Modelica extensions for Highly Dynamic Systems via Julia and OpenModelica?

John Tinnerholm[1] , Adrian Pop[1], Martin Sjölund[1]
[1] Department of Computer and Information Science, Linköping University, Sweden

LARGEDYN

Agenda

- Introducing a Modelica Compiler in Julia
- Initial preliminary benchmarks
  - Frontend
  - Backend
- Some suggested preliminary extensions for highly dynamic systems in Modelica

Introduction and motviation

- Increasingly complex Cyber-Physical Systems
  - Increased requirements on tools
  - Modelica is limited when dealing with highly dynamic systems
- Attempt at a Compiler with explicit backward compatability as the goal:
  - Research languages
  - Embedding: Constrained by the host language regarding expressivity and semantics
- This presentation presents our effort in providing a standard-compliant Modelica Compiler in Julia

Research Aims

- Investigating support for highly dynamic systems using a standard-compliant compiler via source-to-source compilation to Julia[1]:
  - Is Julia suitable to achieve this goal?
  - How to map MetaModelica to Julia?
  - Translation issues?
  - Possible language extensions?

[1]Support for such systems in Julia have been demonstrated during the Modia effort

Comparing MetaModelica to Julia

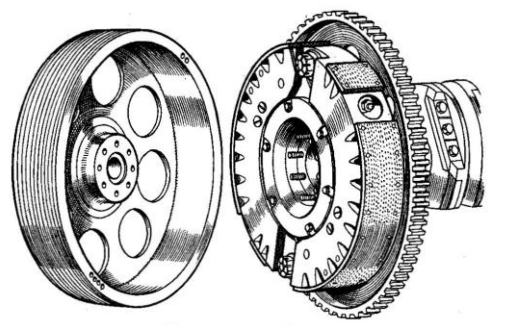
- Similar goals between MetaModelica and Julia
- Similar domain to Modelica
  - Dedicated to numerical computing
  - Capable of handling differential equations via DifferentialEquations.jl
- Similarities to MetaModelica
  - Symbolic-numeric capabilities

MetaModelica	Julia
Syntax influenced by Modelica, Matlab Standard ML, C++	Syntax influenced by Python and Matlab
Verbose syntax, more keywords[1]	Concise syntax[1]
Statically typed	Dynamically typed
Structurally typed with some nominal typing parts	Completely nominal type system
Overloading of functions and operators at compile time	Multiple dispatch at compile time or at runtime
Uniontypes (datatypes) as union of records	Uniontypes as union of any types
Option types with some or none	Option types as union of type vs. nothing

[1]Or Explicit/Implicit

Variable Structure Systems (VSS)

- The meaning of the term VSS varies:
  - Highly Dynamic Systems
  - Multi-Mode DAE Systems
  - VSS
  - ...
- Modelica[1] supports a limited subset
- Embedded languages: Hydra (**Haskell**)[2], Modia (**Julia)**[1]
- Languages and environments: SOL, Mosilab...
- Netiher tool was designed with backward comptability as an explicit goal.



[1]Treated by Elmqvist et al. in Modelica extensions for Multi-Mode DAE Systems 2014

[2]Higher-Order Non-Causal Modelling and Simulation of Structurally Dynamic Systems, Giorgidze & Nilsson

Automatic translation of MetaModelica to Julia

- Motivation
  - Efficient recompilation during runtime. Julia provides just that
  - Automatic translation of the OpenModelica-frontend to aim for backwards compatibility
- Compiler components constructed as a collection of libraries

MetaModelica

julia

```
uniontype Equation
  record EQ_IF
    Exp ifExp "Conditional expression" ;
    list<EquationItem> equationTrueItems "
       true branch" ;
    list<tuple<Exp, list<EquationItem>>>
       elseIfBranches "elseIfBranches" ;
    list<EquationItem> equationElseItems "
       equationElseItems Standard 2-side
       eqn" ;
  end EQ_IF;

  record EQ_EQUALS
    Exp leftSide "leftSide" ;
    Exp rightSide "rightSide Connect stmt"
       ;
  end EQ_EQUALS;
 ...
end Equation
```

