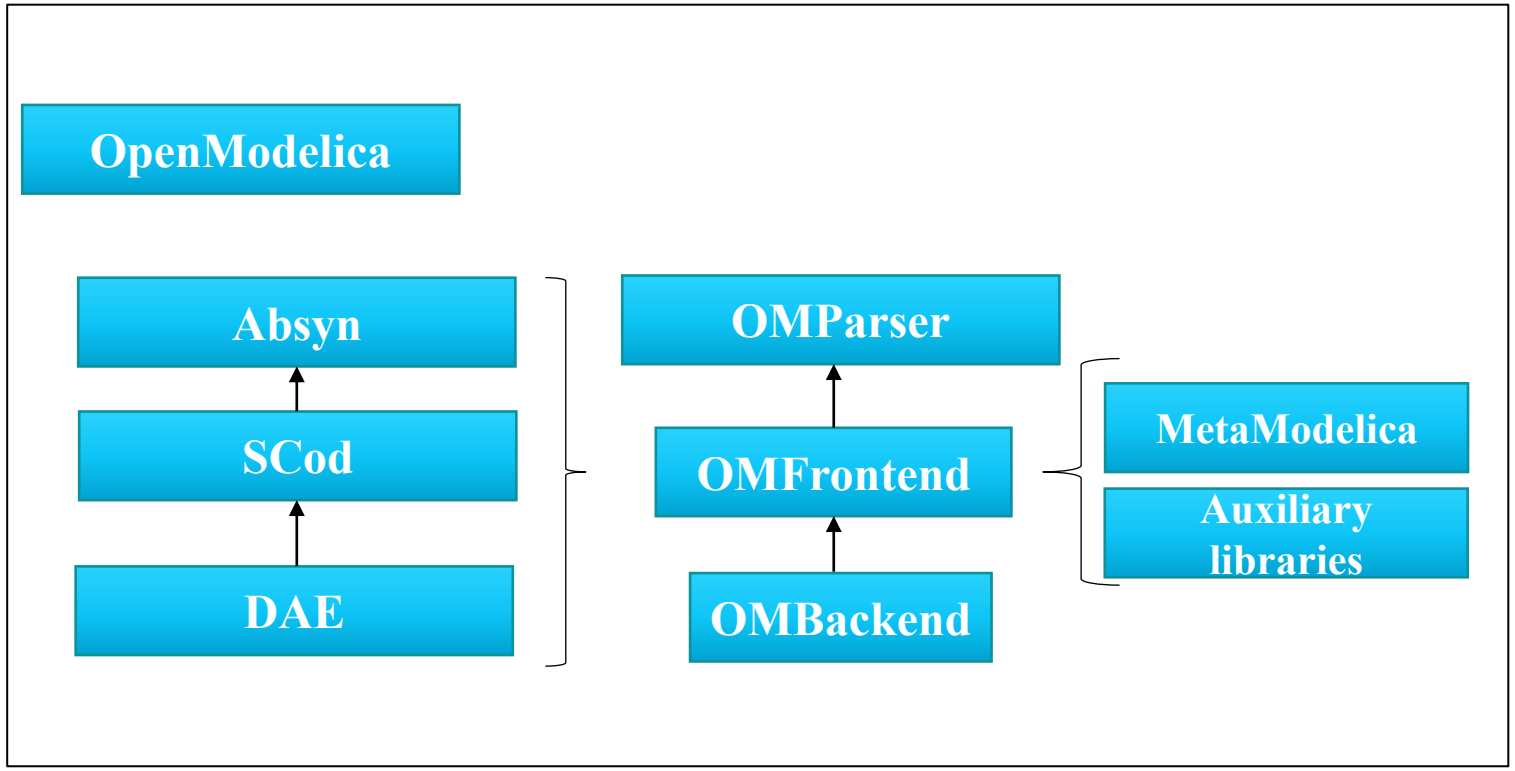
Modelica in the Julia Environment: Latest Developments and Prospects

John Tinnerholm

An OpenModelica Environment in Julia

- Goal
  - An OpenModelica Environment in Julia
- This talk
  - Overview of OpenModelica.jl
  - Some Current Challenges
  - Future Development



Tinnerholm J, Pop A, Sjölund M. A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability. *Electronics*. 2022; 11(11):1772. https://doi.org/10.3390/electronics11111772

