Extensible Modeling Languages and Precision Timed Infrastructures for Cyber-Physical Systems

RAWFP Workshop, Gothenburg, Sweden

May 28, 2012

David Broman
broman@eecs.berkeley.edu

EECS Department
UC Berkeley, USA

and

Linköping University, Sweden

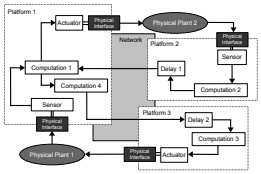
Key collaborator related to the Modelyze project
Jeremey G. Siek

Key contributors and collaborators related to the PRET project

Edward A. Lee	**Isaac Liu**	**Jan Reineke**	Jia Zou
Steven A. Edwards	Slobadon Matic	Sanjit Seshia	Sungjun Kim
Jeff Jensen	Hiren Patel	Michael Zimmer	

Agenda

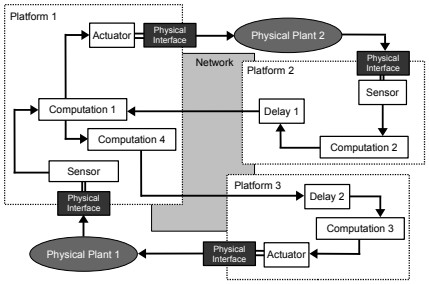
Part I
Cyber-Physical Systems



Part II
Modelyze -
a host language for equation-based languages

Part III
Precision Timed Infrastructure –
making time an engineering abstraction

Part I
Cyber-Physical Systems



Complex Cyber-Physical Systems (CPSs)

