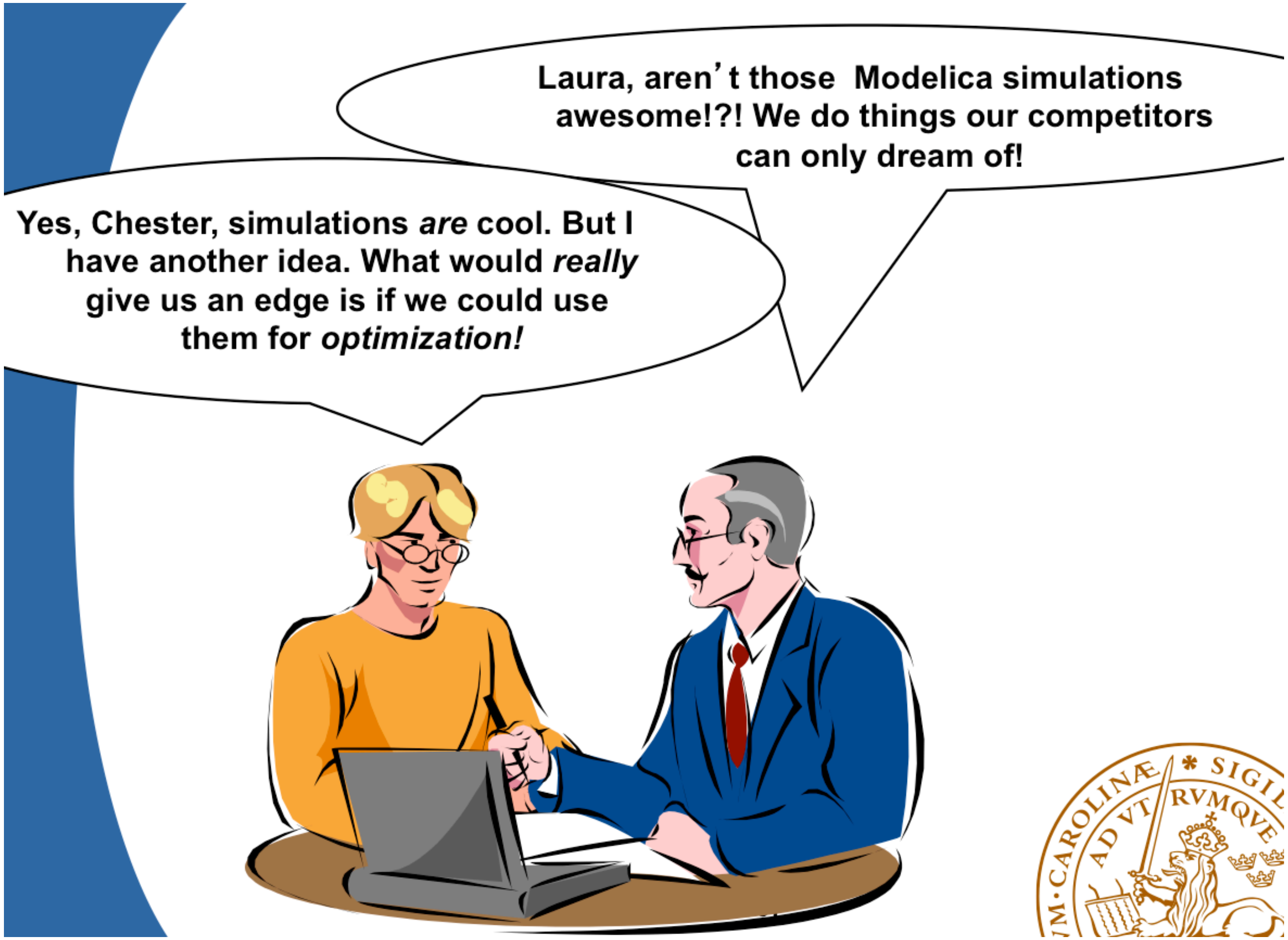
Vertical Integration in Tool Chains for Modeling Simulation and Optimization of Large-Scale Systems

