Modia – A Prototyping Platform for Next Generation Modeling And Simulation Based on Julia

Dr Hilding Elmqvist

CEO Mogram AB and Technical Fellow Modelon AB

Prof Martin Otter

DLR, Institute of System Dynamics and Control

Outline

- Motivation - The *Modia* project
- Introduction to *Modia* language
- *Modiator* web app
- *ModiaMedia*
- Symbolic algorithms
- Summary

Modelica Challenges

- Modelica is powerful (equations, objects, connections)
  - Although static, requiring recompilation if:
    - An array dimension is changing
    - A component class is changing
    - A medium is changing
- Modelica algorithms and functions lack functionalities:
  - Modern data structures
  - Parallelization
  - …
- It is possible to build complex system models, but:
  - Sometimes hard to understand models (3D, media/fluid models, …)
  - Translation should be faster
  - Simulation should be faster

Mogram

Innovation platform - *Modia*

Based on modern language – Julia

- Dynamic typing, Matlab-like notation
- Static typing, efficient, data structures (as C++)
- Multiple dispatch
- Metaprogramming
  - for domain specific language extensions
  - for symbolic processing
- Just-in-time compilation

Open source project consisting of several Julia packages (github.com/ModiaSim)

Modia	Equation-based modeling
Modiator	2D/3D model editor
ModiaMath	Simulation environment
Modia3D	3D geometry and 3D mechanics
ModiaMedia	Thermodynamic property models
Modelia	Modelica model importer (partial)

Contributors:
Hilding Elmqvist, Toivo Henningsson, Martin Otter, Andrea Neumayr, Oskar Åström, Chris Laughman

Mogram

Connectors and Components - Electrical

Modelica

```
@model Pin begin
  v=Float()
  i=Float(flow=true)
end

@model OnePort begin
  p=Pin()
  n=Pin()
  v=Float()
  i=Float()
@equations begin
  v = p.v - n.v # Voltage drop
  0 = p.i + n.i # KCL within component
  i = p.i
  end
end

@model Resistor begin # Ideal linear electrical resistor
  @extends OnePort()
  @inherits i, v
  R=1 # Resistance
@equations begin
  R*i = v
  end
end
```

```
connector Pin
  Modelica.SIunits.Voltage v;
  flow Modelica.SIunits.Current I;
end Pin;

partial model OnePort
  SI.Voltage v;
  SI.Current i;
  PositivePin p;
  NegativePin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end OnePort;

model Resistor
  parameter Modelica.SIunits.Resistance R;
  extends Modelica.Electrical.Analog.Interfaces.OnePort;
equation
  v = R*i;
end Resistor;
```

Mogram DLR

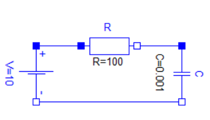
Elmqvist/Henningsson/Otter 2017: Innovations for Future Modelica

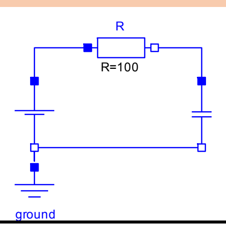
Coupled Models - Electrical Circuit

Modelica

```
@model LPfilter begin
  R = Resistor(R=100)
  C = Capacitor(C=0.001)
  V = ConstantVoltage(V=10)
@equations begin
  connect(R.n, C.p)
  connect(R.p, V.p)
  connect(V.n, C.n)
  end
end
```

```
model LPfilter
  Resistor R(R=100);
  Capacitor C(C=0.001);
  ConstantVoltage V(V=10);
  Ground ground;
equation
  connect(R.n, C.p);
  connect(R.p, V.p);
  connect(V.n, C.n);
  connect(V.n, ground.p);
end Lpfilter;
```