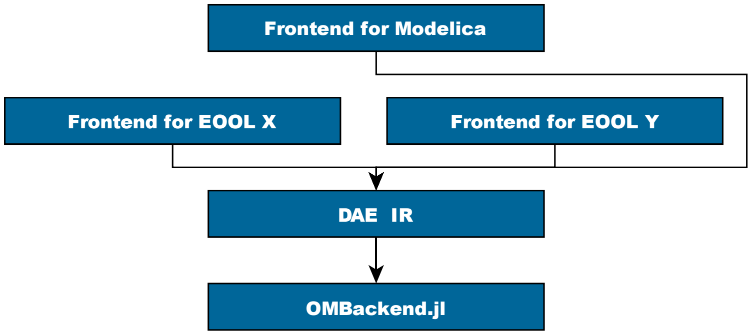
```
@Uniontype Equation begin
 @Record EQ_IF begin
   ifExp::Exp # conditional expression
   equationTrueItems::IList # then
   elseIfBranches::IList # elseif
   equationElseItems::IList # else
 end

 @Record EQ_EQUALS begin
   leftSide::Exp # left hand side
   rightSide::Exp # right hand side
 end
 ...
end
```

Towards OpenModelica.jl

- Most MetaModelica constructs could be mapped into Julia without manual interference.
- Over 100 000 lines of MetaModelica code translated into Julia
  - The complete new frontend
- Algorithms translated into Julia tend to have better performance[1]
- Issues
  - Julia not allowing mutally recursive dependencies
    - No access modifiers
  - Most issues are on the project level
  - Difficult to reimplement MetaModelica backtracking model efficiently

- Benefits
  - Frontends of other equation-oriented languages in Julia could share the hybrid DAE
  - The hybrid DAE of OpenModelica now available in Julia
  - Promising backend performance[2]
  - JIT-Compilation possible

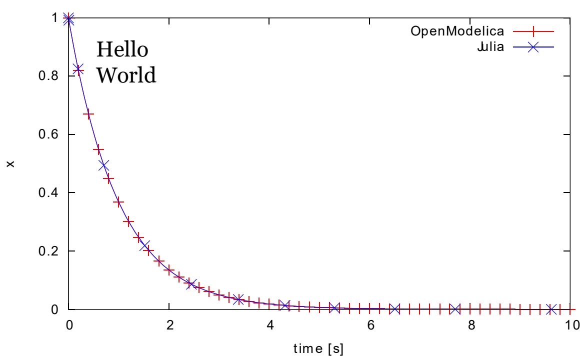


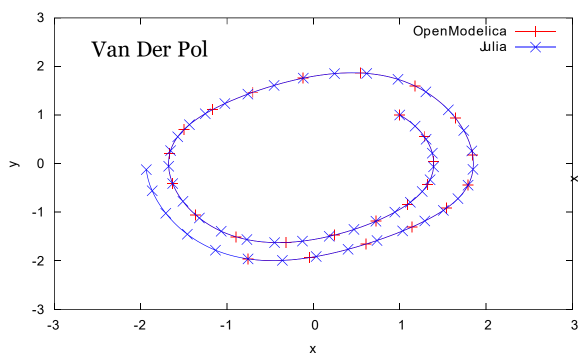
[1]Towards Introducing Just-in-time compilation in a Modelica Compiler

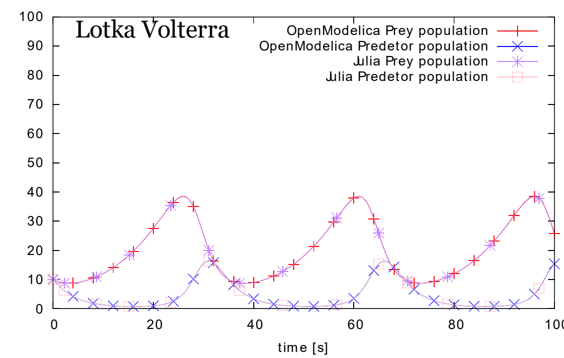
[2]Since DifferentialEquation.jl is used, DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia

