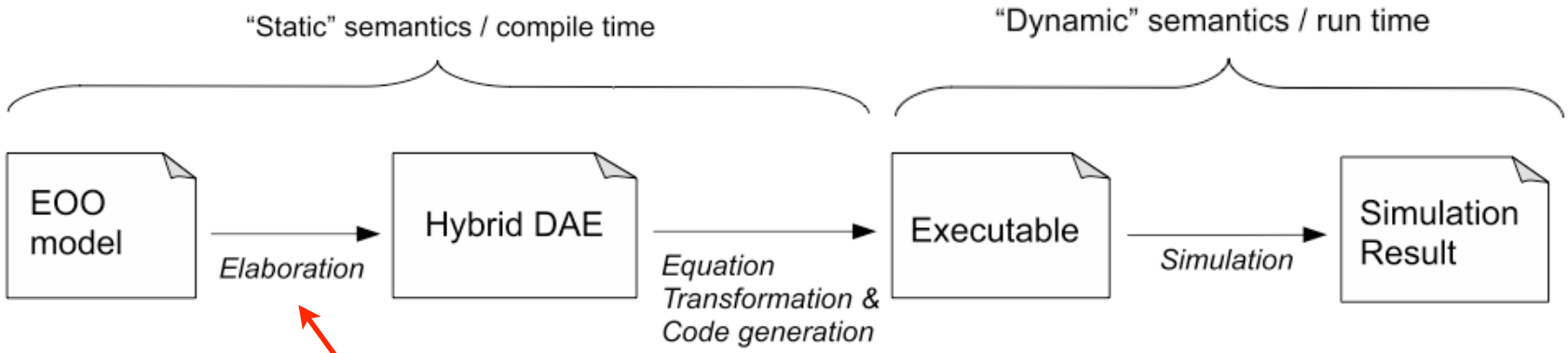
David Broman
davbr@ida.liu.se

DEFINITION 2 (Acausal Model).
An acausal model is an abstraction that encapsulates and composes

1. *continuous-time behavior in form of differential algebraic equations (DAEs).*
2. *other interconnected acausal models, where the direction of information flow between sub-models is not specified.*

Also common in EOOL

- Connections between models can typically both express potential connections (across) and flow (also called through).
- Possibility to express discrete events



Our semantic in this work concerns the elaboration phase.

Higher-Order Acausal Models

David Broman
davbr@ida.liu.se

DEFINITION 3 (Higher-Order Acausal Model (HOAM)).
A higher-order acausal model is an acausal model, which can be

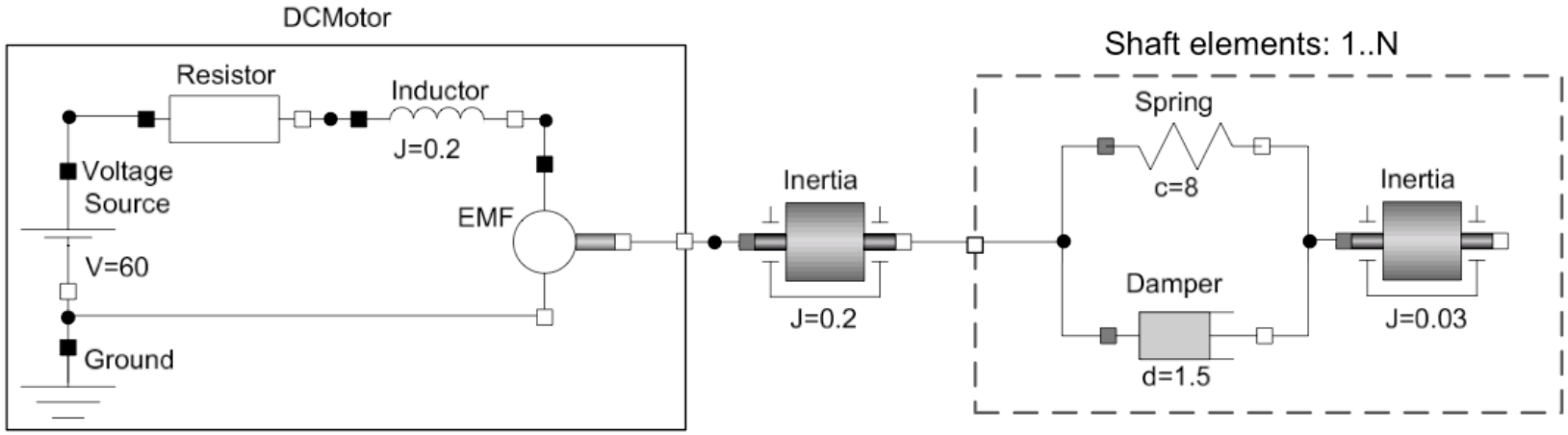
1. *parametrized with other HOAMs.*
2. *recursively composed to generate new HOAMs.*
3. *passed as argument to, or returned as result from functions.*