Experimental OpenModelica Environment in Julia

- **OpenModelica.jl:** A Modular and Extensible Modelica compiler framework in Julia
- **Translated the high-performance front end.**
- **Able to execute and translate Modelica/MetaModelica functions**
- **Able to simulate discrete-hybrid systems + regular continuous systems**
- **Experimental backends developed**
  - Targeting DifferentialEquations.jl and ModelingToolkit.jl (MTK)
  - Casualization sorting, matching…
  - Integrated LightGraphs.jl package, DAG representation of the hybrid DAE
  - Integration with Sundials. IDAS used for numerical integration
  - Integrated Plots.jl for interactive plotting and animation
- Alpha is released, a *Beta Release in the workings*
  - **MSL support *(New 2022/2023)***
    - ***Working on full coverage in the frontend***
    - ***Full code generation for the backend***
  - **New Low Level Code Generator (New 2023)**
  - **Support for Algorithmic Code Generation (New 2023)**
  - **A System Dynamic importer in progress**
  - **Uppdated for Julia 1.10 (January 2024)**
  - **Optmization of both runtime and compiler structures in the frontend and in the backend**
  - **MsC Thesis using OpenModelica.jl to propose a new language done at TU-Dresden Autumn 2023**
  - **Two Bachelor Thesises in progress at TU Dresden**
- Supporting Experimental Modelica Features:
  - **Language extensions for variable-structure system support (2022)**
  - **Dynamic Overconstrained Connectors (2022)**
  - **THETA (New 2023)**

Visualization of OpenModelica.jl by Chat GPT

OpenModelica.jl aims to be a complete Modelica Environment

```
model FreeFall
  parameter Real e=0.7;
  parameter Real g=9.81;
  Real x;
  Real y;
  Real vx;
  Real vy;
equation
  der(x) = vx;
  der(y) = vy;
  der(vy) = -g;
  der(vx) = 0.0;
end FreeFall;

model Pendulum
  parameter Real x0 = 10;
  parameter Real y0 = 10;
  parameter Real g = 9.81;
  parameter Real L = sqrt(x0^2 + y0^2);
  /* Common variables */
  Real x(start = x0);
  Real y(start = y0);
  Real vx;
  Real vy;
 /* Model specific variables */
  Real phi(start = 1., fixed = true);
  Real phid;
equation
  x = L * sin(phi);
  y = -L * cos(phi);
  der(x) = vx;
  der(y) = vy;
  der(phi) = phid;
  der(phid) = -g / L * sin(phi);
end Pendulum;

model BreakingPendulumStatic
structuralmode Pendulum pendulum;
structuralmode FreeFall freeFall;
equation
  initialStructuralState(pendulum);
  structuralTransition(/* From */ pendulum, /* To */freeFall, time <= 5 /*Condition*/);
end BreakingPendulumStatic;

model BreakingPendulumStaticBouncingBall
structuralmode Pendulum pendulum;
structuralmode BouncingBall bouncingBall;
equation
  initialStructuralState(pendulum);
  structuralTransition(/* From */ pendulum, /* To */bouncingBall, time <= 5 /*Condition*/);
end BreakingPendulumStaticBouncingBall;
```

```
struct FLAT_MODEL <: FlatModel
  name::String
  variables::Vector{Variable}
  equations::Vector{Equation}
  initialEquations::Vector{Equation}
  algorithms::Vector{Algorithm}
  initialAlgorithms::Vector{Algorithm}
  #= VSS Modelica extension =#
  structuralSubmodels::List{FlatModel}
  scodeProgram::Option{SCode.CLASS}
  #= Dynamically Overconstrained connectors =#
  DOCC_equations::List{Equation}
  #= Contains the set of unresolved connect equations =#
  unresolvedConnectEquations::List{Equation}
  active_DOCC_Equations::Vector{Bool}
  #= End VSS Modelica extension =#
  comment::Option{SCode.Comment}
end
```

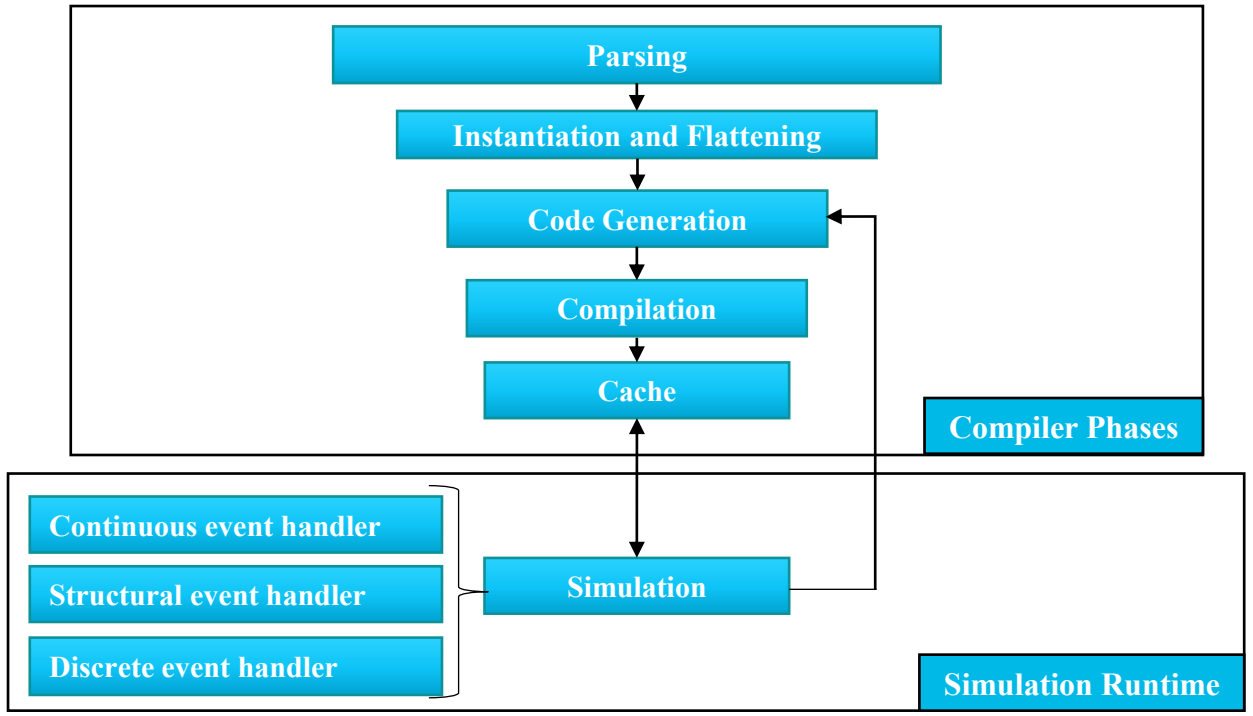
Static Reconfiguration via separate flattening