Verifing OMFrontend.jl using OMBackend.jl

- Verification
  - Verification of Syntax (ANTLR + OMFrontend.jl)
  - Verification of Semantics (OMFrontend.jl[1] + OMBackend.jl)







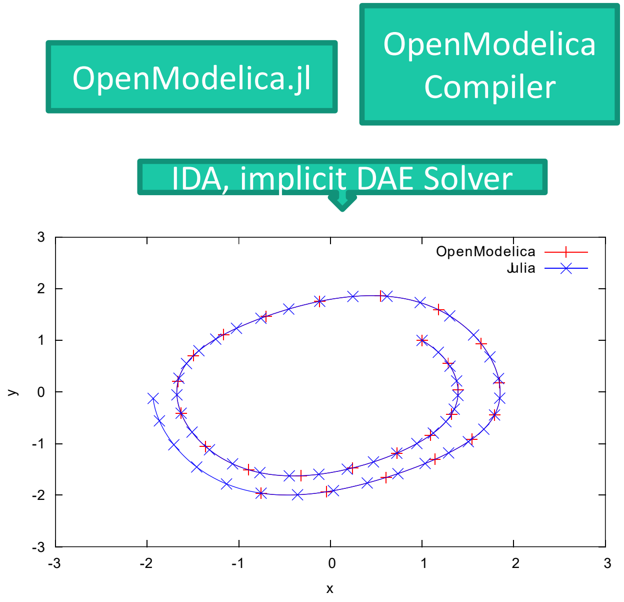
[1]Compile-time metaprogramming

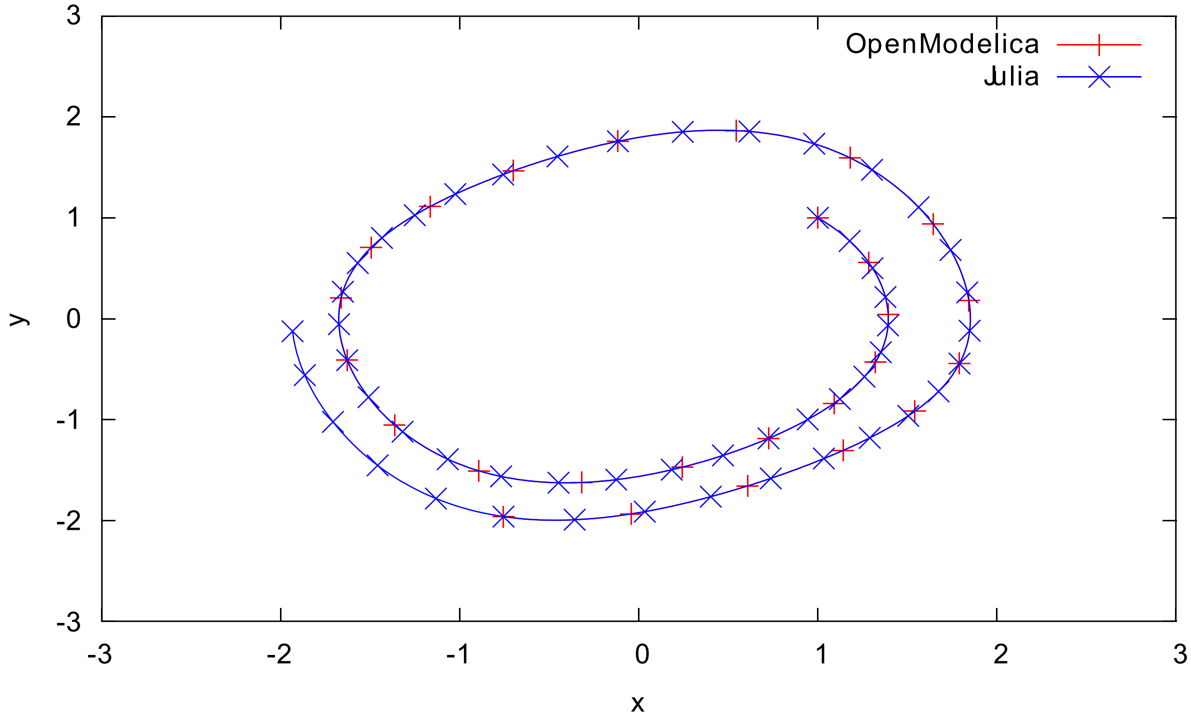
Verifing semantics using OMBackend.jl

- Initial verification of OMCompiler.jl via OpenModelica
- Identical results
- Foundation for more advanced models

```
model VanDerPol
  parameter Real epsilon = 1;
  Real y1(start = 1, fixed = true);
  Real y2(start = 1, fixed = true);
equation
  der(y1) = y2;
  der(y2) = epsilon * (1 - y1 ^ 2) * y2 - y1;
end VanDerPol;
```

$$y_1{}' = y_2$$
$$y_2{}' = \varepsilon \cdot (1 - y_1^2) \cdot y_2 - y_1$$




OpenModelica
Julia
3
2
1
0
-1
-2
-3
y
-3
-2
-1
0
1
2
3
x