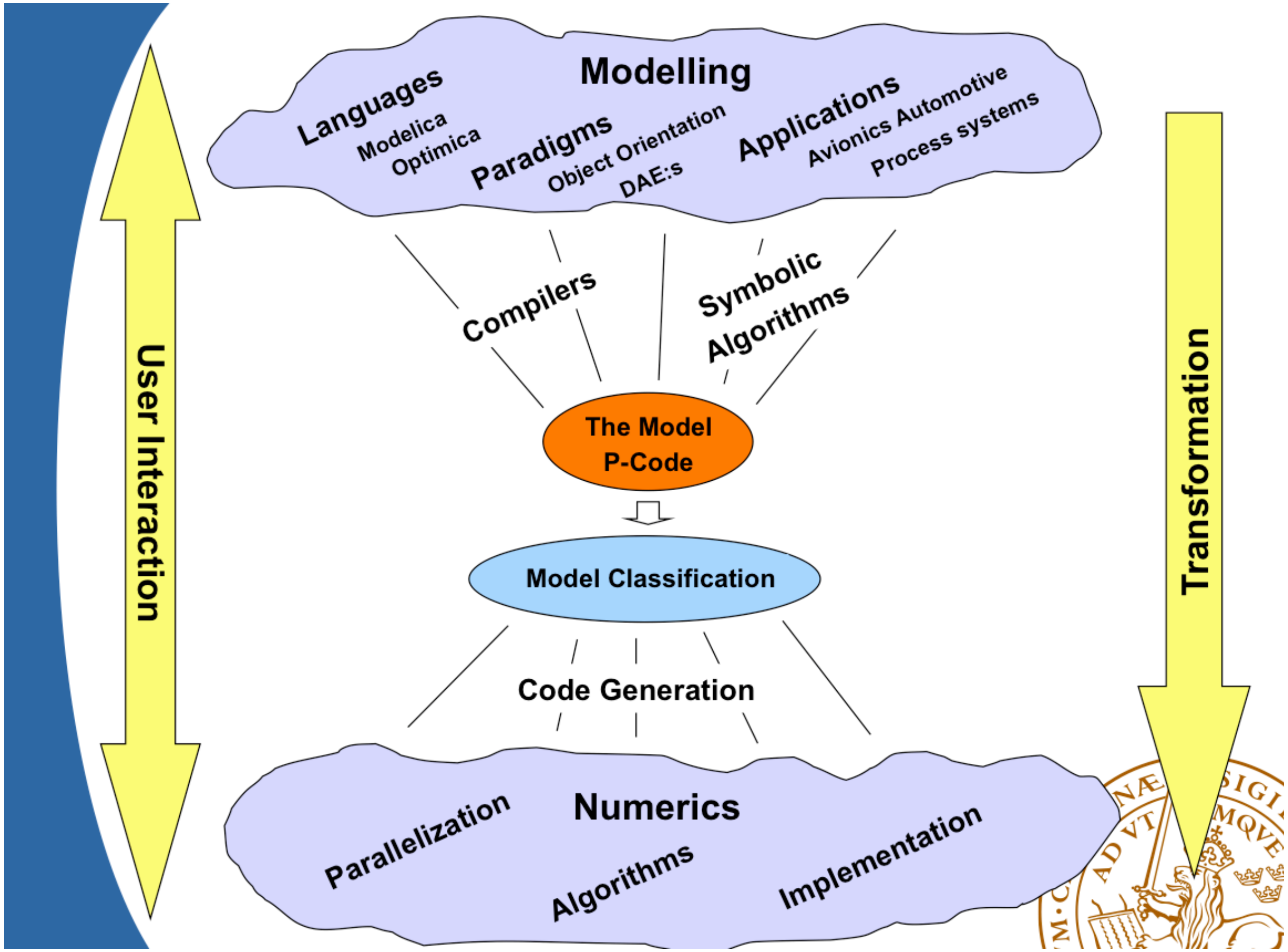
Johan Åkesson, Modelon AB/Lund University

Thanks to
Joel Andersson, Niklas Andersson, Magnus Gäfvert, Staffan Haugwitz,
Görel Hedin, Per-Ola Larsson, Alexandra Lind, Kilian Link,
Fredrik Magnusson, Elin Sällberg, Stephane Velut

In 2006…


Laura, aren't those Modelica simulations awesome!?! We do things our competitors can only dream of!
Yes, Chester, simulations *are* cool. But I have another idea. What would *really* give us an edge is if we could use them for *optimization!*

The Landscape



Outline

- Modelica
- Application examples
- Extension example
- Interface example
- Towards a vertically integrated tool chain
- Challenges

What is Modelica?

- A language for modeling of complex heterogeneous physical systems
  - Open language
    - Modelica Association (www.modelica.org)
  - Several tools supporting Modelica
    - Dymola
    - OpenModelica (free)
    - MosiLab
    - Scilab/Scicos (free)
  - Extensive (free) standard library
    - Mechanical, electrical, thermal etc.

Key Features of Modelica

- Declarative equation-based modeling
  - Text book style equations
- Multi-domain modeling
  - Heterogeneous modeling
- Object oriented modeling
  - Inheritance and generics
- Software component model
  - Instances and (acausal) connections
- Graphical and textual modeling

A Simple Modelica model

Differential equation

$$\dot{x}(t) = ax(t) + bu(t)$$

Class definition

Parameter declaration

Variable declaration

Initialization

Derivative operator

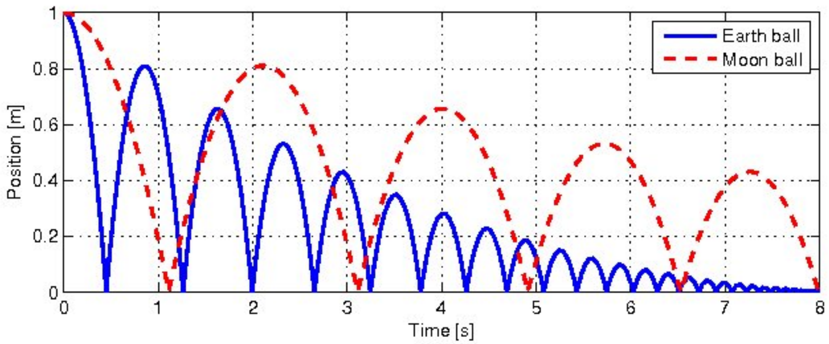
Equation

```
model FirstOrder
  input Real u;
  parameter Real b = 1;
  parameter Real a = -1;
  Real x(start=1);
equation
  der(x) = a*x + b*u;
end FirstOrder;
```