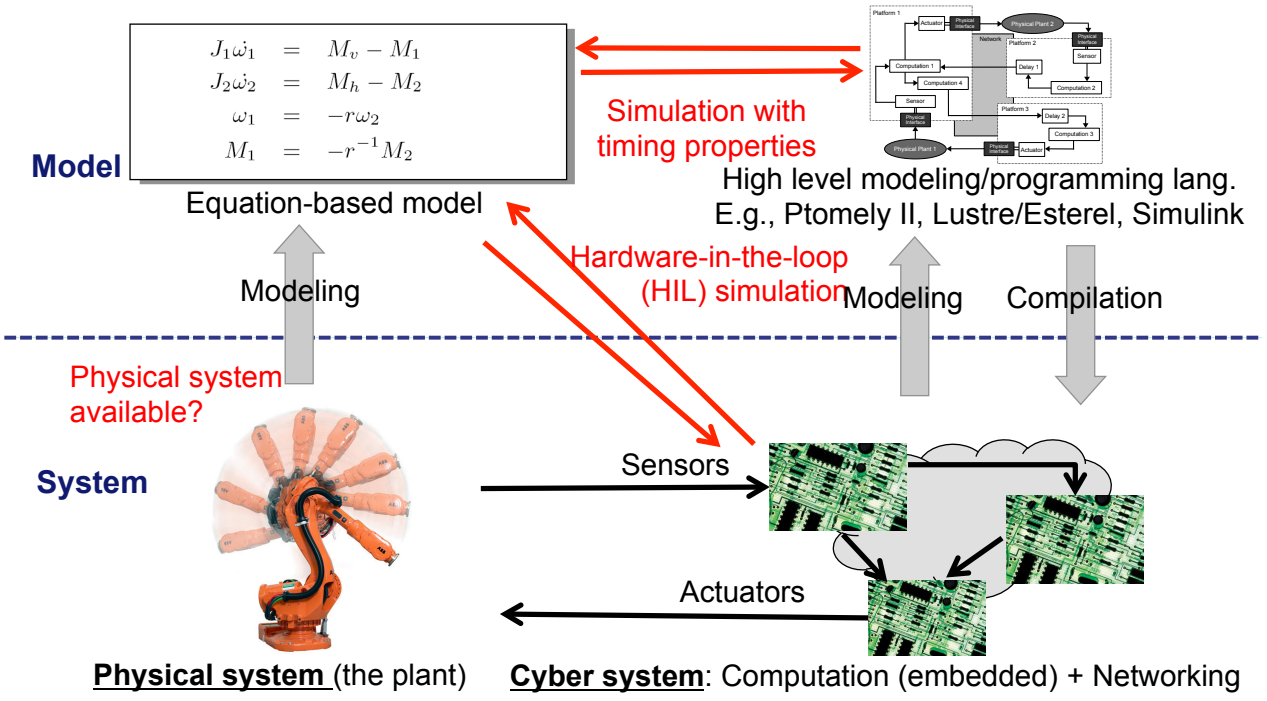
Industrial Robots

Power Plants

Aircraft

Modeling, Simulating, and Running Cyber-Physical Systems

broman@eecs.berkeley.edu



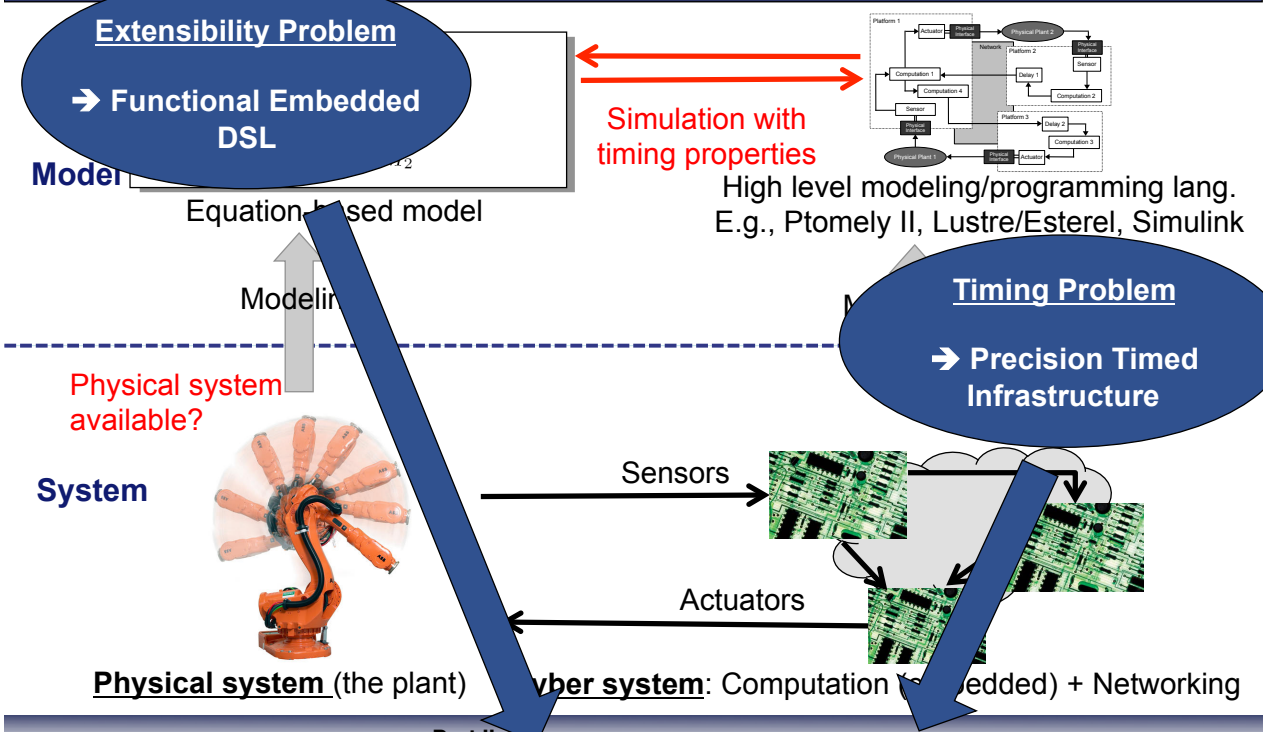
Part I
Cyber-Physical Systems

Part II
Modelyze - a host language for equation-based languages

Part III
Precision Timed Infrastructure - making time an engineering abstraction

Modeling, Simulating, and Running Cyber-Physical Systems

broman@eecs.berkeley.edu

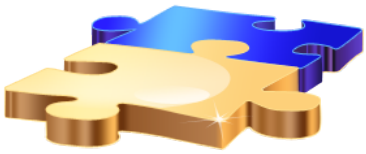


Part I
Cyber-Physical Systems

Part II
Modelyze - a host language for equation-based languages

Part III
Precision Timed Infrastructure - making time an engineering abstraction

Part II
Modelyze
a host language for equation-based languages



In collaboration with

Jeremy G. Siek
University of Colorado at Boulder

Problem: Expressiveness and Extensibility

Cannot express all modeling or analysis needs.
Limited to what the modeling language can provide.

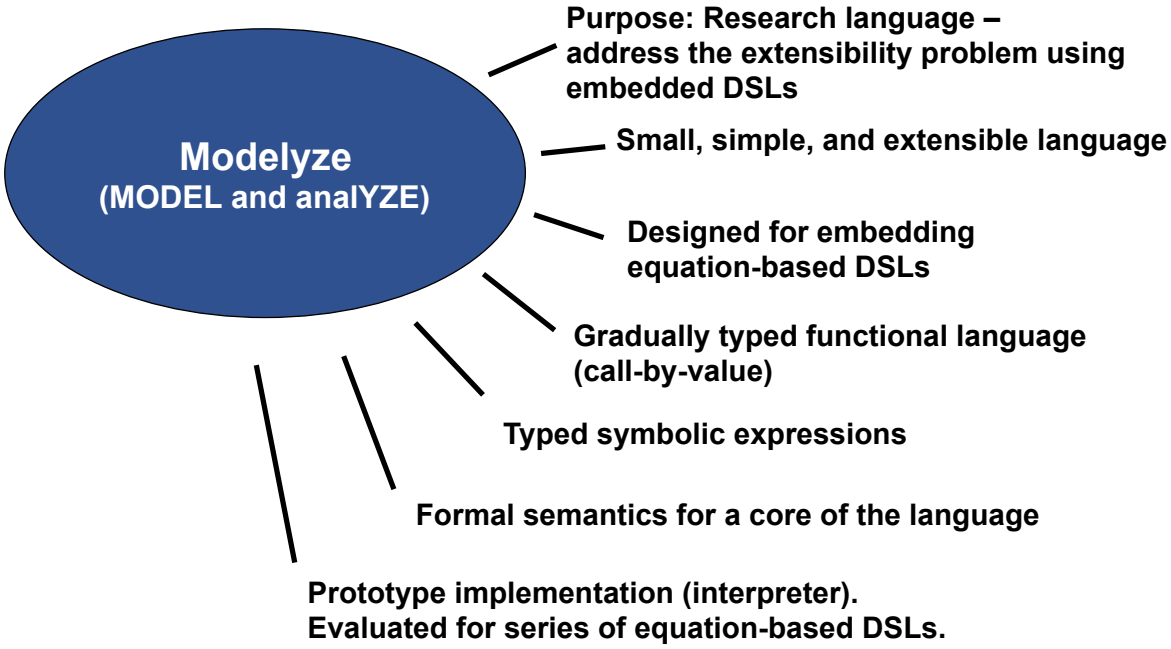
Language versions: A, v1.0 → A, v1.1 → A, v2.0 → A, v2.2

