Modeling Kernel Language (MKL)

A formal and extensible approach to equation-based modeling languages

Guest Talk, EECS, Chess, UC Berkeley
February 17, 2011

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

Agenda

Part I
What is an EOO Language?

\[
\begin{align*}
J_1 \omega_1 &= M_v - M_1 \\
J_2 \omega_2 &= M_h - M_2 \\
\omega_1 &= -r \omega_2 \\
M_1 &= -r^{-1} M_2
\end{align*}
\]

Part II
Why MKL?

Part III
Expressiveness, Extensibility, and Formalization
Part I

What is an EOO language?

\[
\begin{align*}
J_1 \omega_1 &= M_v - M_1 \\
J_2 \omega_2 &= M_h - M_2 \\
\omega_1 &= -r \omega_2 \\
M_1 &= -r^{-1} M_2
\end{align*}
\]

What is Modeling and Simulation?

- Mathematical Model
  - Differential-Algebraic Equations (DAEs)

- System
- Model
- Simulation

experiment on...

answer questions about...
Equation-Based Object-Oriented (EOO) Languages

Domain-Specific Language (DSL)
- Primarily domain: Modeling of physical systems
- Multiple physical domains: e.g., mechanical, electrical, hydraulic

Models and Objects
- Object in e.g., Java, C++: object = data + methods
- Objects in EOO languages: object = data + equations

Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility and Formalization

Equation-Based Object-Oriented (EOO) Languages

Models and Objects

• Object in e.g., Java, C++: object = data + methods
• Objects in EOO languages: object = data + equations

Equation-Based Object-Oriented (EOO) Languages

Domain-Specific Language (DSL)
- Primarily domain: Modeling of physical systems
- Multiple physical domains: e.g., mechanical, electrical, hydraulic

Models and Objects

• Object in e.g., Java, C++: object = data + methods
• Objects in EOO languages: object = data + equations
Equation-Based Object-Oriented (EOO) Languages

Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility and Formalization

Domain-Specific Language (DSL)
• Primarily domain:
  Modeling of physical systems
• Multiple physical domains:
  e.g., mechanical, electrical, hydraulic

Equation-Based Object-Oriented (EOO)

Models and Objects
• Object in e.g., Java, C++:
  object = data + methods
• Objects in EOO languages:
  object = data + equations

Acausality
• At the equation-level
  \[ u = R \cdot i \]
• At the object connection level

Equation-Based Object-Oriented (EOO) Languages

Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility and Formalization

Domain-Specific Language (DSL)
• Primarily domain:
  Modeling of physical systems
• Multiple physical domains:
  e.g., mechanical, electrical, hydraulic

Models and Objects
• Object in e.g., Java, C++:
  object = data + methods
• Objects in EOO languages:
  object = data + equations

Acausality
• At the equation-level
  \[ u = R \cdot i \]
• At the object connection level

Equation-Based Object-Oriented (EOO) Languages

Direction not determined at modeling time

Physical topology is lost
Equation-Based Object-Oriented (EOO) Languages

Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility and Formalization

Domain-Specific Language (DSL)

• Primarily domain: Modeling of physical systems

• Multiple physical domains: e.g., mechanical, electrical, hydraulic

Models and Objects

• Object in e.g., Java, C++:
  object = data + methods

• Objects in EOO languages:
  object = data + equations

Equation-Based Object-Oriented (EOO)

Acausality

• At the equation-level
  \[ u = R \times i \]

• At the object connection level

Acausal (non-causal)

Causal

Equations

- Modelica
- VHDL-AMS
- gPROMS
Part II

Why MKL?

Expressiveness

Expressiveness – ease and possibility of expressing complex models or tasks

Language versions:
- A, v1.0
- A, v1.1
- A, v2.0
- A, v2.2

Standard library versions:
- L, v1.0
- L, v1.1
- L, v2.0
- L, v2.2
Extensibility – mechanisms to add new language features

**Uses**
- Simulation
- Optimization
- Code generation for real-time
- Model export
- Grey-box system identification etc.

Different language versions:
- A, v1.0
- A, v1.1
- A, v2.0
- A, v2.2

- B, v1.0
- C, v1.0

Gives many dialects and different languages.

Gives larger and more complex languages.