Hybrid modeling

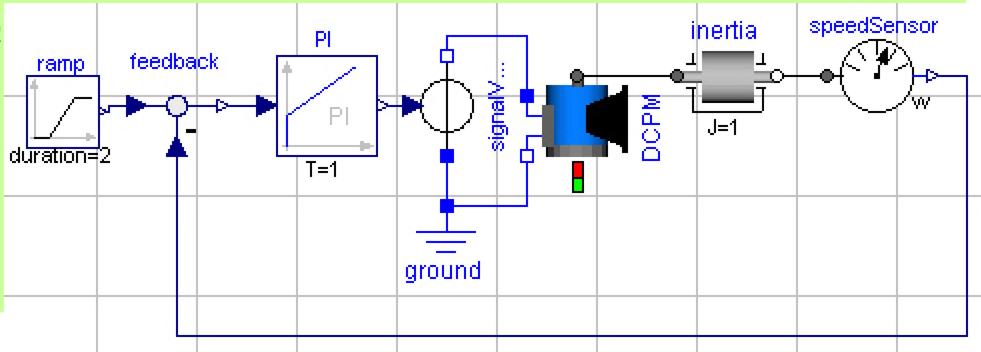
```
class BouncingBall //A model of a bouncing ball
  parameter Real g = 9.81; //Acceleration due to gravity
  parameter Real e = 0.9; //Elasticity coefficient
  Real pos(start=1); //Position of the ball
  Real vel(start=0); //Velocity of the ball
equation
  der(pos) = vel;   // Newtons second law
  der(vel) = -g;
  when pos <=0 then
    reinit(vel,-e*pre(vel));
  end when;
end BouncingBall;
```

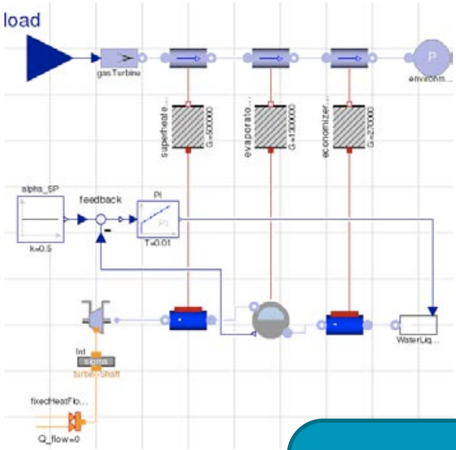
```
class BBex
  BouncingBall eBall;
  BouncingBall mBall(g=1.62);
end BBex;
```



Graphical Modeling

```
model MotorControl
  Modelica.Mechanics.Rotational.Inertia inertia;
  Modelica.Mechanics.Rotational.Sensors.SpeedSensor speedSensor;
  Modelica.Electrical.Machines.BasicMachines.DCMachines.DC_PermanentMagnet DCPM;
  Modelica.Electrical.Analog.Basic.Ground ground;
  Modelica.Electrical.Analog.Sources.SignalVoltage signalVoltage;
  Modelica.Blocks.Math.Feedback feedback;
  Modelica.Blocks.Sources.Ramp ramp(height=100, startTime=1);
  Modelica.Blocks.Continuous.PI PI(k=-2);
equation
  connect(inertia.flange_b, speedSensor.flange_a);
  connect(DCPM.flange_a, inertia.flange_a);
  connect(speedSensor.w, feedback.u2);
  connect(ramp.y, feedback.u1);
  connect(signalVoltage.n, DCPM.pin_ap);
  connect(signalVoltage.p, ground.p);
  connect(ground.p, DCPM.pin_an);
  connect(feedback.y, PI.u);
  connect(PI.y, signalVoltage.v);
end MotorControl;
```





A Modelica-based Tool Chain

Flattening of Modelica source code
Compiler front-end

→ Unstructured Flat DAE

→ Symbolic maniulation
Index reduction
Analytic solution of simple equations

→ Transformed flat DAE

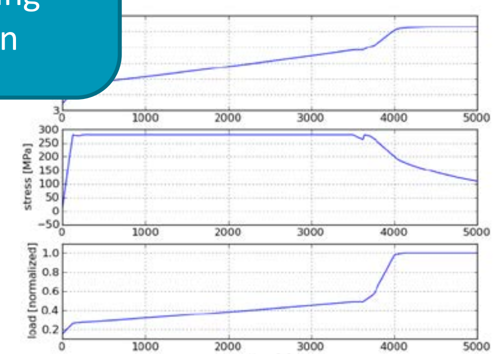
→ Code generation
Residual equations
Analytic Jacobians