- Frontend was modified s.t it can flatten models in separation
- Note requires the structural mode keyword
- Possible to formulate model with varying index and compile AOT.
- To the left we can see the flat model definition in the Julia Modelica compiler and an example of a breaking pendulum model.

Language extensions for variable-structure system support (2022)

```
model SimpleClockArrayGrow
   parameter Integer N = 10;
   Real x[N](start = {i for i in 1:N});
equation
/* Resize this problem once every 20 seconds  */
  when sample(0.0, 20.0) then
    recompilation(N/* Parameter  */, /* What we change the parameter */ N + 10);
  end when;
  for i in 1:N loop
    x[i] = der(x[i]);
  end for;
end SimpleClockArrayGrow;
```

```
model SimpleClockParameter
  Real x;
  parameter Integer N = 2;
equation
  when sample(0.0, 0.2) then
    recompilation(N, N * 2);
  end when;
  der(x) = time * N;
end SimpleClockParameter;
```



- Possible to formulate recursive models that expand during simulation

Tinnerholm J, Pop A, Sjölund M. A Modular, Extensible, and Modelica-Standard-Compliant OpenModelica Compiler Framework in Julia Supporting Structural Variability. *Electronics*. 2022; 11(11):1772. https://doi.org/10.3390/electronics11111772

Dynamic Overconstrained Connectors

- Currently, Overconstrained Connectors in Modelica can not be used in If-Equations[3]
  - Relaxing constraints
- Allowing a special If-Equation construct where the **Connectors.branch** operator is allowed
  - Allowing changing the connection graph dynamically at runtime.
- More efficient simulations
- Allows the simulations of models current tools are unable to simulate

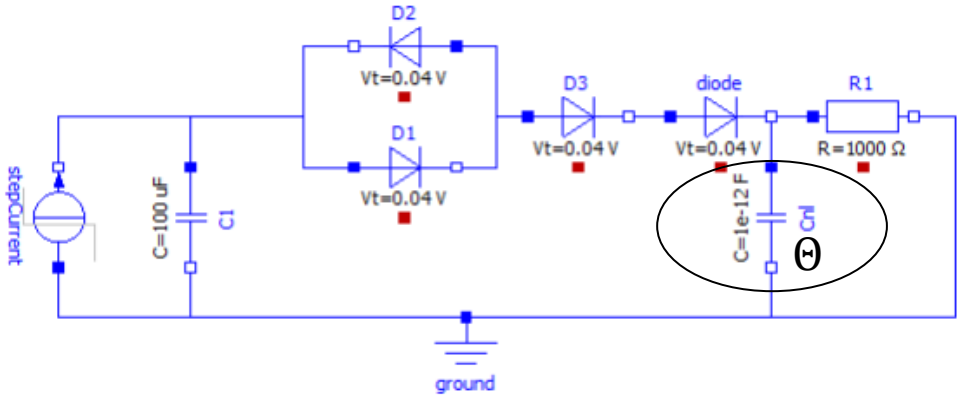
```
model TransmissionLineVariableBranch
  extends TransmissionLineBase;
equation
  if closed then
    port_a.omegaRef = port_b.omegaRef;
    Connections.branch(port_a.omegaRef,
                       port_b.omegaRef);
  end if;
end TransmissionLineVariableBranch;
```

➢ Restricted case of VSS, efficient no recompilation needed. Could be implemented in a traditional compiler using value propagation and pointer swapping

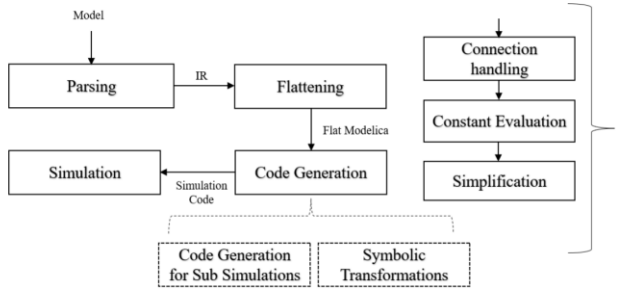
Tinnerholm, J., Casella, F., & Pop, A. (2022, November). Towards Modeling and Simulation of Dynamic Overconstrained Connectors in Modelica. In *Modelica Conferences* (pp. 35-44).