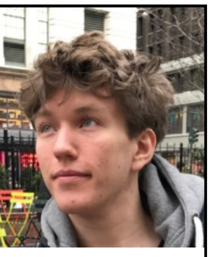


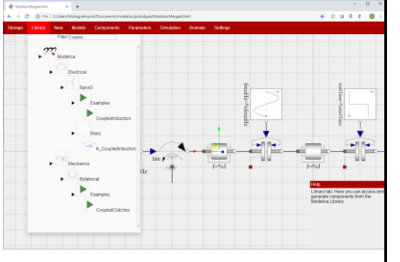


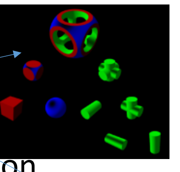
Mogram DLR

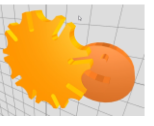
Modiator – web app

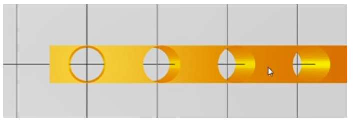
- Summer 2019 prototype
- Summer intern: Oskar Åström
- Joint project between Modelon and Mogram
- Cooperation with Martin Otter, DLR
- Modelica diagrams
- Exploring fundamentals
  - CSG – Constructive Solid Modeling
  - Shape parametrization
- Focus: 3D model composition and animation
  - Modia3D
  - Modelica…MultiBody







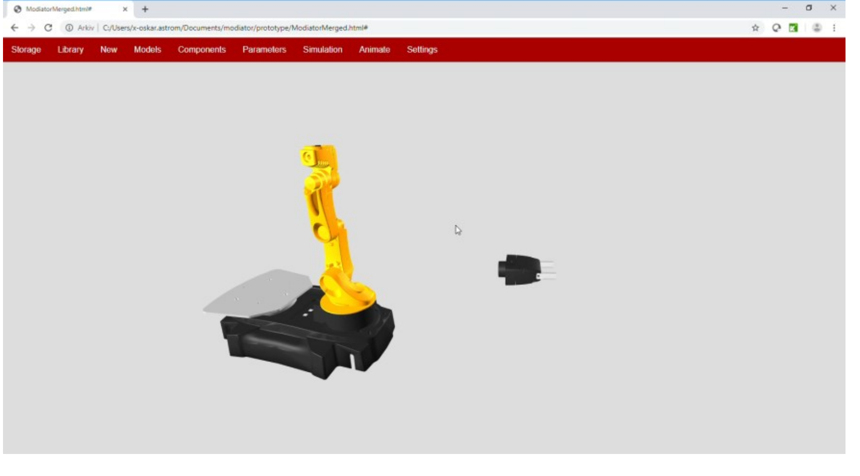




Mogram DLR

3D model composition

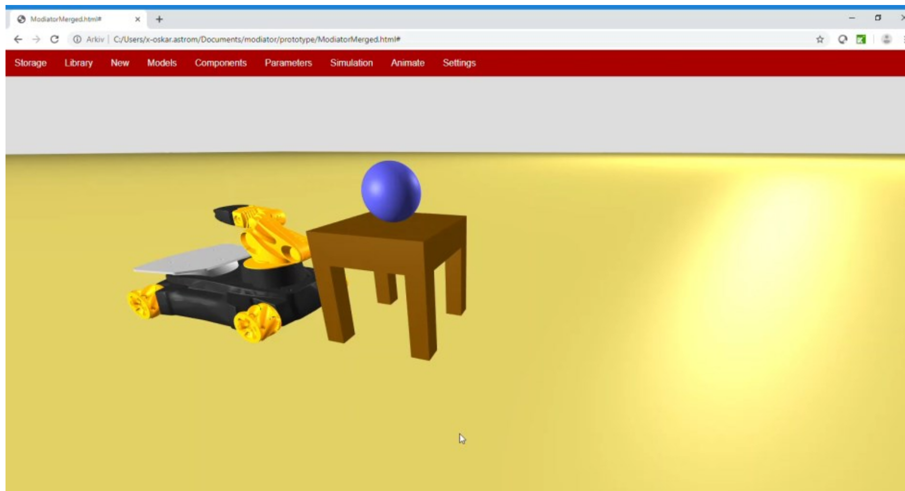
- Mechanism composition
- Introducing joints
- Parametrization
- Immediate kinematic animation
- Exploded view



Mogram DLR

3D Animation

- Generate *Modia3D* model
  - Determine properties from geometries (mass, ...)
  - 3D mechanics algorithms
  - Collision handling
  - Fast translation
- Client/server communication between web app and Julia
- Simulate
- Animate result in *Modiator*