Standard library versions: L, v1.0 → L, v1.1 → L, v2.0 → L, v2.2

Modelica: A new language definition approximately every second year

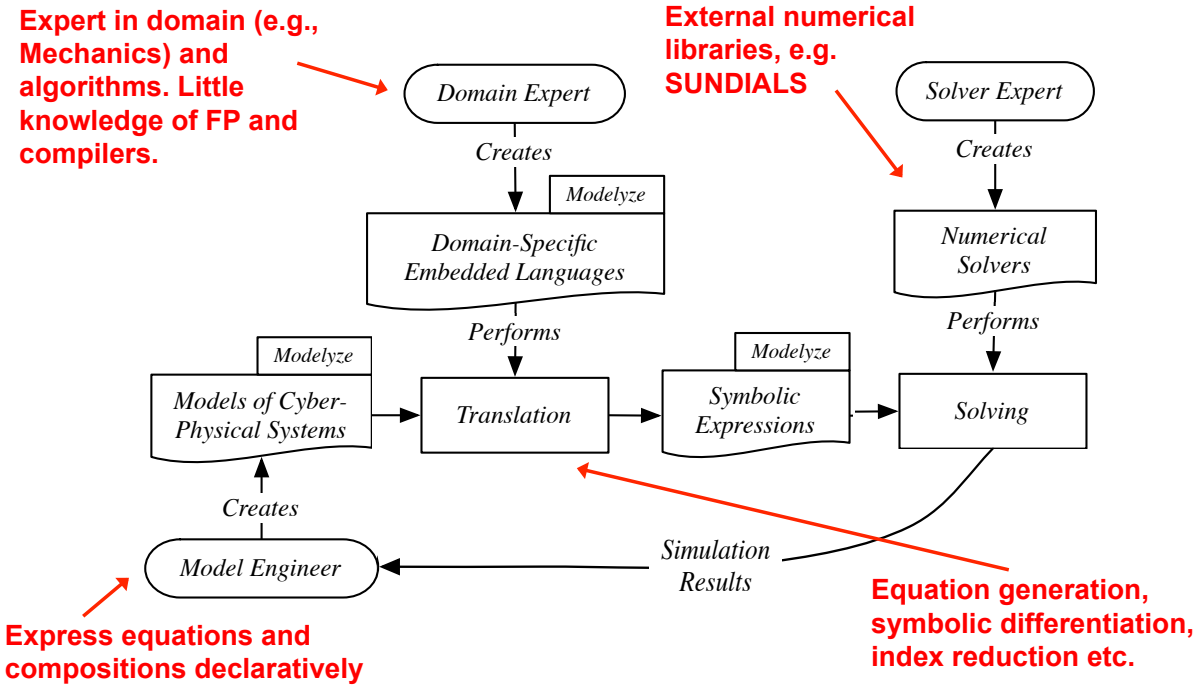
What is Modelyze?

broman@eecs.berkeley.edu



Process Overview

broman@eecs.berkeley.edu



Challenges and Contributions

broman@eecs.berkeley.edu

1. Provide seamless integration between host language and embedded DSL

Performed together with type checking.

Symbol Lifting Analysis

2. Make it easy to transform and analyze equations (domain expert)

Pattern matching on symbolic expressions

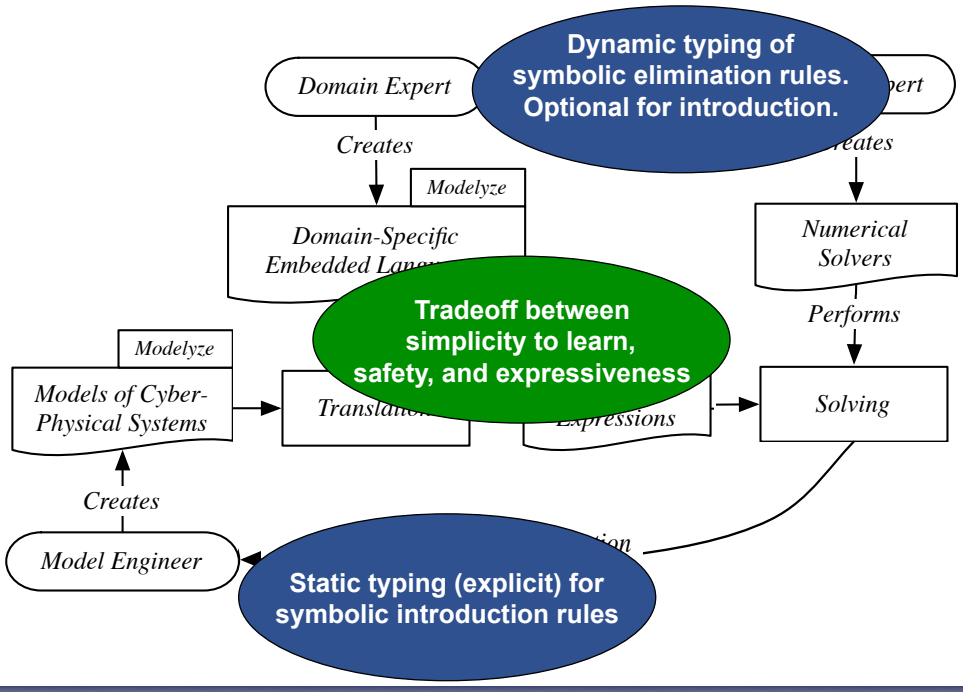
Gradual Typing + Typed Symbolic Expressions

3. Provide domain specific errors (model engineer)

Static type checking

Process Overview

broman@eecs.berkeley.edu



Related Work

broman@eecs.berkeley.edu

Implementing DSLs

Compiler construction

- JastAdd (Ekman & Hedin, 2007)
- MetaModelica (Pop & Fritzson, 2006)

Preprocessing and template metaprogramming

- C++ Templates (Veldhuizen, 1995)
- Template Haskell (Sheard & Peyton Jones, 2002)
- Stratego/XP (Bravenboer et al., 2008)

Embedded DSLs

- Haskel DSELs, e.g., Fran (Ellito & Hudak, 1997), Lava (Bjesse et al. 1998), and Paradise(Augustsson, 2008)
- FHM (Nilsson et al., 2003)
- Pure embedding (Higher-order functions, polymorphism, lazy evaluation, type classes) (Hudak, 1998)

Combining Dynamic and Static Typing

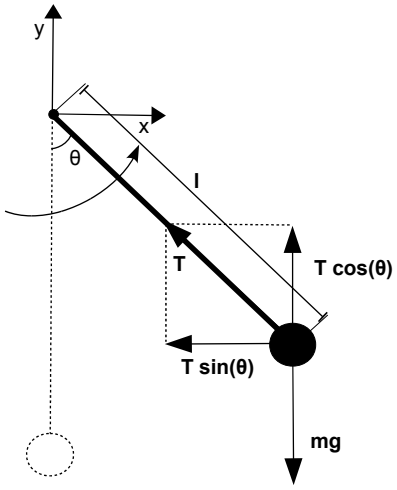
- Gradual Typing (Siek & Taha, 2007)
- Soft Typing (Cartwright & Fagan, 1991)
- Dynamic type with typecase (Abadi et al., 1991)
- Typed Scheme, Racket (Tobin-Hochstadt, Felleisen, 2008)
- Thorn, like types (Wrigstad et al., 2010)