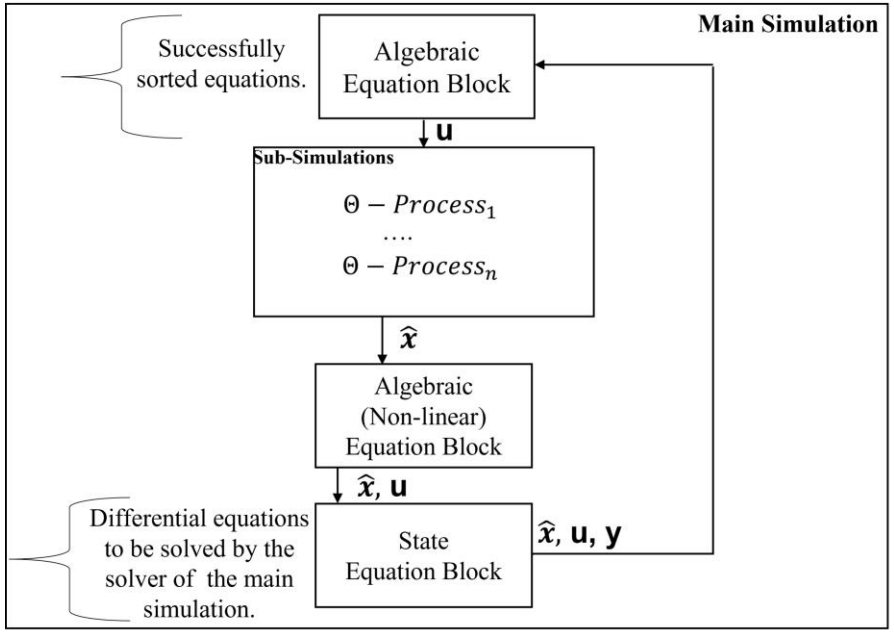
[3]https://specification.modelica.org/maint/3.5/connectors-and-connections.html#restrictions-of-connections-and-connectors

THETA-Operator




```
package CircuitTest
  model ThetaCircuit2Dynamic
    parameter Real THETA = 1.0;
    extends Circuit1Static;
    Capacitor Cp(C = 1e-12 * THETA);
  equation
    connect(Cp.n, ground.p);
    connect(diode.n, Cp.p);
  end ThetaCircuit2Dynamic;
end CircuitTest;
```





Tinnerholm, J., Casella, F., & Pop, A. (2023, December) Supporting Infinitely Fast Processes in Continuous System Modeling in the Proceedings of the 15th International Modelica Conference

System Dynamics and Algorithmic Modelica

The Resulting Translation

The Translation was done according to the XMILE specification

SD Elements were mapped to the corresponding Modelica Elements

Table 1. Subset of Modelica to SD type matchings

SD Type	Modelica Formulation
stock	$der(stock) = inflows - outflows$
smooth	$der(smooth) = \frac{input - smooth}{averagingTimeVariable}$
flow	$flow = inflow$

```
model ESCIMO
  constant Real Future_volcanic_emissions(unit =
  ↪  "GtVAe/yr") = 0.0 "CONST";
  constant Real Albedo_Antarctic_sens(unit = "fraction") =
  ↪  0.7 "CONST";
  constant Real
  ↪  Annual_pct_increase_CH4_emissions_from_2015_pct_yr(unit
  ↪  = "1/yr") = 0.0 "CONST";
...
initial equation
  Antarctic_ice_volume_km3 =
  ↪  Antarctic_ice_volume_in_1850_km3   "STOCK";
  Arctic_ice__on_sea__area_km2 =
  ↪  Arctic_ice_area_in_1850_km2   "STOCK";
  C_in_permafrost_in_form_of_CH4 = 1200.0   "STOCK";
...
equation
...
  der(DESERT_Mkm2) =
  ↪  flow_Shifting_GRASS_to_DESERT_Mkm2_yr -
  ↪  flow_Sifting_DESERT_to_GRASS_Mkm2_yr   "STOCK";
  der(Fossil_fuel_reserves_in_ground_GtC) =   -
  ↪  flow_Man_made_fossil_C_emissions_GtC_yr   "STOCK";
  der(GRASS_area_burnt_Mkm2) =  flow_GRASS_burning_Mkm2_yr
  ↪  - flow_GRASS_regrowing_after_being_burnt_Mkm2_yr
  ↪  "STOCK";
...
  UNIT_conversion_for_CH4_from_CO2e_to_C =  1/(16/12 *
  ↪  Global_Warming_Potential_CH4)   "AUX";
  UNIT_conversion_for_CO2_from_CO2e_to_C =  12/44   "AUX";
  UNIT_conversion_from_MtCH4_to_GtC =  1 /( 1000 / 12 *
  ↪  16)   "AUX";
...
  flow_SW_surface_absorption=SW_surface_absorption
  ↪  "FLOW";
  flow_GRASS_runoff=GRASS_runoff  "FLOW";
  flow_NATURE_CCS_Fig3_GtC_yr=NATURE_CCS_Fig3_GtC_yr
  ↪  "FLOW";
...
end ESCIMO
```