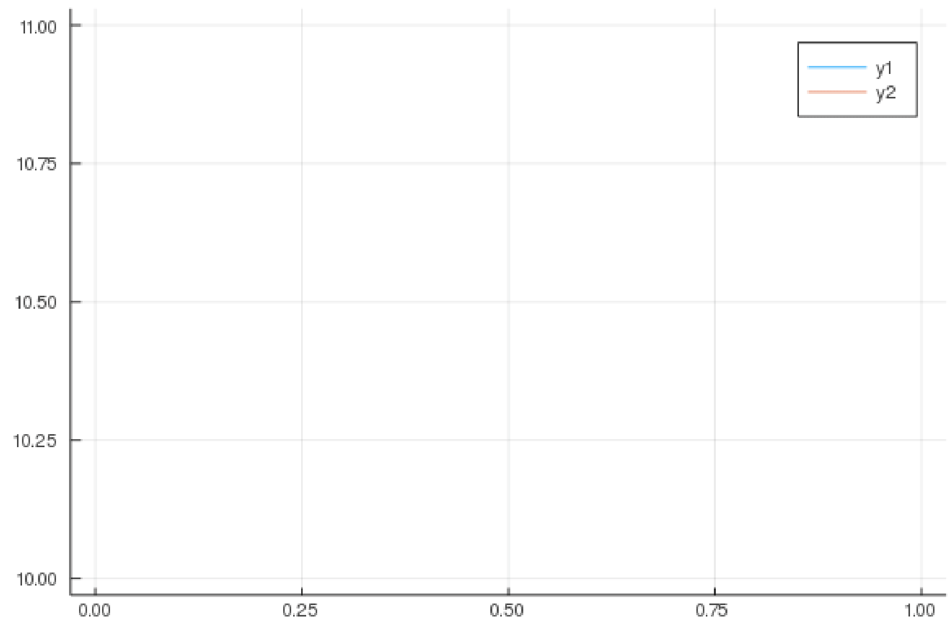
Interactive programming environments via Julia

- As mentioned, several packages make up OMCompiler.jl
- Existing numerical libraries can be repurposed with minimal integration
  - Libraries for symbolic differentiation, the Reduce computer algebra system
- Better integration with OMJulia
- Using packages such as LightGraphs.jl for graph-datastructures and Plots.jl to allow interactive plotting and animation
- Dynamic Optimization
  - A unified language permits Optimization of the model and the optimizer at the same time (Within the same framework)

```
sim = OMBackend.LotkaVolterraSimulate((0.0, 100.0))
arr1 = []
arr2 = []
anim = @animate for t in 1:length(sim.t)
  push!(arr1, sim[t][1])
  push!(arr2, sim[t][2])
  plot(sim.t[1:t], [arr1, arr2])
end
```