Emphasizes that HOAMs are first-class citizens, i.e., values that can be passed around.

David Broman
davbr@ida.liu.se

Part II

Higher Order Modeling in MKL

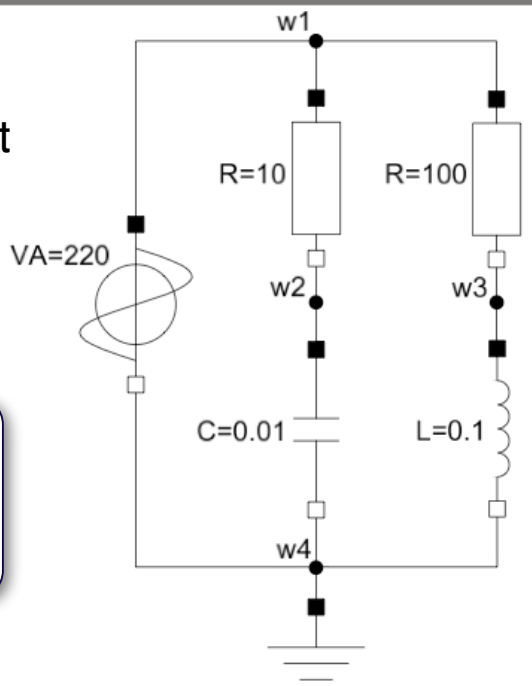


Basic Physical Modeling in MKL

David Broman
davbr@ida.liu.se

```
def Circuit = model(){
    def w1 = Wire();
    def w2 = Wire();
    def w3 = Wire();
    def w4 = Wire();
    Resistor(w1,w2,10);
    Capacitor(w2,w4,0.01);
    Resistor(w1,w3,100);
    Inductor(w3,w4,0.1);
    VSourceAC(w1,w4,220);
    Ground(w4);
};
```

Wires are used to connect model instances.




```
def Wire = func(){
    (var(),flow())
};
```

```
def TwoPin = model((pv,pi),(nv,ni),v){
    v = pv - nv;
    0 = pi + ni;
};
```

```
def Inductor = model(p,n,L){
    def (_,pi) = p;
    def v = var(0);
    TwoPin(p,n,v);
    L*der(pi) = v;
};
```

TwoPin is used by composition.

Differential equations

Higher-Order Acausal Models

David Broman
davbr@ida.liu.se

DEFINITION 3 (Higher-Order Acausal Model (HOAM)).
A higher-order acausal model is an acausal model, which can be

1. *parametrized with other HOAMs.*
2. *recursively composed to generate new HOAMs.*
3. *passed as argument to, or returned as result from functions.*

1. Parameterized with other HOAMs

David Broman
davbr@ida.liu.se

```
def Automobile = model(Engine, Tire){
    def c1 = Connection();
    def c2 = Connection();
    Engine(c1);
    Gearbox(c1,c2);
    Tire(c2); Tire(c2); Tire(c2); Tire(c2)
};
```

Two formal parameters: Engine and Tire

Creating automobile instances with different engines

```
Automobile(EngineV6,TireTypeA);
```

```
Automobile(EngineV8,TireTypeA);
```

Note: Similar to the Modelica declare construct.