Initial support for Algorithmic Modelica was added in order to simulate this model

- **Ongoing work on expanding support for algorithmic Modelica**

Tinnerholm, J., Zapatero, M., Pop, A., Fritzson, P., & Castro, R. (2023). Automatic Translator from System Dynamics to Modelica with Application to Socio-Bio-Physical Systems. *Scandinavian Simulation Society*, 302-309.

Translating System Dynamics into Modelica

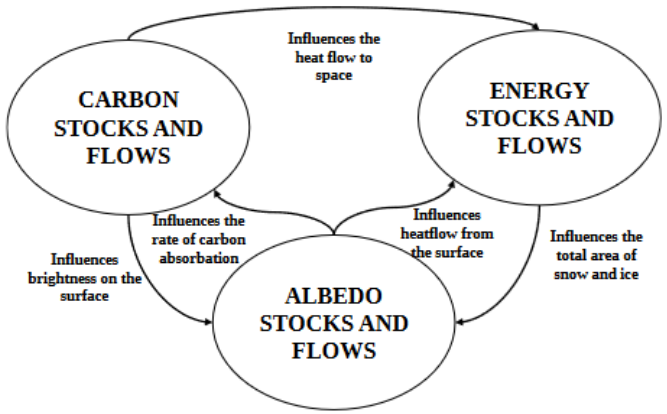


Figure 1. The three sectors of the ESCIMO climate model as described by (Randers et al., 2016).

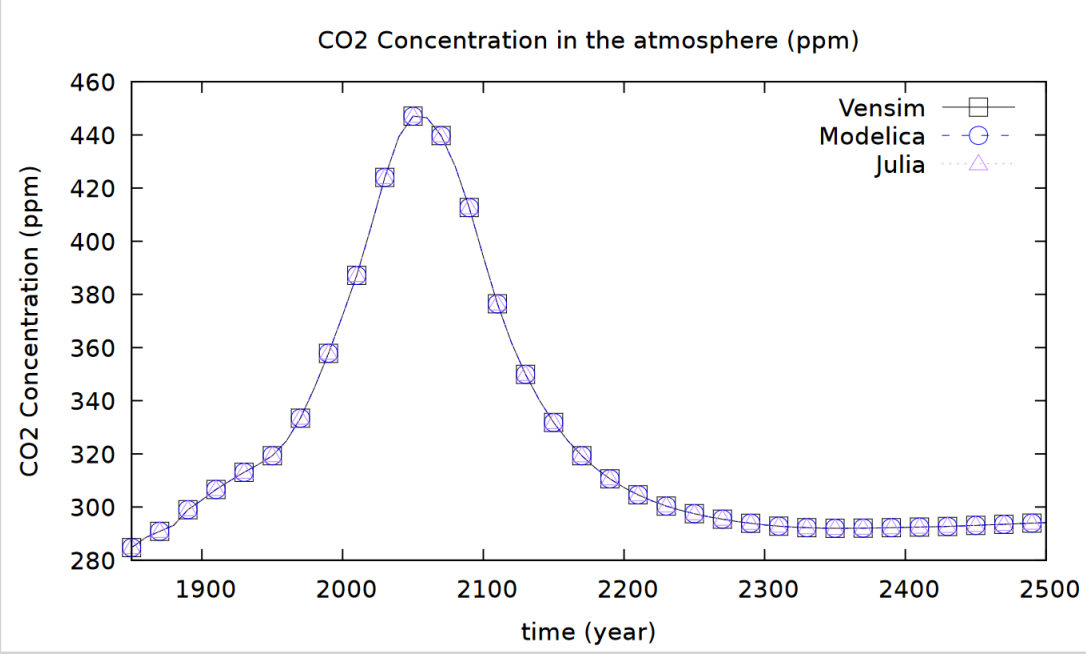
```
<model>
  <sim_specs> <!-- OPTIONAL-->
  ...
  </sim_specs>
  <behavior> <!-- OPTIONAL-->
    ...
  </behavior>
  <variables> <!-- REQUIRED -->
    ...
  </variables>
  <views> <!-- OPTIONAL-->
   ...
  </views>
</model>
```

- In order to achieve this a *SD to Modelica translator* was developed
- Mapping XMILE to Modelica
  - Oasis XML Interchange
- We used ESCIMO, which is a fairly complicated SD model, to validate the translator

Randers, J., Golüke, U., Wenstøp, F., & Wenstøp, S. (2016). A user-friendly earth system model of low complexity: The ESCIMO system dynamics model of global warming towards 2100. *Earth System Dynamics*, *7*(4), 831-850.