Julia for interactive programming environments

```
sim = OMBackend.LotkaVolterraSimulate((0.0, 100.0))
arr1 = []
arr2 = []
anim = @animate for t in 1:length(sim.t)
  push!(arr1, sim[t][1])
  push!(arr2, sim[t][2])
  plot(sim.t[1:t], [arr1, arr2])
end
```



Preliminary performance evaluation

Experimental setup

- The scaleable testsuite[1]
  - N cascaded first order system
  - 100, 200, 400 800 1600 3200 6400
- Hardware specficiation
  - Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz
  - Architecture: x86
  - CPU(s): 12
  - OS: Microsoft Windows
- OpenModelica version
  - OpenModelica Compiler OpenModelica 1.17.0~dev.alpha0

[1] Casella, Francesco (2015). ”Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives”

Frontend performance

Note on Frontend design

- As discussed the frontend is identical to the frontend currently present in the OpenModelica Compiler.
  - All steps in the Julia equvivalent are handled in the same way as in OMC.
- The output is the corresponding Hybrid-DAE

Frontend performance

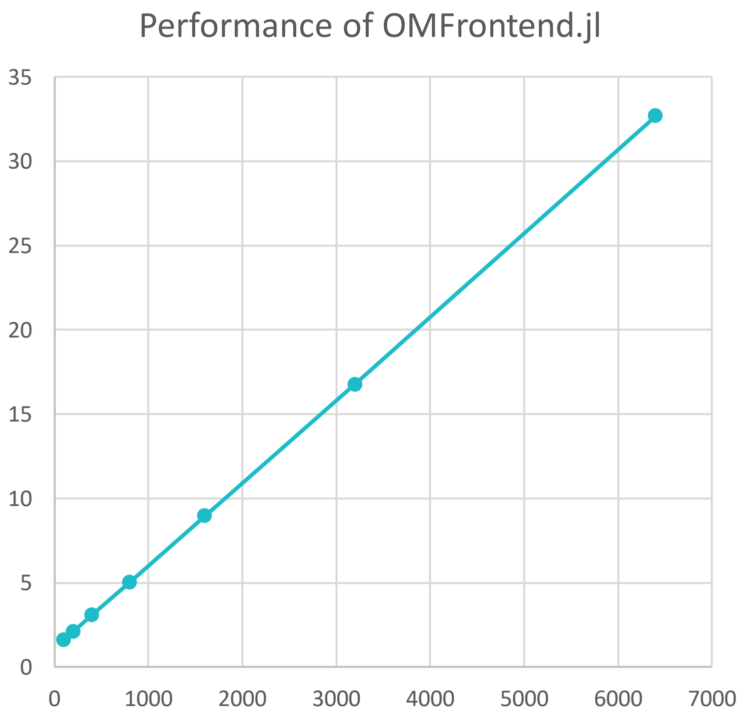
- Issues
  - Recursive structure of MetaModelica programs
  - Previous issues due to the Julia type inference algorithm have been adjusted, however not all together.
- Performance comparision using the scaleable testsuite[1]

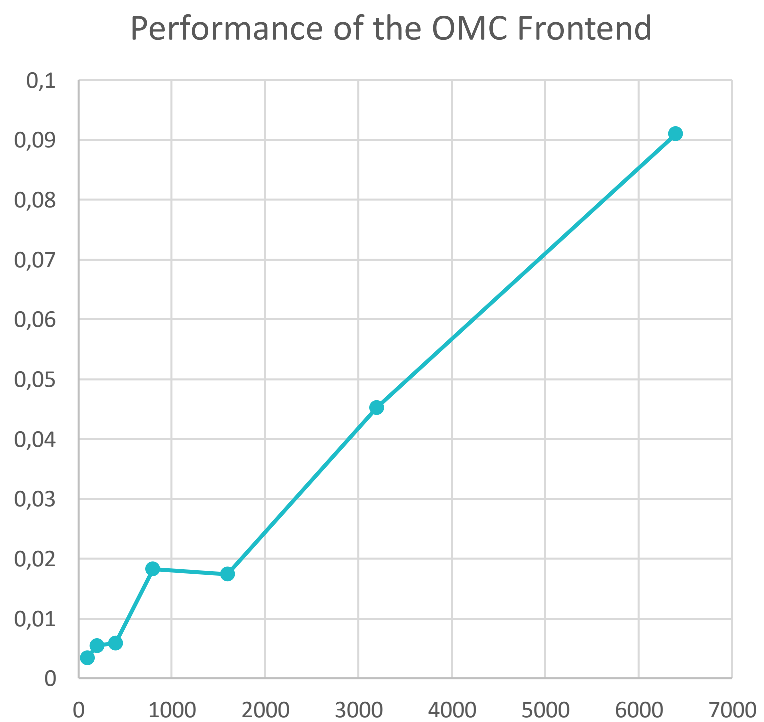
[1] Casella, Francesco (2015). ”Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives”