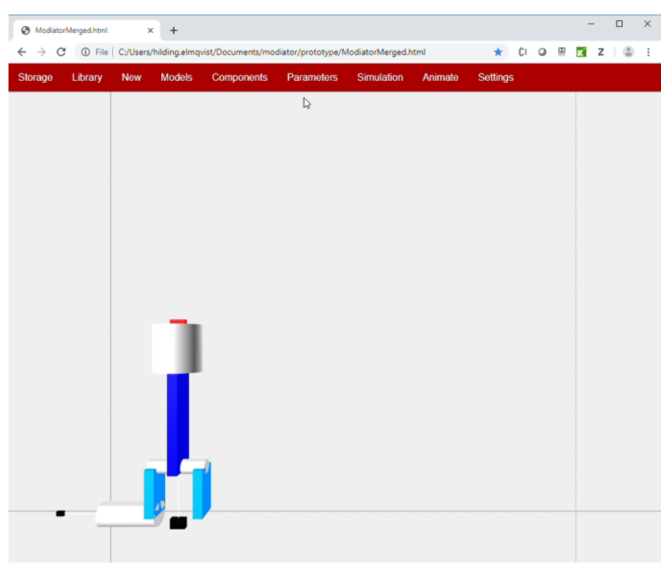


Mogram

DLR

Modelica Multibody 3D parametric preview

- Kinematic animation
- Parametric animation
- Spanning tree view
- Interpretation of Modelica AST
- Evaluation of Modelica expressions



Mogram

DLR

ModiaMedia - Thermodynamic property models

Developers: Martin Otter (DLR), Hilding Elmqvist (Mogram), Chris Laughman (MERL); Paper at Modelica'2019

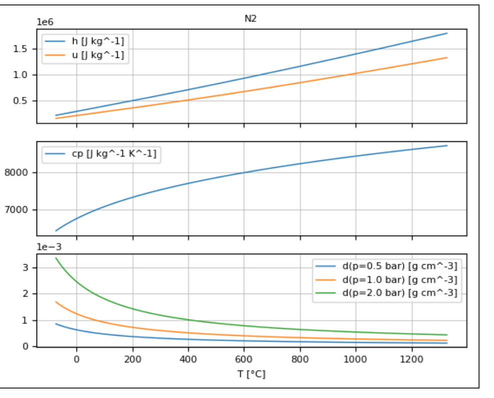
```
using ModiaMedia

Medium = getMedium("N2")       # dictionary

p = 1e5
T = 300.0
state = setState_pT(Medium, p, T)  # construct
setState_pT!(state, 2*p, 2*T)       # update

d = density(state)               # get other properties
h = specificEnthalpy(state)

listMedia()                  # list all supported media
standardPlot(Medium)    # plot Medium
```



- **Much simpler and more powerful as Modelica.Media**
- Fluid network: **state** propagated/updated along connection structure (Medium defined at **one** state instance)

Mogram DLR

Symbolic Algorithms

- For $\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, t)$
- Can be used directly in current Modelica tools

Mogram DLR

Modelica translation today - flattening

- Object-oriented modeling approach
  - allows building large models with millions of equations
- Semantic specification is based on **flattening**
  - **i.e. cloning variables and equations** of each component instance
- And most tools also expands matrix equations

Negative consequences:

- A lot of **memory** is needed for variables and equations during translation
- **Translation time** is unnecessary long
  - same analysis (flattening, symbolic processing, etc) is performed repeatedly for each instance of a component
- **C-code** gets large and **compilation** takes long time

Remedy: Separate Translation

- Parts of the equations of a component
  - are always executed in the **same order** and with the **same causality**
  - independently of how the component is connected
- Such sequences of equations can be put into **functions**
  - which are reused for all components of the same class
  - less C-code gives **shorter compilation time**
- Finding such sequences can be made once for each model class
  - **faster translation** and **less memory use**