Tinnerholm, J., Zapatero, M., Pop, A., Fritzson, P., & Castro, R. (2023). Automatic Translator from System Dynamics to Modelica with Application to Socio-Bio-Physical Systems. *Scandinavian Simulation Society*, 302-309.

Validation of the Translator



- The translator was validated by first simulating the results in Vensim and then simulating the same model in Modelica and in Julia
- The simulation was run from 1850 to 2500
- We then compared the results for each tenth-year to examine deviations from the original model
- Three variables were examined in detail
- Temperature surface anomaly compared to 1850 (Celsius), that is, the difference in average global surface temperature compared to 1850.
- pH in warm surface water, that is, the acidity of warm surface water
- CO2 Concentration in PPM, that is, the concentration of carbon dioxide in the atmosphere.
- Solver Settings
- Runge-Kutta-4 (Vensim)
- DASSL with absolute and relative tolerance of 1E-6 for Modelica
- Rodas5 with absolute and relative tolerance of 1E-6 for Julia

Tinnerholm, J., Zapatero, M., Pop, A., Fritzson, P., & Castro, R. (2023). Automatic Translator from System Dynamics to Modelica with Application to Socio-Bio-Physical Systems. *Scandinavian Simulation Society*, 302-309.

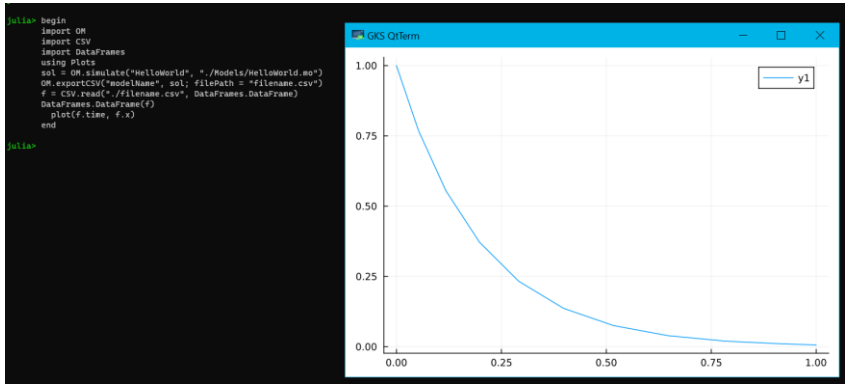
Practical Examples

Simulating models and profile memory

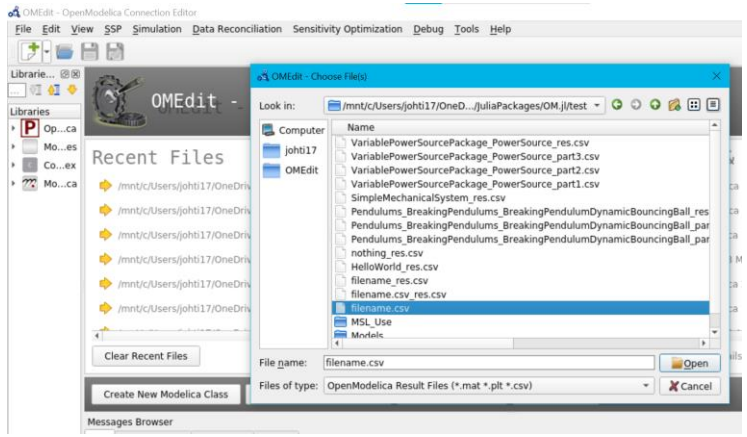
```
function profileMemory(model = "Modelica.Mechanics.MultiBody.Examples.Loops.Engine1a")
  # Collect a profile
  Profile.clear()
  #= Precompile j i c=#
  println("Test to flatten a model")
  @time flattenModelInMSL_TST();
  println("Running  a second time")
  @time flattenModelInMSL_TST(model);
  #=
  Try to flatten an engine model in the multibody library.
  =#
  println("Profile memory allocations of that model")
  Profile.Allocs.@profile bsample_rate=0.1  flattenModelInMSL_TST(model);
  println("Running profiler")
  PProf.Allocs.pprof()
end
```