→ C code

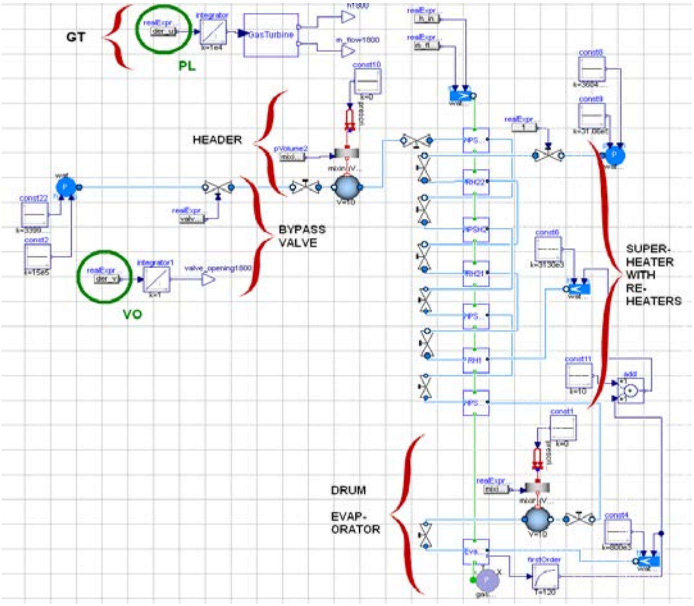
→ Numerical solvers
NLP algorithms
Integrators

→ Solution profiles

→ Result
Post processing
Visualization



Industrial Application I
Power Plant Start-up Optimization



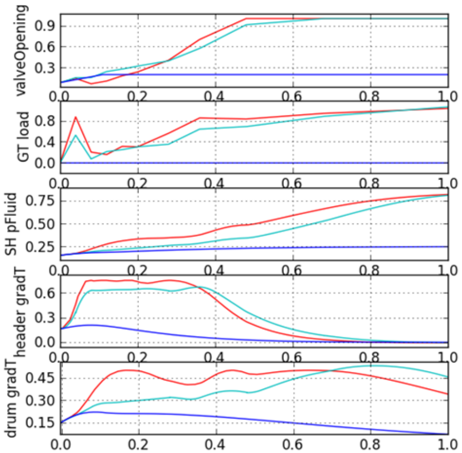
- Start-up optimization of combined cycle power plants
- Reduce start-up time
- Model-based optimization
- Siemens AG, LU, Modelon collaboration

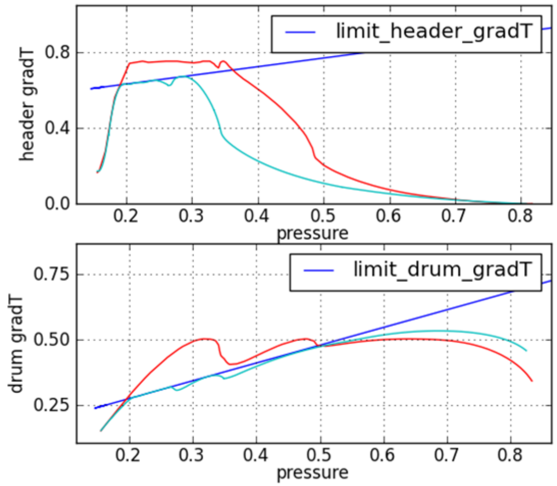
Continuous time states: **39**

Scalar equations: **569**

Algebraic variables: **530**

NLP equations: **26824**





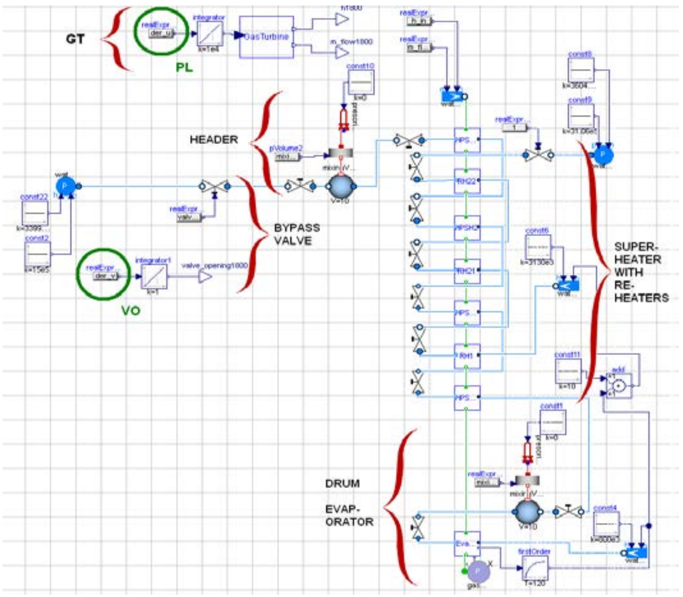
Industrial Application I
Power Plant Start-up Optimization

- ☺ Design-patterns from Modelica media model libraries applied to optimization-friendly models
- ☺ Intuitive high-level descriptions of dynamic optimization problem appreciated by users – a vehicle for communicating ideas