Component Model Equations

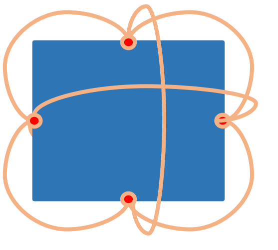
DAE:

$f(\dot{x}, x, c_p, c_f, u, y, v, p)$=0

- x – differentiated variables
- c_p – potentials of the connectors
- c_f – flows and streams of the connectors
- u – inputs
- y – outputs
- v – other variables
- p – parameters
- dim(f) = dim($\dot{x}$)+dim(c_p)+dim(y)+dim(v)

Generic environment of model:

- Generic environment needs to relate all connector variables
- $g(c_p, c_f, u, y)$=0
  - dim(g) = dim(c_f)+dim(u)
  - g has full incidence
- Might also contain derivatives

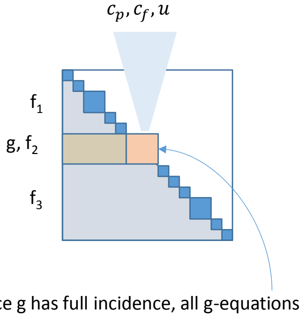


Mogram DLR

Model equations partitioning

- First blocks (always same causality, use function):
  - $\dot{x}_1, y_1, v_1 = f_1(\boldsymbol{x}, \boldsymbol{p})$
- Middle block (f_2 kept as equations):
  - $g(c_p, c_f, u, y)$=0
  - $f_2(\dot{\boldsymbol{x}}_1, \dot{x}_2, \boldsymbol{x}, c_p, c_f, u, \boldsymbol{y}_1, y_2, \boldsymbol{v}_1, v_2, \boldsymbol{p})$=0
- Last blocks (always same causality, use function):
  - $\dot{x}_3, y_3, v_3$=$f_3(\dot{\boldsymbol{x}}_1, \dot{\boldsymbol{x}}_2, \boldsymbol{x}, \boldsymbol{c}_p, \boldsymbol{c}_f, \boldsymbol{u}, \boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{p})$

Perform BLT on f and g function incidences

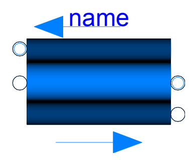


Since g has full incidence, all g-equations will appear in the same block

Known variables marked in **bold face**

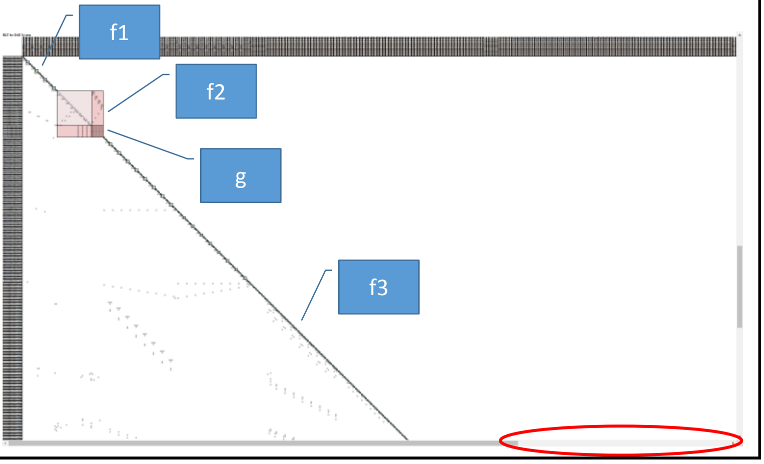
Mogram DLR

Example: Heat exchanger model

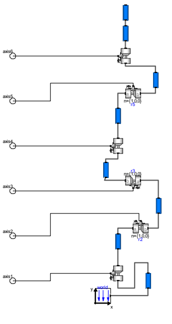


MSL BasicHX

- flow models close to connectors
- **10 spatial segments**
- 50 dynamic states
- 514 equations
- dim(f2) = 24