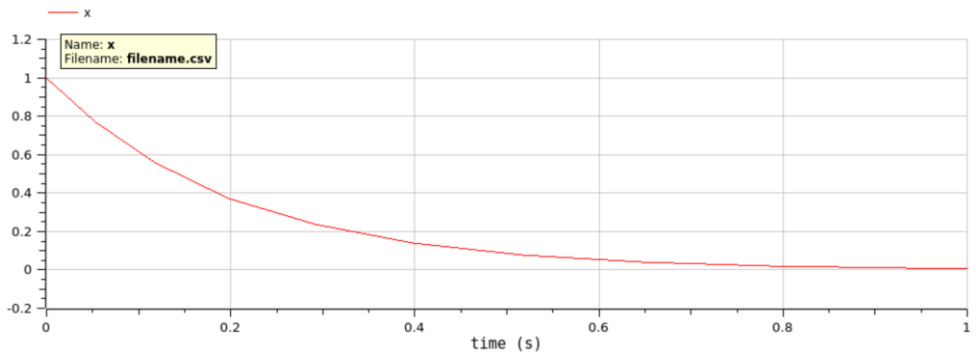
Simulating and Plotting

```
julia> begin
import OM
import CSV
import DataFrames
using Plots
sol = OM.simulate("HelloWorld", "./Models/HelloWorld.mo")
OM.exportCSV("modelName", sol; filePath = "filename.csv")
f = CSV.read("./filename.csv", DataFrame)
DataFrames.DataFrame(f)
  plot(f.time, f.x)
end
```



Practical Examples Continued





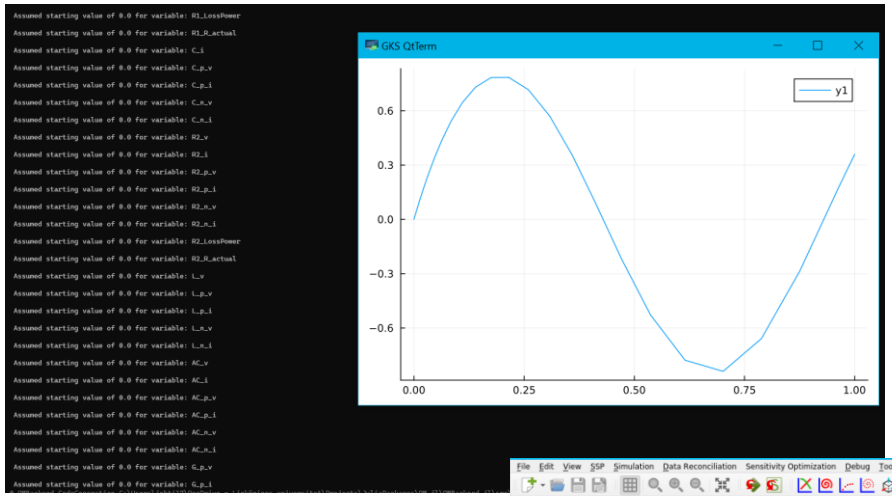
Possible to simulate a model and export the result to OMEdit

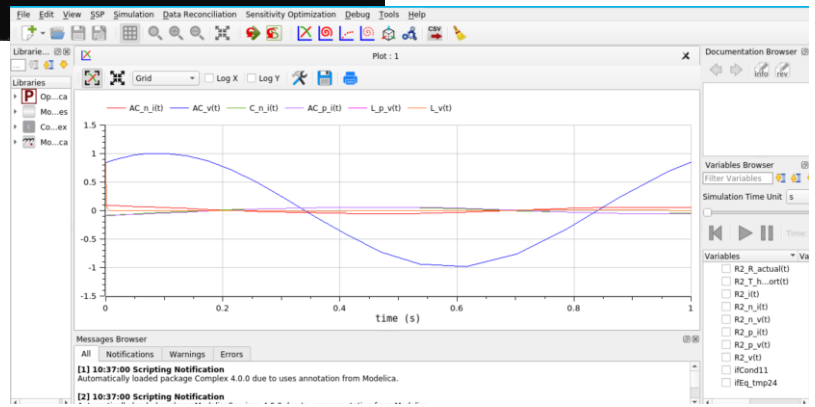
```
OM.exportCSV("modelName", sol; filePath = "filename.csv")
```