- ☹ Large effort to develop models suitable for optimization
- ☹ Scaling of problem significantly more challenging than in simulation
- ☹ Convergence and robustness of numerical algorithms

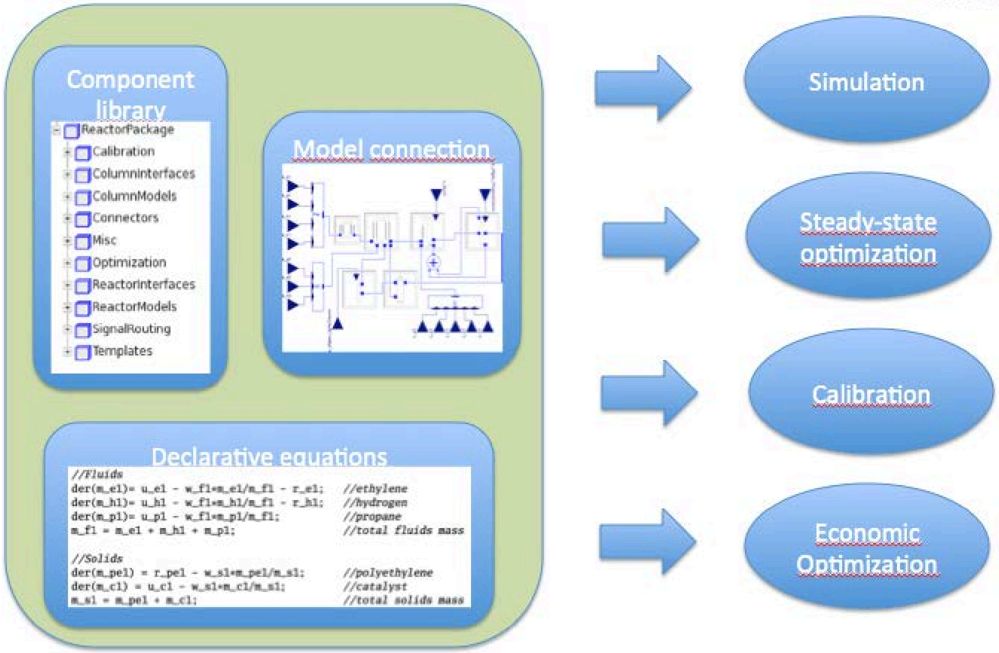
Lessons learnt

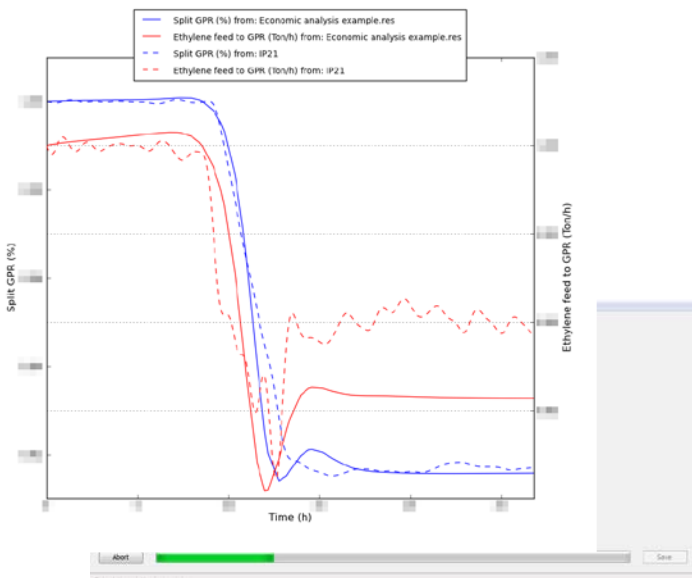
- Modeling for optimization is significantly different from modeling for simulation
- Numerical optimization algortihm is significantly less robust than simulation algorithm
- Scaling of problem and initial guesses have major impact



Industrial Application II
Grade Changes in Polyethylene Production

- Optimization of economics of polyethylene grade changes
- Model calibration to data
- Modeling with Modelica and Optimica
- Development of end-user GUI
- PIC-LU – Lund University and Borealis





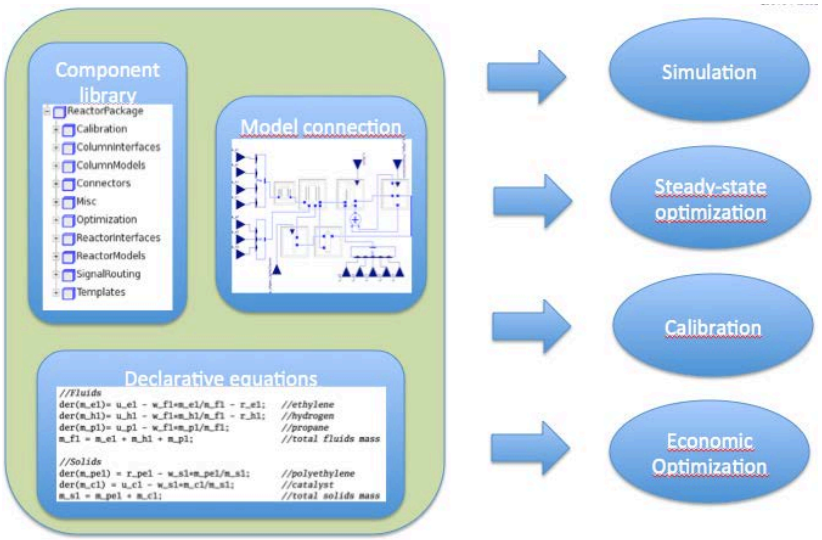
Industrial Application II
Grade Changes in Polyethylene Production