---

Formalization – precise semantics “meaning” of the language

**Language Specifications of state-of-the-art are informally defined**

- Hard to interpret unambiguously when developing compilers
- Hard to reason about when extending the language
- Hard to formalize e.g. Modelica due to size and complexity
Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility, and Formalization
Higher-Order Acusal Models (HOAM)

**Expressiveness - HOAM**

**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.

Replaces several of Modelica’s constructs with one concept, e.g.,
- Conditional components
- For-equations
- Redeclare construct
Example of a mechatronic system with a DC motor and a flexible shaft

**Part I**
What is an EOO language?

**Part II**
Why MKL?

**Part III**
Expressiveness, Extensibility and Formalization

---

**HOAM – Example**

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

- **DCMotor**
  - Resistor
  - Inductor \( J=0.2 \)
  - EMF
  - Voltage Source \( V=60 \)
  - Ground

- **Shaft elements**: 1..N
  - Spring \( c=8 \)
  - Damper \( d=1.5 \)
  - Inertia \( J=C.5 \)

---

**let** MechSys =

- **let** r1:Rotational in
- **let** r2:Rotational in
- **let** r3:Rotational in
- DCMotor r1;
- Inertia 0.2 r1 r2;
- FlexibleShaft 120 r2 r3

*Creates a flexible shaft with 120 shaft elements.*

---

**let** Inductor L:Real -> p:Electrical -> n:Electrical -> Equations =

- **let** i:Current in
- **let** v:Voltage in
- ElectricalBranch i v p n;
- \( L \times (\text{der } i) = v \)
Example of a mechatronic system with a DC motor and a flexible shaft

**Part I**
What is an EOO language?

**Part II**
Why MKL?

**Part III**
Expressiveness, Extensibility and Formalization

**HOAM – Example**

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

```haskell
let ShaftElement flangeA:Rotational -> flangeB:Rotational ->
    Equations =
    let r1:Rotational in
    Spring 8. flangeA r1;
    Damper 1.5 flangeA r1;
    Inertia 0.5 r1 flangeB
```

One shaft element is created by standard components.

**Part I**
What is an EOO language?

**Part II**
Why MKL?

**Part III**
Expressiveness, Extensibility and Formalization

**HOAM – Example**

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

```haskell
let FlexibleShaft n:Int -> flangeA:Rotational -> flangeB:Rotational -> Equations =
    if n == 1 then
        ShaftElement flangeA flangeB
    else
        let r1:Rotational in
        ShaftElement flangeA r1;
        FlexibleShaft (n-1) r1 flangeB
```

The flexible shaft is recursively defined by creating ShaftElements.

The recursion terminates after n steps (in the example 120 steps)
**HOAM – Example**

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

\[
\text{let MechSys} = \\
\text{let r1:Rotational in} \\
\text{let r2:Rotational in} \\
\text{let r3:Rotational in} \\
\text{DCMotor r1;} \\
\text{Inertia 0.2 r1 r2;} \\
\text{FlexibleShaft 120 r2 r3}
\]

Do we always need a special recursive model?

---

**HOAM – Example**

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

\[
\text{let MechSys} = \\
\text{let r1:Rotational in} \\
\text{let r2:Rotational in} \\
\text{let r3:Rotational in} \\
\text{DCMotor r1;} \\
\text{Inertia 0.2 r1 r2;} \\
(\text{serializeRotational 120 ShaftElement}) \text{ r2 r3}
\]

Higher-order function that can compose any mechanical component in series
Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility and Formalization

Language Specification
- Type checking
- Collapsing the instance hierarchy
- Connection Semantics
- Simulation (Runtime)

Benefits
• Tool vendors – no need to update tool after lib ext.
• Library developer - less dependent on tool vendors
• A model behaves the same way in different tools

Language Specification
- Type checking
- Collapsing the instance hierarchy
- Connection Semantics
- Simulation (Runtime)
**Intensional Analysis and Model Lifting**

**Static Semantics**

- Lifted Model
- Model Lifting
- Type Checking
- MKL Model

**Dynamic Semantics**

- Lifted Model
- Lifted Model
- Collapsed using evaluation
- Equation System
- Analysis, models treated as data. Connection Semantics, Simulation, etc.