Higher-Order Acausal Models

David Broman
davbr@ida.liu.se

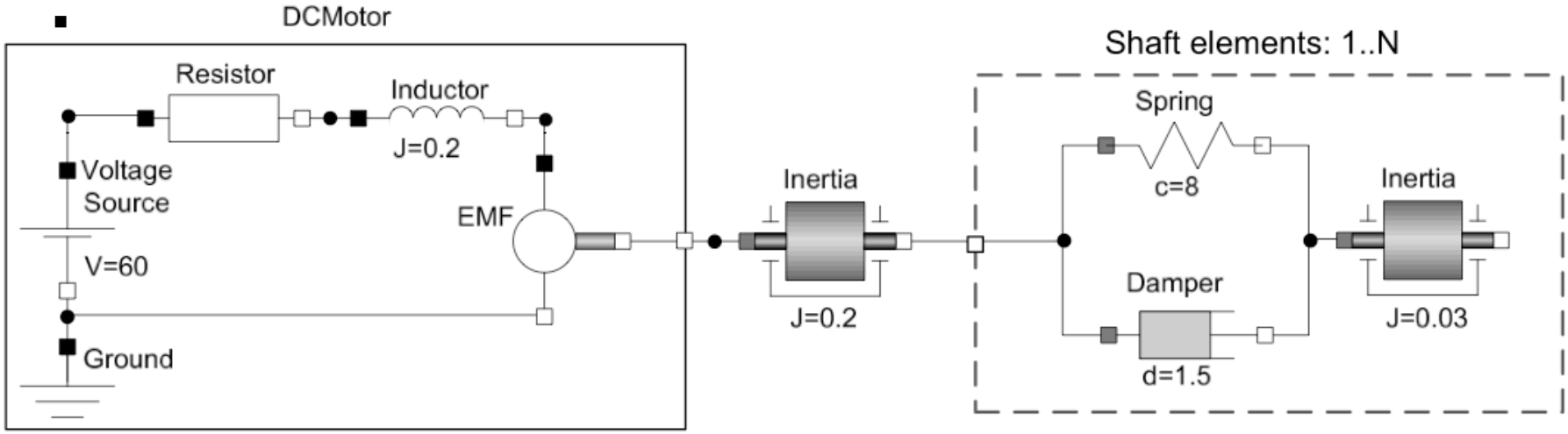
DEFINITION 3 (Higher-Order Acausal Model (HOAM)).
A higher-order acausal model is an acausal model, which can be

1. *parametrized with other HOAMs.*
2. *recursively composed to generate new HOAMs.*
3. *passed as argument to, or returned as result from functions.*

2. Recursively composed to generate new HOAMs

id Broman
@ida.liu.se

Example of a Mechatronic system with a DC motor and a flexible shaft




```
def MechSys = model(){
    def c1 = RotCon();
    def c2 = RotCon();
    DCMotor(c1);
    Inertia(c1,c2,0.2);
    FlexibleShaft(c2,RotCon(),120);
};
```

A rotational connector in the mechanical domain.

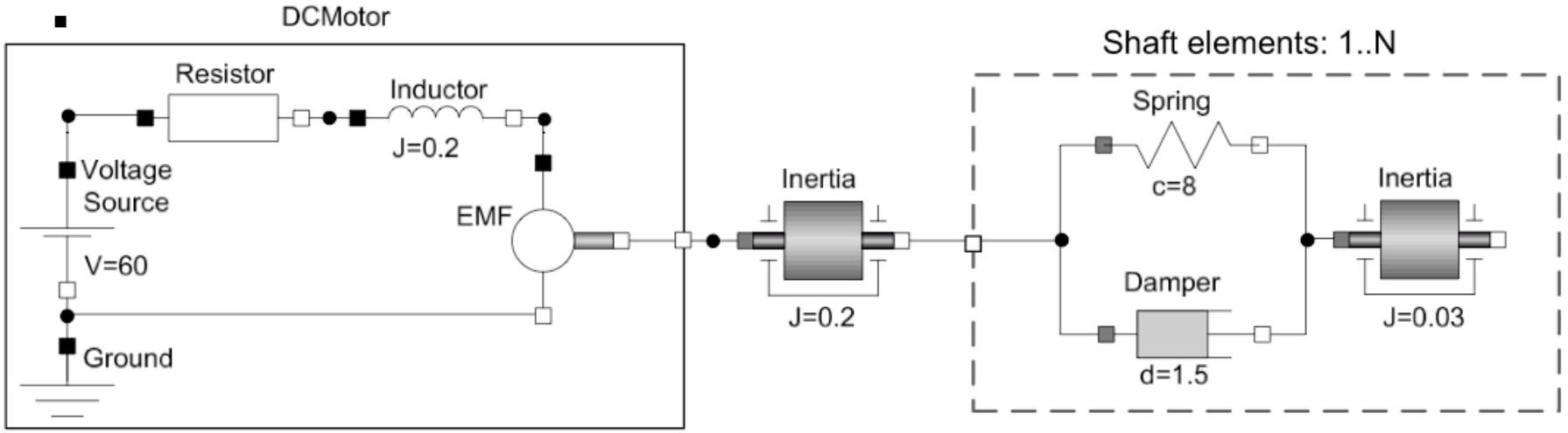
Creates a flexible shaft with 120 shaft elements.