- ☺ Model reuse across different computations
- ☺ High-level model and optimization problem formulation enabled promoted focus on problem formulation
- ☺ Custom GUI in Python appreciated by end-users

- ☹ Careful manual scaling of problem required for convergence
- ☹ Difficult to tailor collocation optimization formulation to problem description
- ☹ Non-standard economic cost difficult to handle

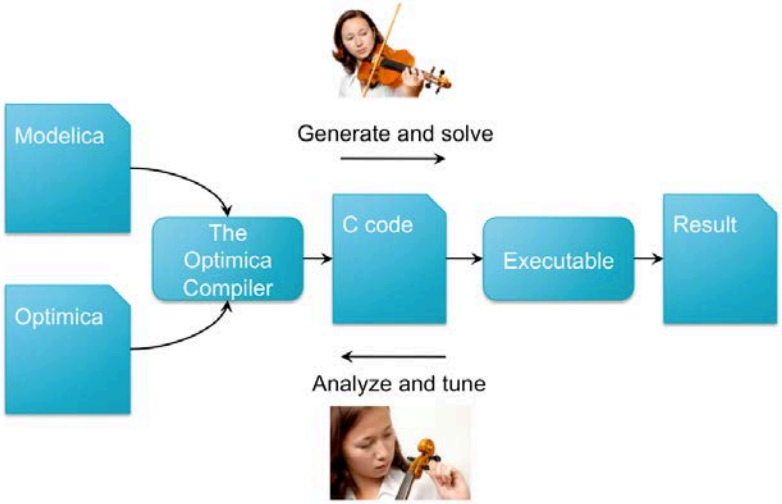
Lessons learnt

- Significant advantages from Modelica technology – same model used for steady-state, dynamic simulation, calibration and optimization
- Increased interaction with discretization sometimes important



Extension Example – Optimica

- High-level description of optimization problems
  - Steady-state
  - Dynamic
- Extension to Modelica
  - Optimization of physical models




```
optimization VDP_Opt(objective=cost(finalTime),
     startTime=0,
     finalTime(free=true, initialGuess=1))
  VDP vdp(u(free=true,initialGuess=0.0));
   Real cost (start=0);
equation
   der(cost) = 1;
constraint
   vdp.x1(finalTime) = 0;
   vdp.x2(finalTime) = 0;
   vdp.u >= -1; vdp.u <= 1;
end VDP_Opt;
```

$$\min_{u(t),\,p} \Psi(\bar{z}, p)$$

subject to the dynamic system

$$F(\dot{x}(t), x(t), y(t), u(t), p, t) = 0, \quad t \in [t_0, t_f]$$

and the constraints

$$c_{ineq}(x(t), y(t), u(t), p) \leq 0, \quad t \in [t_0, t_f]$$
$$c_{eq}(x(t), y(t), u(t), p) = 0, \quad t \in [t_0, t_f]$$
$$c^p_{ineq}(\bar{z}, p) \leq 0$$
$$c^p_{eq}(\bar{z}, p) = 0$$

where

$$\bar{z} = [x(t_1), \ldots, x(t_{N_p}), y(t_1), \ldots, y(t_{N_p}), u(t_1), \ldots, u(t_{N_p})]^T, \quad t_i \in [t_0, t_f]$$

Extension Example – Optimica

- ☺ High-level problem descriptions promote focus on formulation rather than encoding
- ☺ New users without optimization experience quickly gets up to speed
- ☺ Model reuse for different usages
- ☺ Automatic model transformation reduce user effort