Frontend performance

Order N	OMFrontend.j	OMC-Frontend
100	1.603 s	0.003492 s
200	2.080 s	0.005483 s
400	3.096 s	0.005883 s
800	5.007 s	0.01826 s
1600	8.956 s	0.01741 s
3200	16.758 s	0.04532 s
6400	32.683 s	0.0910142

A closer look at Frontend performance

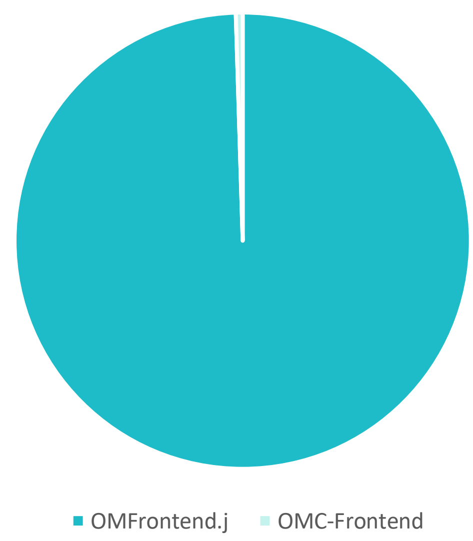




A closer look at Frontend performance

- Reason for the performance difference
  - Issues with the Julia type inference algorithm
  - OMFrontend.jl does interpretation
- How does the translation to SCode scale?
- Performance still needs to be adjusted

Order N	OMFrontend.j	OMC-Frontend
6400	2.149	0.01124



Backend performance

Short note on backend design

- DAE-Mode[1]
- The Reduce Computer algebra system for symbolic manipulation
- Graph data structure implemented using LightGraphs.jl[2]
- Casualisation: Matching, Sorting..
  - Separation of the dynamic and static parts
- OMBackend.jl does currently not generate separate functions for each equation
  - Julia was unable to compile large equation systems because of the size of the ODE.
  - Similar scheme to OMC will be used.
- Numerical integration
  - IDA

[1]Solving large-scale Modelica models: new approaches and experimental results using OpenModelica

[2]LightGraphs.jl: An optimized lightweight graphs package for Julia. Implementations in Julia of standard Graphs algorithms and analytics

Backend performance

- C-Code need not to be generated
  - Faster resimulation
  - Fast feedback loops
    - Important from a development perspective
  - Fast recompilation
- Fast resimulation
- Julia libraries can be integrated seamlessness
  - Further options for post processing
- Still more work is needed

Short demo

Proposed extensions for highly dynamical systems

Initial approach (work in progress)

- Initial scheme
  - Ongoing work
- **state-if**
  - Controls the mode of the system
- A model consists of
  - A set of common variables
  - A set of modes or states
  - One model active at the time
  - Operator that allows adding/removing components(?)
- When the state is changed, complete recompilation to be requested
  - Outer model to be merged with the active state
- Challenges
  - Handling initial conditions during mode changes
  - Caching

Conclusion

- A Modelica Compiler in Julia is possible
- Performance is still somewhat lacking
  - To be expected with automatic translation
  - Library support will help augment this issue
  - Currently could be used for tasks such as teaching
- Backend still work in progress
- Efficient recompilation

Thanks!

...Questions?

www.liu.se

LINKÖPING UNIVERSITY