How is this model defined?

2. Recursively composed to generate new HOAMs

id Broman
@ida.liu.se

Example of a Mechatronic system with a DC motor and a flexible shaft



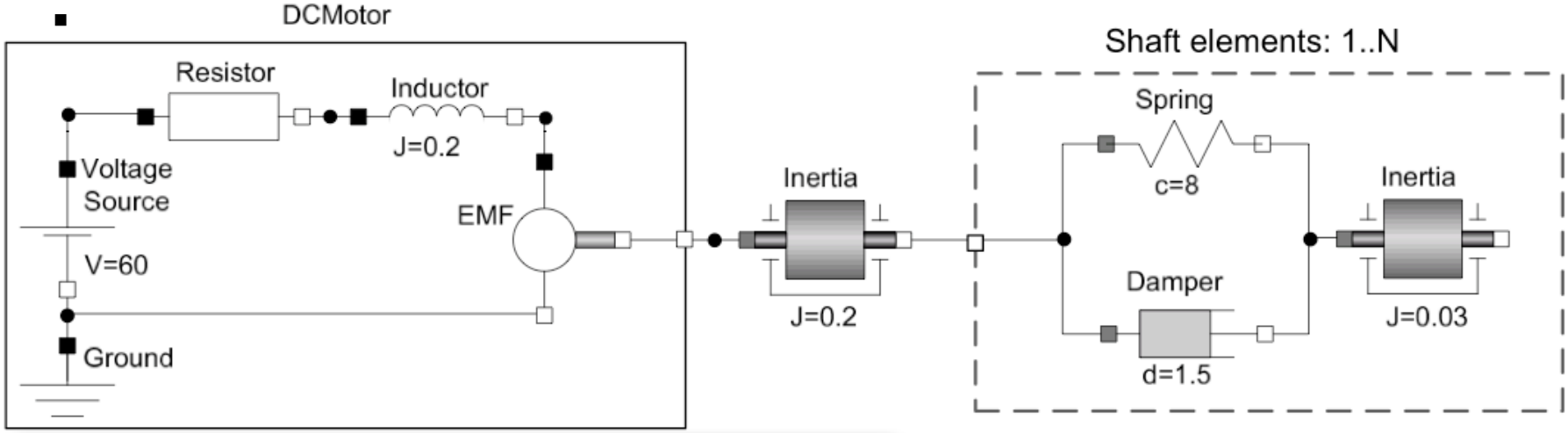
One shaft element is created by standard components.

```
def ShaftElement = model(ca,cb){
    def c1 = RotCon();
    Spring(ca,c1,8);
    Damper(ca,c1,1.5);
    Inertia(c1,cb,0.03);
};
```

2. Recursively composed to generate new HOAMs

id Broman
@ida.liu.se

Example of a Mechatronic system with a DC motor and a flexible shaft




```
defrec FlexibleShaft = model(ca,cb,n){
    if(n==1)
        ShaftElement(ca,cb)
    else{
        def c1 = RotCon();
        ShaftElement(ca,c1);
        FlexibleShaft(c1,cb,n-1);
    };
};
```