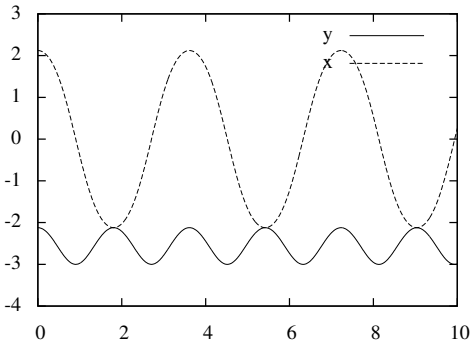
Representing Code and Data type

- Dynamic languages LISP, Mathematica
- MetaML <T> (Taha & Sheard, 2000)
- GADT (Peyton Jones et al.,2006; Xi et al., 2003; Cheney & Ralf, 2003)
- Open Data types (Löh & Hinze, 2006)
- Pattern Calculus (Jay, 2009)

Pendulum Example

broman@eecs.berkeley.edu





$$-T \cdot \frac{x}{l} = m\ddot{x}$$

$$-T \cdot \frac{y}{l} - mg = m\ddot{y}$$

$$x^2 + y^2 = l^2$$

$$x(0) = l \sin(\theta_s)$$

$$y(0) = -l \cos(\theta_s)$$

Differential-Algebraic equations

Algebraic constraint

Initial values

Declarative Mathematical Model

broman@eecs.berkeley.edu

Using function abstraction to define the model

Unknowns are given types but not bound to values

Equations and initial values are defined declaratively, just as the mathematical equations

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

$$-T \cdot \frac{x}{l} = m\ddot{x} \qquad x(0) = l\sin(\theta_s)$$
$$-T \cdot \frac{y}{l} - mg = m\ddot{y} \qquad y(0) = -l\cos(\theta_s)$$
$$x^2 + y^2 = l^2$$

Declarative Mathematical Model

broman@eecs.berkeley.edu

Which parts are part of the host language (Modelyze)?

Unknowns are internally represented as typed symbols

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

$s:\tau$

Fresh (unique) symbol

Tagged with a type

Variable `x` is bound to fresh symbol of type `<Real>`

$<\tau>$ ← Symbolic type

Release the user from annotation burden

broman@eecs.berkeley.edu

Symbols cannot be bound to values, so `x^2` would crash at runtime

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Use quasi-quoting to mix symbolic expressions and program code?

Using MetaML syntax `< >` for quotation and `~` for anti-quoting (escape)

```
<~x^2 + ~y^2 = ~((fun t -> <t>)l^2)>;
```

Heavy annotation burden for the end-user

Symbol Lifting Analysis (SLA)

broman@eecs.berkeley.edu

Symbol Lifting Analysis (SLA): During type checking, lift expressions that cannot be safely evaluated at runtime into symbolic expressions (data).

$$\Gamma \vdash_L e \rightsquigarrow e' : \tau$$

Rewritten to prefix curried form

```
((() x) l)
```

where

```
(/):Real-> Real -> Real
x:<Real>
l:Real
```

```
def Pendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

```
(((lift(/):Real-Real->Real) @ x) @ (lift l:Real))
```

Resulting type `<Real>`

Division cannot be performed, lift expression to type `<Real-> Real -> Real>`.

Term `lift e:T` wraps `e` and results in type `<T>`

Term `e1@e2` is a symbolic application, represented as a tuple.

Symbols used to construct equation systems

broman@eecs.berkeley.edu

```
def Pendulum(m:Real,l:Real,angle:Real) -> Equations = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y (-l*cos(angle));

    -T*x/l = m*x'';
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Symbolic data type

Type of **Pendulum** : Real -> Real -> Real -> Equations

Typed symbol used as constructors

```
type Equations
def (=) : Real -> Real -> Equations
def (;) : Equations -> Equations -> Equations
```

Data types are open with no exhaustive checking for patterns

```
def der : Real -> Real
def (')  = der
def init : Real -> Real -> Equations
```


Pattern Matching on Symbolic Expressions

broman@eecs.berkeley.edu

Dynamic symbolic type **<?>**

Accumulator Sets of symbolic type **<Real>**

Query for all unknowns in a model instance

```
def getUnknowns(exp:<?>, acc:(Set <Real>)) -> (Set <Real>) = {
    match exp with
    | e1 e2 -> getUnknowns(e2,getUnknowns(e1,acc))
    | sym:Real -> Set.add exp acc
    | _ -> acc
}
```

Uniform data structure, no boilplate code (matching on symbolic applications)

Match all symbols of type **<Real>** i.e., unknowns in the model.

```
getUnknowns(Pendulum(5,3,45*pi/180),Set.empty)
```

Returns a set with 3 symbols (representing **x**, **y**, and **T**).

Static Error Checking at the DSL Level

broman@eecs.berkeley.edu

Syntactically correct model (host syntax)

Static type error instead of dynamic error during translation/pattern matching.