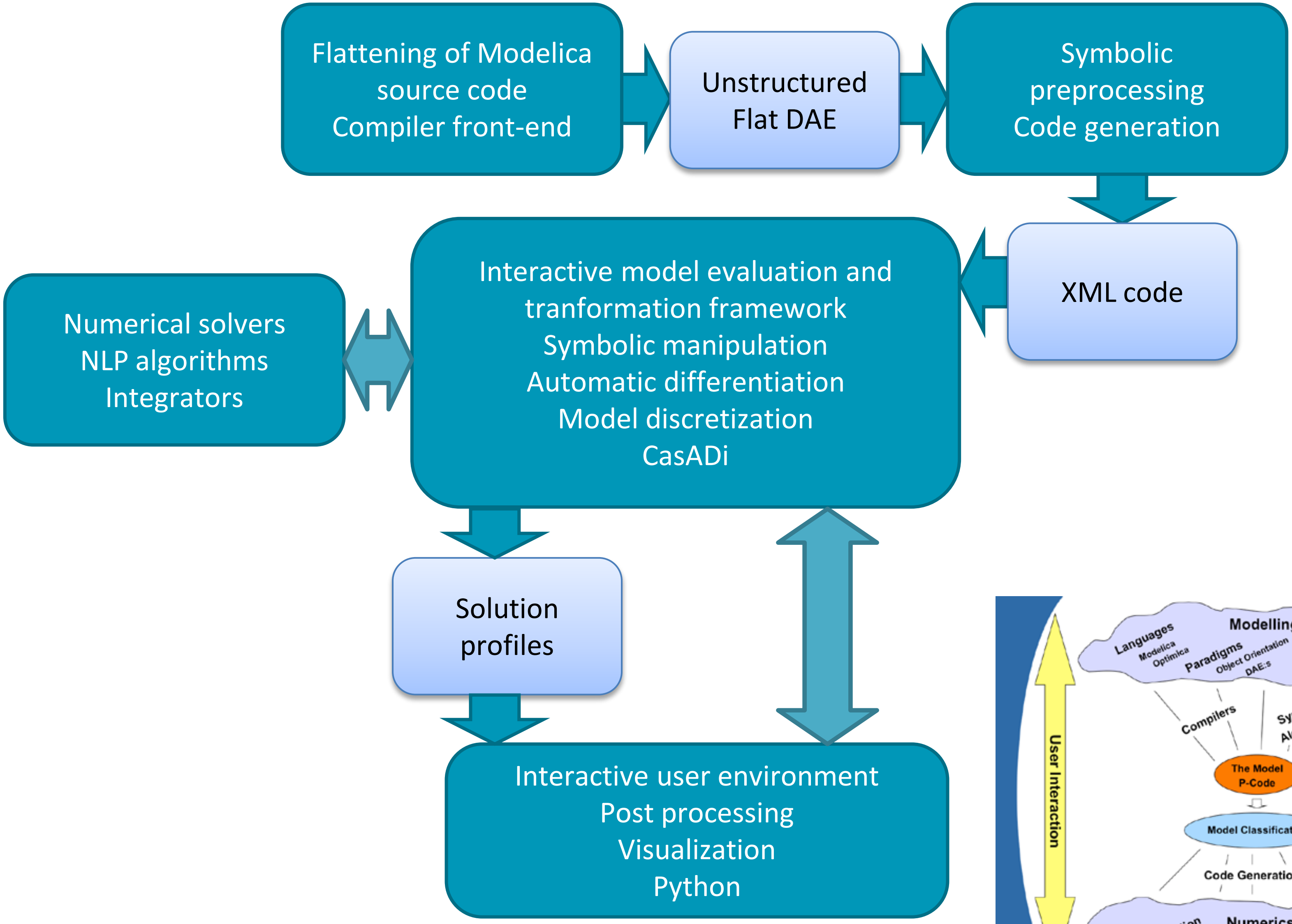
- ☹ Tailoring of problem discretization difficult, but sometimes needed
- ☹ Power-users of dynamic optimization tools feel constrained

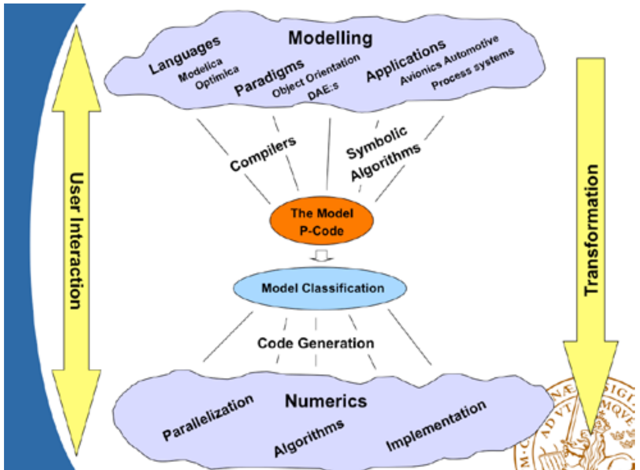
Lessons learnt

- High-level descriptions make optimization technology available to non-experts
- Automatic model transformation reduces design cycle times
- Modern compiler construction technology is accessible to non-experts (e.g., JastAdd)

```
optimization VDP_Opt(objective=cost(finalTime),
     startTime=0,
     finalTime(free=true, initialGuess=1))
  VDP vdp(u(free=true,initialGuess=0.0));
   Real cost (start=0);
equation
   der(cost) = 1;
constraint
   vdp.x1(finalTime) = 0;
   vdp.x2(finalTime) = 0;
   vdp.u >= -1; vdp.u <= 1;
end VDP_Opt;
```