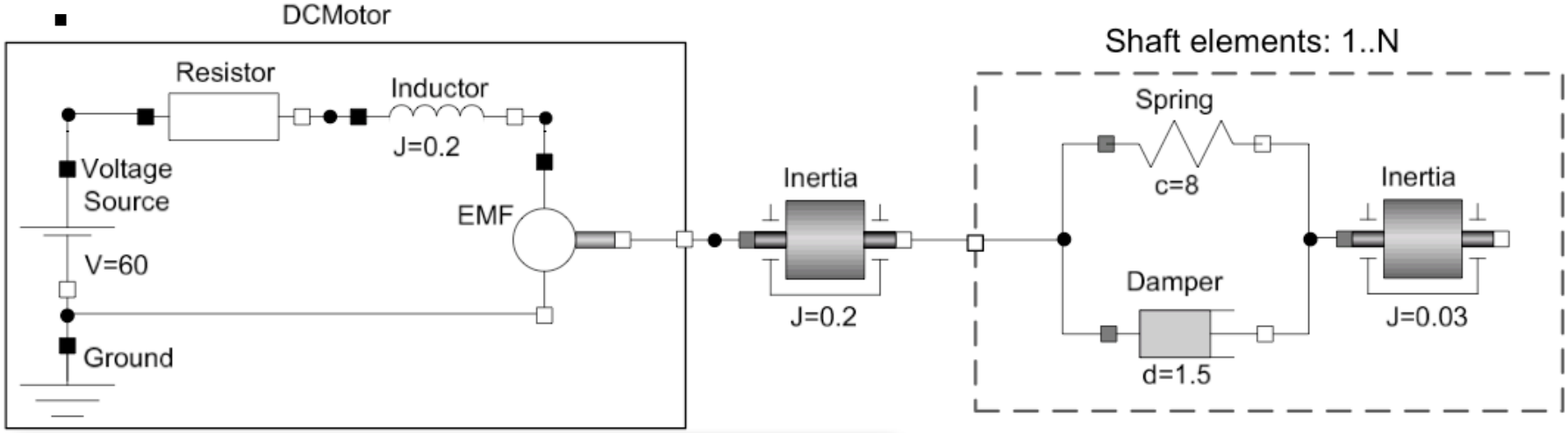
The flexible shaft is recursively defined by creating ShaftElements.

The recursion terminates after n steps (in the example 120 steps)

2. Recursively composed to generate new HOAMs

id Broman
@ida.liu.se

Example of a Mechatronic system with a DC motor and a flexible shaft




```
defrec FlexibleShaft = model(ca,cb,n){
    if(n==1)
        ShaftElement(ca,cb)
    else{
        def c1 = RotCon();
        ShaftElement(ca,c1);
        FlexibleShaft(c1,cb,r
    };
};
```

The flexible shaft is recursively defined by creating ShaftElements.

nates

Do we always need to write a new recursive definition of a model when we for example want to serialize a number of models?

Higher-Order Acausal Models

David Broman
davbr@ida.liu.se

DEFINITION 3 (Higher-Order Acausal Model (HOAM)).
A higher-order acausal model is an acausal model, which can be

1. *parametrized with other HOAMs.*
2. *recursively composed to generate new HOAMs.*
3. *passed as argument to, or returned as result from functions.*

3. Passed as argument to, or as result from functions

id Broman
@ida.liu.se

```
def composeparallel = func(M1,M2){
    model(p,n){
        M1(p,n);
        M2(p,n);
    }
};
```

Composes model M1 and M2 in parallel and returns a new model.

However, e.g. an inductor takes 3 arguments!

```
def Inductor = model(p,n,L){
```

```
def set = func(M,val){
    model(p,n){
        M(p,n,val);
    }
};
```

We can use a set function that defines e.g. the resistance or inductance.

We can now create a new composed model Foo.

```
def Foo = composeparallel(set(Resistor, 100),
                          set(Inductor, 0.1));
```

3. Passed as argument to, or as result from functions

id Broman
@ida.liu.se

```
def composeparallel = func(M1,M2){
    model(p,n){
        M1(p,n);
        M2(p,n);
    }
};
```

Composes model M1 and M2 in parallel and returns a new model.

However, e.g. an inductor takes 3 arguments!

```
def Inductor = model(p,n,L){
```

```
def set = func(M,val){
    model(p,n){
        M(p,n,val);
    }
};
```

We can use a set function that defines e.g. the

Why is this more expressive than defining the composed model directly?

new

o.