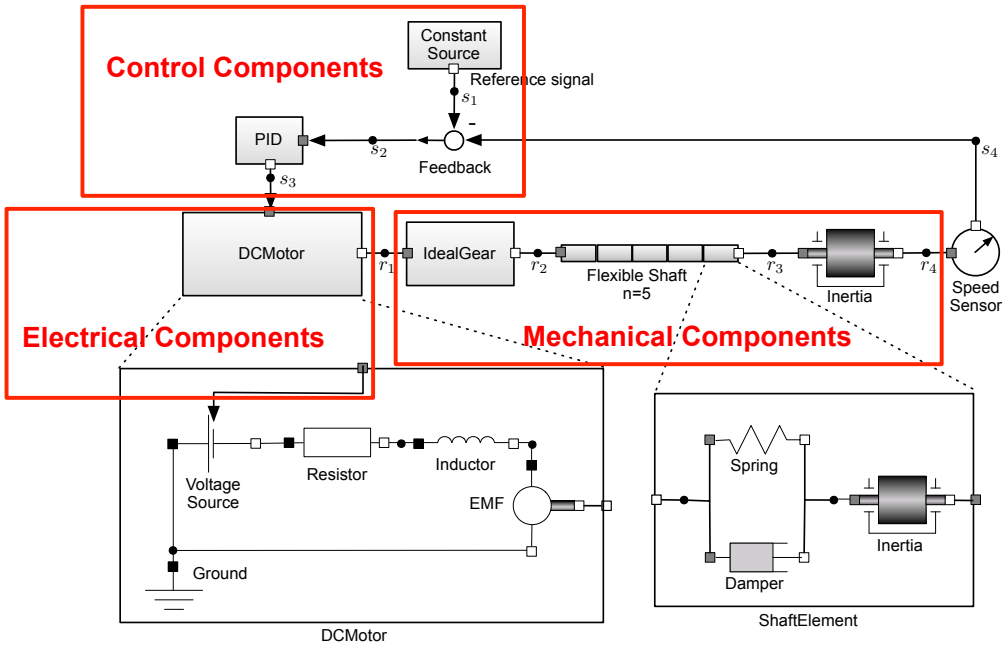
```
def ModifiedPendulum(m:Real,l:Real,angle:Real) = {
    def x,y,T:Real;
    init x (l*sin(angle));
    init y;                    //Error 1: Missing initial value

    -T*der/l = m*x'';          //Error 2: der used incorrectly
    -T*y/l - m*g = m*y'';
    x^2 + y^2 = l^2;
}
```

Still challenging to provide meaningful error messages.
"Expected type `<Real>` but found `<Real->Real>`"

Mechatronic Control Example

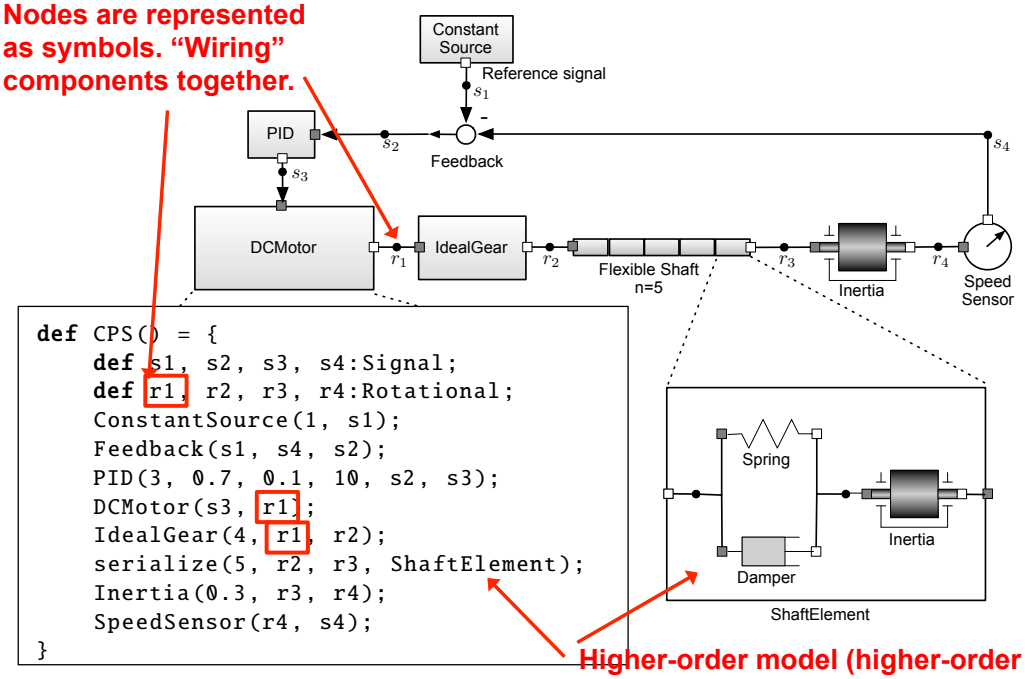
broman@eecs.berkeley.edu



Mechatronic Control Example

broman@eecs.berkeley.edu

Nodes are represented as symbols. "Wiring" components together.



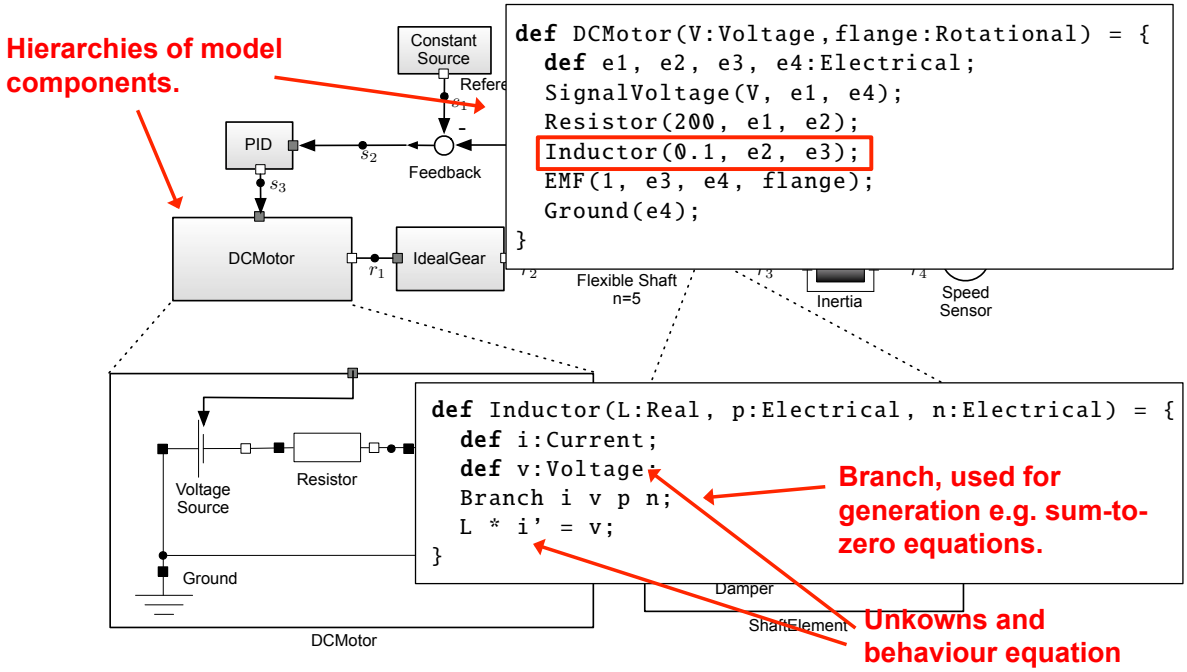

```
def CPS() = {
    def s1, s2, s3, s4:Signal;
    def r1, r2, r3, r4:Rotational;
    ConstantSource(1, s1);
    Feedback(s1, s4, s2);
    PID(3, 0.7, 0.1, 10, s2, s3);
    DCMotor(s3, r1);
    IdealGear(4, r1, r2);
    serialize(5, r2, r3, ShaftElement);
    Inertia(0.3, r3, r4);
    SpeedSensor(r4, s4);
}
```