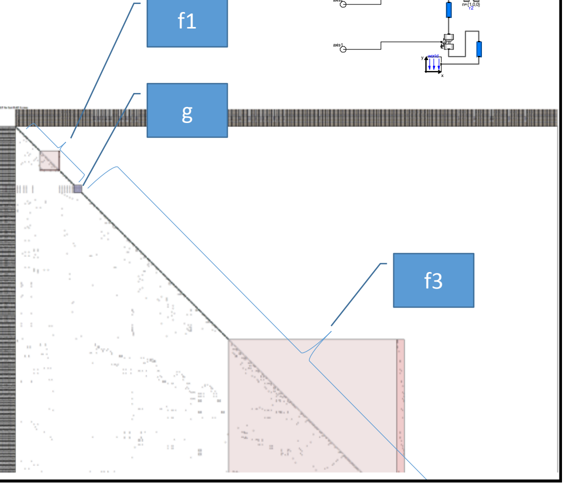
Example: Multibody Robot model



MSL Robot with der(phi)

- With der(phi) in g(…)
  to enable connecting dampers
- 12 dynamic states
- 391 equations
- dim(f2) = 0

- Modia3D is a manual
  implementation of this approach



Separate Translation - Summary

- Systematic method for splitting model equations into acausal and causal partitions
  - User does not need to consider which equations can be moved to functions
  - Local index reduction is performed
  - Global index reduction requires automatic differentiation of the functions
- Limited testing shows that substantial part of the equations can be moved to separately compiled functions
- Less time and memory for both translation and simulation
- This approach could be combined into a generalized FMU.

Mogram DLR

Index Reduction of Array Equations

Core algorithm in Modelica tools

- Structural algorithm to reduce DAE index to 0 (= solve state constraints)
- Often: *Pantelides 1988*.
- Map scalar equations ⟶ scalar equations (array properties lost during transformation)

New algorithm *Otter, Elmqvist 2017 (section 3)*

- Generalization of *Pantelides 1988*
- Map array equations ⟶ array equations
- More efficient machine code possible

Example

$$
\begin{aligned}
\mathbf{r} &= \mathbf{n}s \\
\mathbf{v} &= \dot{\mathbf{r}} \\
m\dot{\mathbf{v}} &= \mathbf{f} + m\mathbf{g} + \mathbf{u} \\
0 &= \mathbf{n}\cdot\mathbf{f} \\
\mathbf{u} &= -(cs + d\dot{s})\mathbf{n}
\end{aligned}
$$

BLT Block 1	*solve for*
$\mathbf{u} = -(cs + d\dot{s})\mathbf{n}$	$\mathbf{u}$
BLT Block 2	
BLT Block 2.1	
$\mathbf{r} = \mathbf{n}s$	$s, \mathbf{r}$
BLT Block 2.2	
$\dot{\mathbf{r}} = \mathbf{n}\dot{s}$ $\mathbf{v} = \dot{\mathbf{r}}$	$\dot{s}, \dot{\mathbf{r}}, \mathbf{v}$
BLT Block 2.3	
$\ddot{\mathbf{r}} = \mathbf{n}\ddot{s}$ $\dot{\mathbf{v}} = \ddot{\mathbf{r}}$ $m\dot{\mathbf{v}} = \mathbf{f} + m\mathbf{g} + \mathbf{u}$ $\mathbf{0} = \mathbf{n}\cdot\mathbf{f}$	$\ddot{s}, \ddot{\mathbf{r}}, \dot{\mathbf{v}}, \mathbf{f}$

Mogram DLR

Tearing with retained solution space

Core algorithm in Modelica tools

- Reduce the size of algebraic equation systems
- Reduce number of states

$$\mathbf{0} = \boldsymbol{g}(\boldsymbol{z}) \longrightarrow \begin{array}{l} \boldsymbol{z}_e := \boldsymbol{g}_e(\boldsymbol{z}_e, \boldsymbol{z}_t) \\ 0 = \boldsymbol{g}_r(\boldsymbol{z}_e, \boldsymbol{z}_t) \end{array}$$

$\boldsymbol{z} = [\boldsymbol{z}_e, \boldsymbol{z}_t]$
solve explicitly as much as possible, without changing solution space