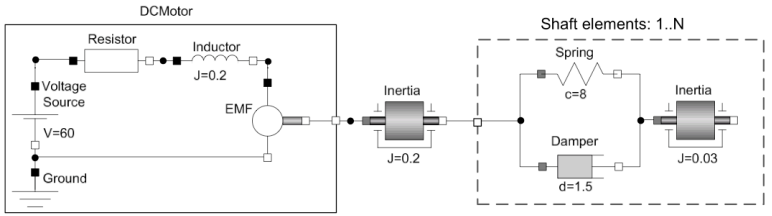
```
def Foo = composeparallel(set(
                          set(Ind
```

It's not, but imagine that you should compose 120 elements...

3. Passed as argument to, or as result from functions

id Broman
@ida.liu.se

```
defrec recmodel = model(M,C,ca,cb,n){
    if(n==1)
        M(ca,cb)
    else{
        def c1 = C();
        M(ca,c1);
        recmodel(M,C,c1,cb,n-1);
    };
};
```



Similar to the flexibleshaft model, but now with an arbitrary model M and connection constructor C.

After encapsulation, we have a transformation function that returns a new serialized model with two pins.

```
def serialize = func(M,C,n){
    model(ca,cb){
        recmodel(M,C,ca,cb,n);
    }
};
```

3. Passed as argument to, or as result from functions

id Broman
@ida.liu.se

```
def MekSys2 = model(){
    def c1 = RotCon();
    def c2 = RotCon();
    DCMotor(c1);
    Inertia(c1,c2,0.2);
    def FlexibleShaft =
        serialize(ShaftElement,RotCon,120);
    FlexibleShaft(c2,RotCon());
};
```

We can now use the generic function to serialize 120 shaftelements.

The good news is that once the serialize transformation function is defined, it can be reused with arbitrary model which has two pins.

```
def Res50 =
   serialize(set(Resistor,100), Wire, 50);
```

David Broman
davbr@ida.liu.se

Part III

Related Work and Future Perspective

Related Work (1/2)

David Broman
davbr@ida.liu.se

Functional Hybrid Modeling (FHM)

(Nilsson, Peterson, and Hudak, 2007)

- Have a similar concept called *first-class relations on signals*.
- Similarity: First-class and can be recursively defined.
- Difference: MKL models can be parameterized on any type, where first-class relations on signals in FHM are parameterized using ordinary function abstraction.
- Compared to MKL, FHM has yet no published formal semantics.

Metaprogramming and Metamodeling

E.g. MetaML and Template Haskell

- Metaprograms are programs that take other programs / models as data and produces new programs / models as output.
- Approach of HOAMs enables access to transform models direct in the language without representing models as data.
- Metaprogramming can on the other hand enables greater generality of model transformations.

Related Work (2/2)

David Broman
davbr@ida.liu.se

Modelica semantics

(Modelica Association, 2007)

- The redeclare construct have similar ability as passing HOAMs to other HOAMs.
- For-equations can be used to create sequences of connected models, i.e. same as recursive HOAMs.
- It is not yet possible to create model transformation functions in Modelica (such as the serialize function), since models cannot be passed into functions.
- The Modelica semantics are informally defined using natural language. It's semantics are complex and large. MKL on the other hand has a very small formal semantics.

Future Perspective

David Broman
davbr@ida.liu.se

Current Limitations

- HOAMs as presented here are limited to the elaboration phase.

Interesting Future Research

- HOAMs as part part of the run-time (simulation-time), i.e., run-time creation of models, compositon of models.
- A general approach to structurally variable systems, i.e. models can be transformed, instantiated and destroyed at run-time.

Research challenges

- How can we guarantee static type-safety?
- How can we preserve high performance? E.g. how do we handle index reduction?
- Is it possible to define a formal sound semantics of such a language?

Conclusions

David Broman
davbr@ida.liu.se

Higher-Order Functions + **Acausal Models** = **Higher-Order Acausal Models (HOAMs)**

DEFINITION 3 (Higher-Order Acausal Model (HOAM)).
A higher-order acausal model is an acausal model, which can be

1. *parametrized with other HOAMs.*
2. *recursively composed to generate new HOAMs.*
3. *passed as argument to, or returned as result from functions.*

Thanks for listening!