Higher-order model (higher-order function)

Mechatronic Control Example

broman@eecs.berkeley.edu

Hierarchies of model components.




```
def DCMotor(V:Voltage,flange:Rotational) = {
  def e1, e2, e3, e4:Electrical;
  SignalVoltage(V, e1, e4);
  Resistor(200, e1, e2);
  Inductor(0.1, e2, e3);
  EMF(1, e3, e4, flange);
  Ground(e4);
}
```

```
def Inductor(L:Real, p:Electrical, n:Electrical) = {
  def i:Current;
  def v:Voltage;
  Branch i v p n;
  L * i' = v;
}
```

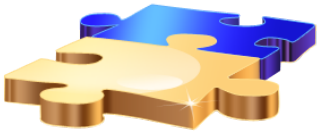
Branch, used for generation e.g. sum-to-zero equations.

Unkowns and behaviour equation

Conclusions

Main takeaway points (Modelyze)

A simple research language – host language for embedding equation-based DSLs



Symbol lifting analysis (SLA) is used to release the annotation burden from the user

Typed symbols with gradual typing is used for pattern matching and DSL-level error reporting

broman@eecs.berkeley.edu

Part II
Precision Timed Infrastructure
Making Time an Engineering Abstraction

Key contributors and collaborators related to the PRET project

Edward A. Lee	**Isaac Liu**	**Jan Reineke**	Jia Zou
Steven A. Edwards	Slobadon Matic	Sanjit Seshia	Sungjun Kim
Jeff Jensen	Hiren Patel	Michael Zimmer	

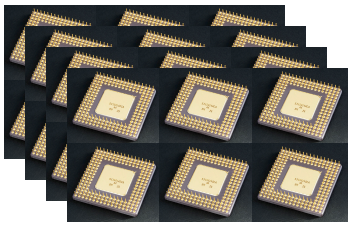
A Story…

broman@eecs.berkeley.edu

Fly-by-wire technology controlled by software.

Safety critical ➔
Rigorous validation and certification

Success?



They have to purchase and store microprocessors for at least 50 years production and maintenance…

Why?

Apparently, the software does not specify the behaviour that has been validated and certified!

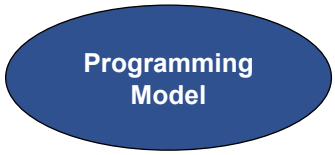
What is PRET?

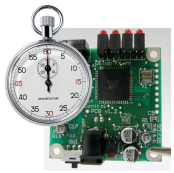
broman@eecs.berkeley.edu

Timing is not part of the software semantics

Correct execution of programs (e.g., in C, C++, C#, Java, Scala, Haskell, OCaml) has nothing to do with how long time things takes to execute.

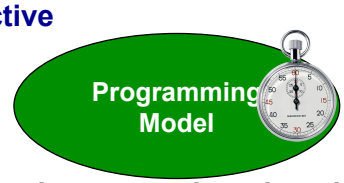
Traditional Approach



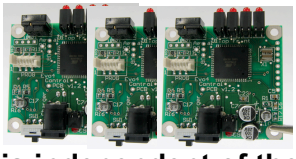


Timing Dependent on the Hardware Platform

Our Objective



Make time an engineering abstraction within the programming model



Timing is independent of the hardware platform (within certain constraints)

What is PRET?

broman@eecs.berkeley.edu

- **PRET = PRE**cision-**T**imed
- **PRET = P**redictable, **RE**peatable **T**iming
- **PRET = P**erformance *with* **RE**peatable **T**iming

PRET Infrastructure

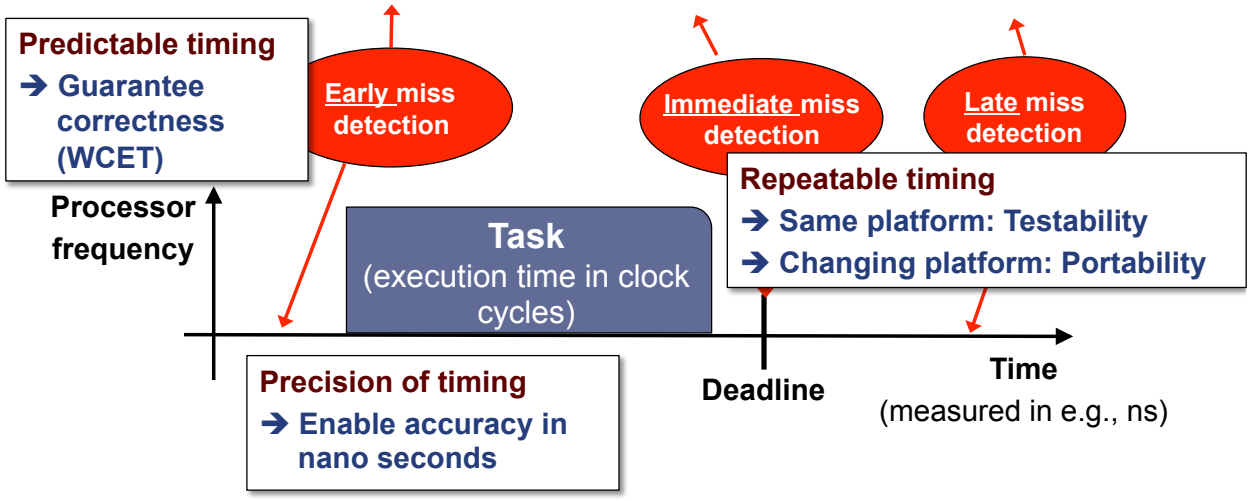
- **PRET Language (Language timing semantics)**
- **PRET Compiler (Timing aware compilation)**
- **PRET Machine (Computer Architecture)**