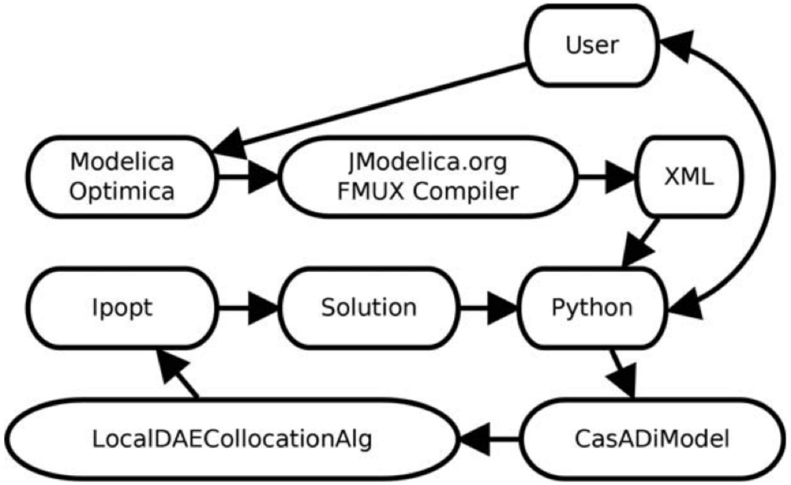
Towards a vertically integrated toolchain

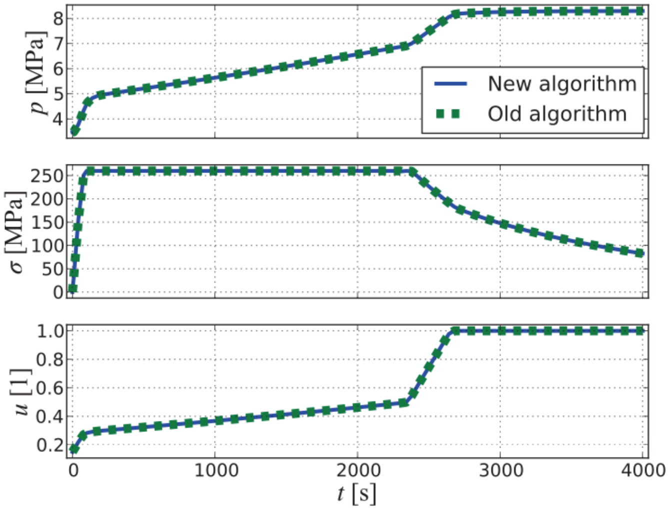




Interfacing Example – Modelica, XML Models and CasADi

- Replace C implementation of a collocation algorithm
- Intermediate symbolic model format in XML
- Decreased solution times by an order of magnitude
- Decreased implementation time by an order of magnitude
- Significantly increased flexibilty
- Tailoring to specific problems





	Off-line	On-line	Total	Iterations
New alg.	4.9	3.0	7.9	79
Old alg.	13.2	23.9	37.2	75

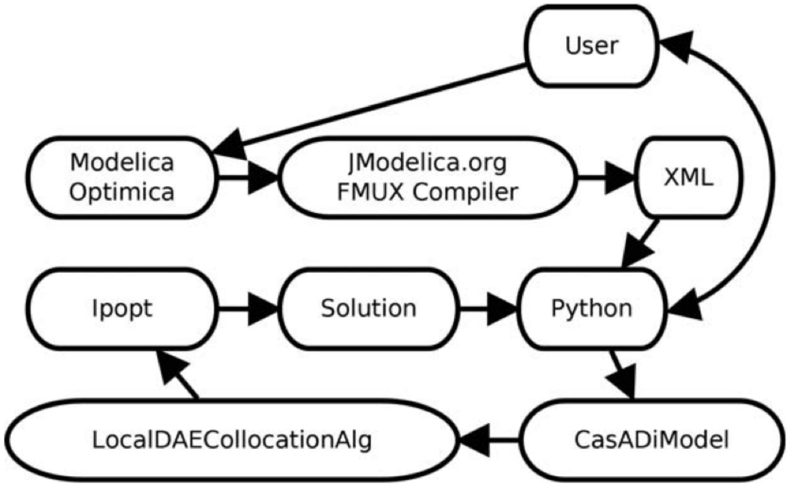
Interfacing Example – Modelica, XML Models and CasADi

- ☺ Rapid prototyping with interactive model evaluation and transformation frameworks
- ☺ Flexibility to tailor model descretization to problem formulation
- ☺ Inspiration for future versions of Optimica

- ☹ Partial problem formulation in high-level format
- ☹ Some of the overview lost when parts of the problem is formulated in Modelica/Optimica some part is in scripting language

Lessons learnt

- Interactive model transformation powerful
- Symbolic model exchange format needed (standardization on-going)
- High performance and flexibility can be combined



Challenges

- *How do we make advanced algorithms in systems design in general and in optimization in particular PhD-free?*
- *How do we combine declarative modeling languages with ideas from interactive model transformation/evaluation frameworks?*
- *How do we propagate consistent error/diagnostics through the tool chain?*
- *Open interfaces and interoperability, FMI and extensions*
- *Classify models applicable to different solution algorithms*

Conclusions

- In users' perception, current optimization algorithms for large-scale non-linear dynamic systems requires high level of expertise
- Very different cultures and best practices in simulation and optimization communities – expectation management
- Users sometimes need to/desire to to interact with both mathematical model and solution algorithm implementation
- Challenges in usability and robustness of numerical algorithms
- Challenges in vertically integrated tool chains – languages and open interfaces and tool decoupling

Thank you!

Questions, comments?