Example

$$\begin{array}{l} z_1 = f_1(z_4) \\ z_2 = f_2(z_1, z_5) \\ z_3 = f_3(z_2, z_1) \\ z_4 = f_4(z_3, z_2) \end{array} \rightarrow \begin{array}{l} \text{input: } z_4 \\ \text{output: } r \\ z_1 := f_1(z_4) \\ z_2 := f_2(z_1, z_5) \\ z_3 := f_3(z_2, z_1) \\ r = z_4 - f_4(z_3, z_2) \end{array}$$

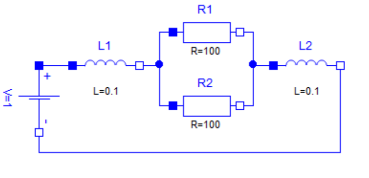
New algorithm:

Otter, Elmqvist 1999 (unpublished) +
Bender, Fineman, Gilbert, Tarjan 2016 (incremental cycle detection in DAGs)
→ *Otter, Elmqvist 2017 (section 4.6)*

O(n) ≤ tearing ≤ O(nm)
Example: Loop with 1 million equations → 1 equation (needs 2s)

Exact Removal of Singularities

Example



Modelica tools can fail on well-defined models:

- Structurally singular at compile-time
- Singular Jacobian at run-time

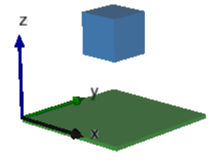
New algorithm *Otter, Elmqvist 2017 (section 5)*

- Extract all linear equations with Integer coefficients from DAE system (e.g.: $0 = i_1 + i_2$; $u_{rel} = u_2 - u_1$):
  $\rightarrow \mathbf{A} \cdot \boldsymbol{x} = \mathbf{B}$, $\mathbf{A} \in \mathbb{Z}^{na1 \times na2}$, $\mathbf{B} \in \mathbb{Z}^{na1 \times nb2}$
- Remove all singularities exactly!!!
- Use as pre-processing step

- Remove redundant equation:
  -L2.n.i - V.n.i = 0
- Make potentials well-defined by adding equation:
  L2.n.v = 0
- Make state constraints structurally visible by replacing
  -R1.p.i - R2.p.i - L1.n.i = 0
  with
  -L1.p.i + L2.p.i = 0

No dynamic state selection

Example



Modelica tools transform $\mathbf{0} = \mathbf{f}_1\,(\dot{\mathbf{x}}, \mathbf{x}, t)$
(conceptually) to index 0 form: $\dot{\boldsymbol{x}}_{red} = \mathbf{f}_2(\mathbf{x}_{red}, t)$

- Sparseness of $\mathbf{f}_1$ might get lost
- Might require dynamic state selection ($\mathbf{x}_{red}$ changed during simulation; might not work well)

New proposal *Otter, Elmqvist 2017*

Transform to special index 1 form

$$\begin{bmatrix} \mathbf{f}_d(\dot{\boldsymbol{x}}_{red}, \mathbf{x}_{red}, t) \\ \mathbf{f}_c(\mathbf{x}_{red}, t) \end{bmatrix} = \mathbf{0} \qquad \begin{bmatrix} \dfrac{\partial \mathbf{f}_d}{\partial \dot{\mathbf{x}}_{red}} \\ \dfrac{\partial \mathbf{f}_c}{\partial \mathbf{x}_{red}} \end{bmatrix} \text{ is regular}$$

- Sparseness is not changed
- No dynamic state selection

Free Body Rotation (with quaternions)

$$\boldsymbol{\omega} = 2\begin{bmatrix} q_4 & q_3 & -q_2 & -q_1 \\ -q_3 & q_4 & q_1 & -q_2 \\ q_2 & -q_1 & q_4 & -q_3 \end{bmatrix} \cdot \dot{\boldsymbol{q}}$$
$$\boldsymbol{\tau}(t) = \boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}$$
$$1 = \boldsymbol{q}^T\boldsymbol{q}$$

- Directly integrate equations (already in special index 1 form)
- Initialization/events:
  - new $\mathbf{x}_{red}$: use Dirac impulse
  - new $\dot{\boldsymbol{x}}_{red}$: use $\frac{d}{dt}(1 = \boldsymbol{q}^T\boldsymbol{q})$