What do precision, predictable, and repeatable timing mean?

broman@eecs.berkeley.edu

Focus on cyber-physical systems with real-time constraints

	Hard task	Firm task	Soft task
Missed deadline	Catastrophic consequence	Result is useless, but causes no damage	Result has still some utility



Instruction set architecture (ISA)

broman@eecs.berkeley.edu

Rethink the ISA

Timing has to be a *correctness* property not only a *performance* (quality) property

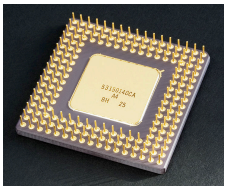


Photo by Andrew Dunn, 2005

PRET Machine

- **Repeatable and predictable execution time (instructions)**
- **Repeatable memory access time**
- **Timing instructions for handling missed deadline detection**

Related Work

Java Optimized Processor (JOP) (Schoeberl, 2008)

ARPRET (Andalam et al., 2009)

ARMv4 ISA extended with timing constructs

broman@eecs.berkeley.edu

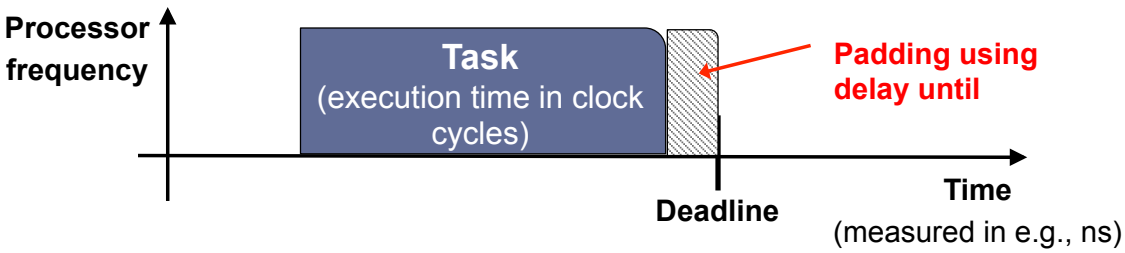
Best effort (with padding)

New instruction *get time* (gt)

```
gt      r1, r2      ; get time (ns)
  -- Code block --
adds    r2, r2, #500  ; add 500 ns
adc     r1, r1, #0    ; add with carry (time in 2 32-bit reg)
du      r1, r2      ; delay until 500ns have elapsed
```

New instruction *delay until* (du)

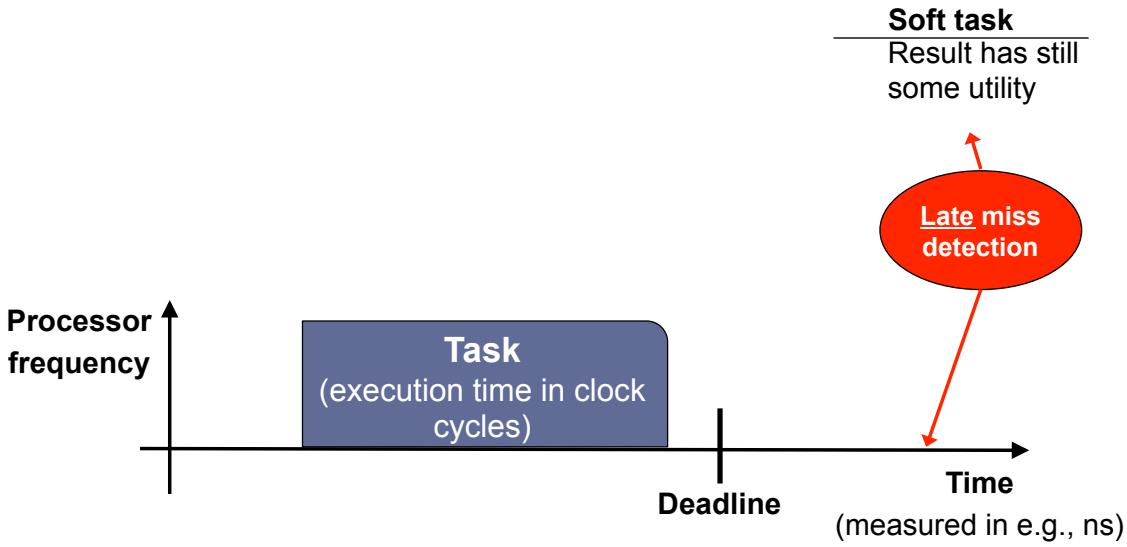
We do not detect missed deadlines!



ARMv4 ISA extended with timing constructs

broman@eecs.berkeley.edu

Late miss detection



ARMv4 ISA extended with timing constructs

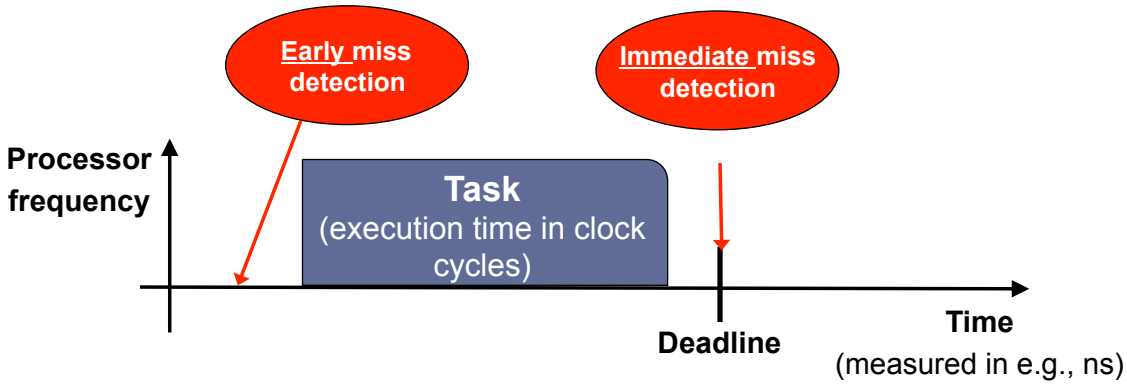
broman@eecs.berkeley.edu

Early and immediate miss detection

Not handled at the ISA level

Needs static timing analysis!