---

**Intensional Analysis – an Example**

```
type InitValMap = (Real => Real)
```

Computing the mapping from unknowns to initial values

```
let initValues eqs:Equations -> InitValMap =
  let get eqs:Equations -> acc:InitValMap -> InitValMap =
    match eqs with
    | e1 ; e2 -> get e2 (get e1 acc)
    | Init x (val v:Real) -> Map.add x v acc
    | _ -> acc
  in get eqs (Map.empty)
```
Part I
What is an EOO language?

Part II
Why MKL?

Part III
Expressiveness, Extensibility and Formalization

How do we verify our solution?

Prototype Implementation
Syntax and Dynamic Semantics

Abstract Syntax (core of MKL)

Variables \( x, y \in X \)
Unknowns \( u \in U \)
Constants \( c \in C \)
Expressions \( e \) ::= \( x \mid \lambda x: \tau. e \mid e e \mid c \mid u: \tau \mid \nu(\tau) \mid e @ e \mid \text{val } e: \tau \mid \text{decon} (e, d, e, e) \)
Deconstruct patterns \( d \) ::= \( \text{uk: } x \mid \text{val } x: \tau \)
Values \( v \) ::= \( \lambda x: \tau. e \mid c \mid u: \tau \mid v @ v \mid \text{val } v: \tau \)
Ground Types \( \gamma \in \Gamma \)
Types \( \tau \) ::= \( \gamma \mid \tau \rightarrow \tau \mid <\tau> \mid <> \)

Big-Step Semantics (selected rule)

\[
\frac{e_1 \mid U_1 \Rightarrow \lambda x: \tau. e_3 \mid U_2}{e_2 \mid U_2 \Rightarrow v_1 \mid U_3 \quad [x \mapsto v_1] e_3 \mid U_3 \Rightarrow v_2 \mid U_4 \quad e_1 e_2 \mid U_1 \Rightarrow v_2 \mid U_4} \quad \text{(BS-APPABS)}
\]

Small-Step and Type System

Small-Step Semantics (selected rules)

Computation Rules

\[
\frac{c \mid U \rightarrow c' \mid U'}{c \mid U \rightarrow c' \mid U'} \quad \text{(E-APP)}
\]

\[
\frac{(\lambda x: \tau_1. c_1) c_2 \mid U \rightarrow [x \mapsto c_2] c_1 \mid U}{U \rightarrow \delta(c_1, c_2) \mid U} \quad \text{(E-Delta)}
\]

\[
\frac{u \notin U}{\nu(\tau) \mid U \rightarrow u: <\tau> \mid U \cup \{u\}} \quad \text{(E-NEWUK)}
\]

Congruence Rules

\[
\frac{c_1 \mid U \rightarrow c'_1 \mid U'}{c_1 \mid U \rightarrow c'_1 \mid U'} \quad \text{(E-APP1)}
\]

\[
\frac{e_2 \mid U \rightarrow e'_2 \mid U'}{e_1 \mid U \rightarrow e_1 e_2 \mid U' \rightarrow e'_1 e_2 \mid U'} \quad \text{(E-APP2)}
\]

Type System (selected rule)

\[
\frac{\Gamma \vdash_L e \leadsto e': \tau}{\Gamma \vdash_L e_1 \mid e_2 \leadsto e'_1 e'_2 : \tau_1 \quad \tau_1 \sim \tau_2}{\Gamma \vdash_L e_1 e_2 \leadsto e'_1 e'_2 : \tau_1 \quad \tau_1 \sim \tau_2} \quad \text{(L-APP)}
\]

Part I

What is an EOO language?

Part II

Why MKL?

Part III

Expressiveness, Extensibility and Formalization
Main Lemmas

Lemma 10.5 (Progress)
If $\vdash e : \tau$ then $e \in \text{Values}$ or for all $U$ there exists $U'$ and $e'$ such that $e \mid U \rightarrow e' \mid U'$.

Lemma 10.8 (Preservation)
If $\Gamma \vdash e : \tau$ and $e \mid U \rightarrow e' \mid U'$ then $\Gamma \vdash e' : \tau$.

Conclusions

Expressiveness (Library Approach)
Extensibility (Library Approach)
Formalization (Operational Semantics)

Modeling Kernel Language (MKL)

Thanks for listening!