Mogram DLR

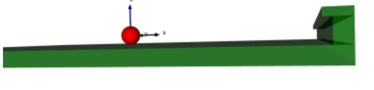
No dynamic state selection - examples

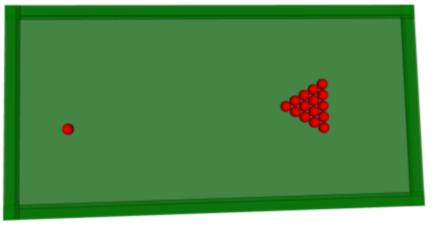
Body attached with spherical joint to ground
(= 7 equations)

$$\boldsymbol{\omega} = 2\begin{bmatrix} q_4 & q_3 & -q_2 & -q_1 \\ -q_3 & q_4 & q_1 & -q_2 \\ q_2 & -q_1 & q_4 & -q_3 \end{bmatrix} \cdot \dot{\boldsymbol{q}}$$
$$\boldsymbol{\tau}(t) = \boldsymbol{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \boldsymbol{I}$$
$$1 = \boldsymbol{q}^T\boldsymbol{q}$$

Modia about 40 % faster as a Modelica tool:

- Modelica: index 0 DAE, changing states, DASSL, 4000 model calls
- Modia : index 1 DAE, fixed states , IDA , 2700 model calls





16 free flying bodies à 13 states = 208 states
≈ 200 possible collision pairs

Mogram DLR

Multi-mode systems with impulses

Example

Previous multi-mode attempts of limited use:

- Changing structure can lead to index change + Dirac impulse
- Not supported by Modelica tools

New proposal

Benveniste, Caillaud, Elmqvist, Ghorbal, Otter, Pouzet 2019

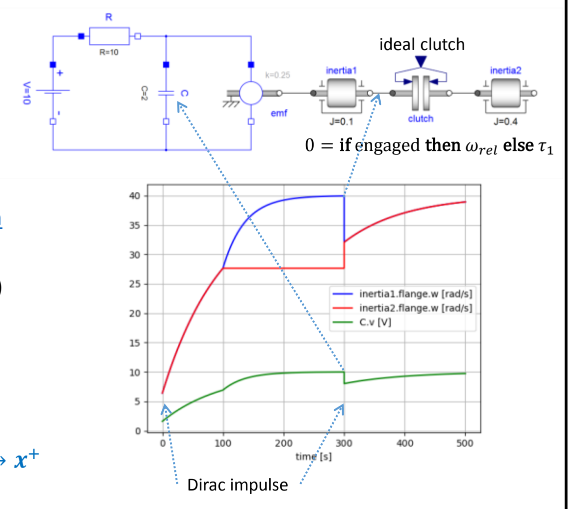
[Multi-Mode DAE Models: Challenges, Theory and Implementation]

Requirement: Special index 1 form linear in derivatives (+ other req.)

$$0 = \begin{bmatrix} \boldsymbol{A}(\boldsymbol{x},t)\dot{\boldsymbol{x}} + \boldsymbol{b}(\boldsymbol{x},t) \\ \boldsymbol{f}_c(\boldsymbol{x},t) \end{bmatrix} \quad (= \boldsymbol{f}_d(\dot{\boldsymbol{x}},\boldsymbol{x},t))$$

Compute $\boldsymbol{x}^+$ from $\boldsymbol{x}^-$ at t_{event}:

implicit Euler $h \rightarrow 0$ (hard) or $0 = \begin{bmatrix} \boldsymbol{A}(\boldsymbol{x}^+, t_{event})(\boldsymbol{x}^+ - \boldsymbol{x}^-) \\ \boldsymbol{f}_c(\boldsymbol{x}^+, t_{event}) \end{bmatrix} \rightarrow \boldsymbol{x}^+$



If index 0 ↔ 1: without re-compilation

Mogram

DLR

Summary

- Modelica needs better **scalability**
  - since users need to simulate more and more complex product designs
- The Modia project provides **freedom for innovation**
- Several **new algorithms** have been designed and tested
  - could be integrated in Modelica tools
- **New user experiences** are evaluated

Mogram

DLR