Handled using timing exceptions
New instructions:
ee - Exception on Expire
de - Deactivate Exception

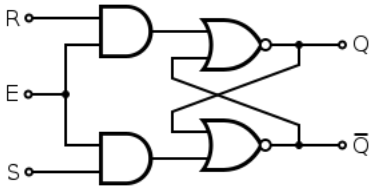


We need implementations that deliver repeatable timing

broman@eecs.berkeley.edu

The good news

Fortunately, electronics technology delivers highly reliable and precise timing



The bad news…

The chip architectures introduces highly non-deterministic behavior (e.g., using caches, pipelines etc.).

We need to rethink the microarchitecture

- **Pipelining**
- **Memory hierarchies**
- **I/O (DMA, interupts)**
- **Power management**
- **On-chip communication**
- **Reseource sharing (e.g., in multicore)**

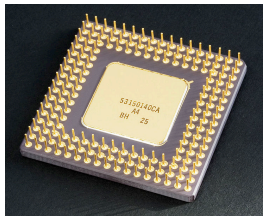
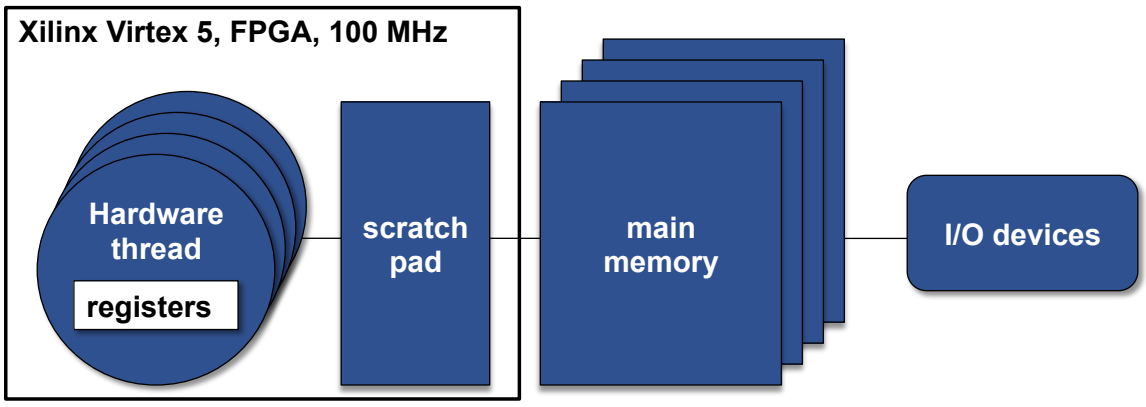


Photo by Andrew Dunn, 2005

Our Current PRET Architecture

broman@eecs.berkeley.edu

PTARM, a soft core on Xilinx Virtex 5 FPGA



Thread-interleaved Pipeline

4 threads, 5 stage pipeline

Scratchpad shared among threads

1 thread cycle for load/store

DRAM main memory, separate banks per thread

Load 4 thread cycle
Store 1-2 thread cycles
(1 thread cycle = 4 processor cycles)

Timing Aware Compilation (Work-in-progress)

broman@eecs.berkeley.edu

Modeling Language

Simulink, Ptolemy, Modelica, Other DSLs

Programming Language

Real-time Concurrent C (Gehani and Ramamritham, 1991)

PRET-C (Andalam et al., 2009)

WCC (Falk and Lokuciejewski, 2010)

Assembly Language

Timing Aware Compilation (Work-in-progress)

broman@eecs.berkeley.edu

Modeling Language

Simulink, Ptolemy, Modelica, Other DSLs

Programming Language

Extended with timing semantics

Java, ptC, Haskell, C/C++

Intermediate Language

Expose timing constructs and WCET

PRETIL

Abstracting away the memory allocation (scratchpad)

Assembly Language

PRET ISA

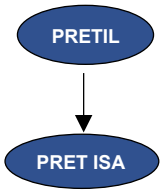
Some of the main challenges

broman@eecs.berkeley.edu

Relating real-time to execution-time

Execution time in clock cycles – oscillators not precise enough

➔ IEEE1588 clock synchronization (needs WCET margin)



Compiling with execution time

```
runfor(10ms){
  <code1>
}
```

(1) Estimate WCET of ASM program
(2) WCET-aware compilation to minimize WCET

Scratchpad allocation is now a compiler problem

➔ Optimization problem with timing constraints. Options:
(1) For a given frequency, optimize (average case) performance
(2) Find lowest frequency fulfilling constraints

Making time independent of the hardware

For the same ISA, different platform parameters affect timing (Scratchpad size, clock frequency, memory latency, etc.)

➔ Compiler generates a certificate – checked at load time

Part I
Cyber-Physical Systems

Part II
Modelyze - a host language for equation-based languages

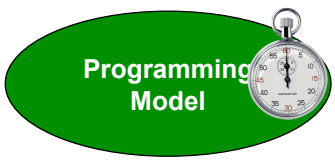
Part III
Precision Timed Infrastructure - making time an engineering abstraction

Conclusions

Main takeaway points (PRET)

Time is a correctness factor – not just a performance (quality) factor

Today, timing behavior is a property of the *realizations* of a software system



The PRET approach aims at making time an engineering abstraction *within* the *programming model*

Thank you for listening!