Practical Examples using the Standard Library

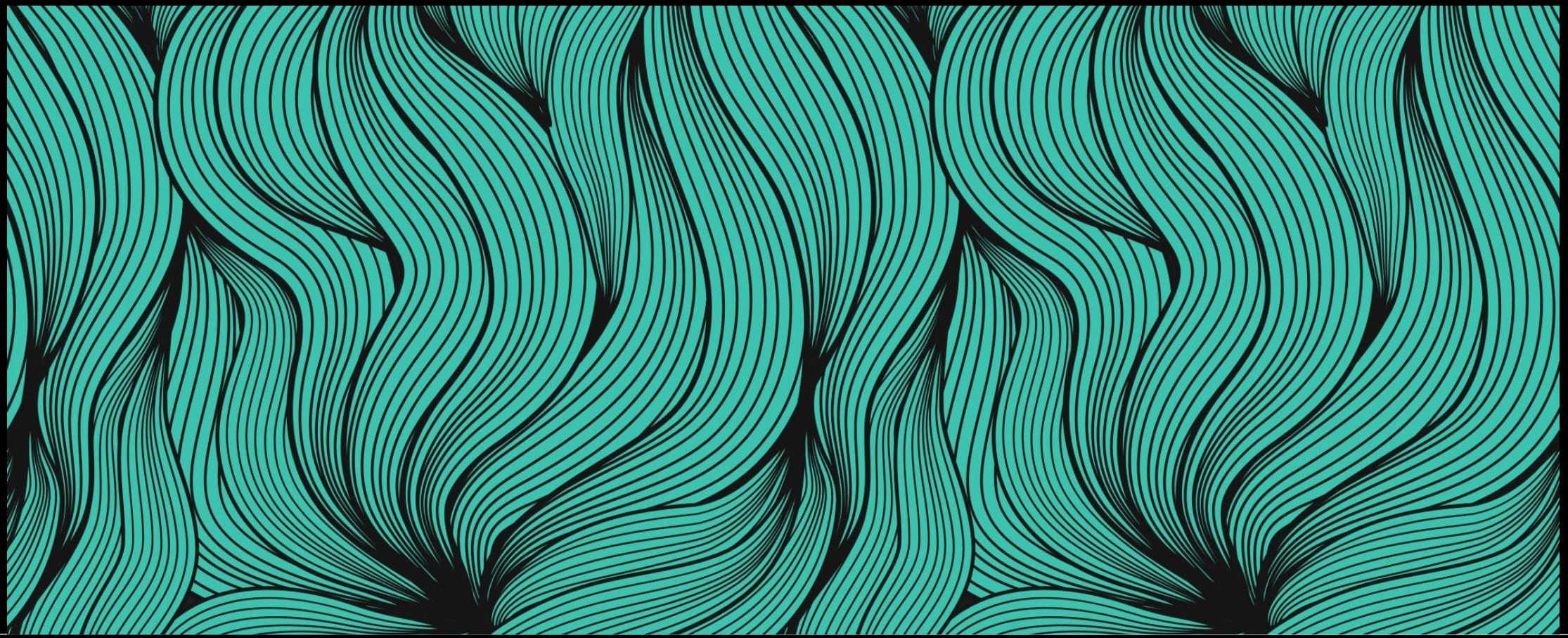
```
model ElectricalComponentTestMSL
//MSL Imports
import Modelica.Electrical.Analog.Basic.Ground;
import Modelica.Electrical.Analog.Basic.Resistor;
import Modelica.Electrical.Analog.Basic.Capacitor;
import Modelica.Electrical.Analog.Basic.Inductor;
import Modelica.Electrical.Analog.Sources.SineVoltage;

//Equation for the serial RLC circuit.
  model SimpleCircuit
    Resistor R1(R=10);
    Capacitor C(C=0.01);
    Resistor R2(R=100);
    Inductor L(L=0.1);
    SineVoltage AC(freqHz = 1., phase = 1.);
    Ground G;
  equation
    connect(AC.p, R1.p); // 1, Capacitor circuit
    connect(R1.n, C.p); // Wire 2
    connect(C.n, AC.n); // Wire 3
    connect(R1.p, R2.p); // 2, Inductor circuit
    connect(R2.n, L.p); // Wire 5
    connect(L.n, C.n); // Wire 6
    connect(AC.n, G.p); // 7, Ground
  end SimpleCircuit;

end ElectricalComponentTestMSL;
```

```
julia> begin
import OM
import CSV
import DataFrames
using Plots
  sol = OM.simulate("ElectricalComponentTestMSL.SimpleCircuit",
                    "./MSL_USE/ElectricalComponentTest.mo";
                    MSL = true, MSL_VERSION = "MSL:3.2.3")
OM.exportCSV("modelName", sol; filePath = "filename.csv")
f = CSV.read("./filename.csv", DataFrames.DataFrame)
plot(f.time, f.C_v)
end
```





Notes on Performance



Performance in OpenModelica.jl

- ***Some History***
- Translation performance has historically been low in the frontend
- First version (Early 2020), correct but around ~ 1 hour to compile simple programs like HelloWorld…
- Issues
  - Julia Type Inference: Julia historically struggled with type inference for mutually recursive data structures with deep recursion
    - Solved by introducing manually introducing barriers to type inference
    - Hindrance of fully automatic translation
  - OpenModelica relies on exceptions for control flow for operations such as lookup and error handling
    - Exceptions, while expensive, are comparable inexpensive in MetaModelica compared to Julia
- Julia 1.5 - 1.6 ~ Gamechanger
  - Could compile model without inference issues
- Julia 1.10
  - Faster Garbage Collector
- More or less OpenModelica.jl has become faster and more memory efficient for each Julia Version since
  - Furthermore, various improvements to the runtime and runtime data structures has been made
  - Examples include statically generating less code when detecting a match expression that can not fail
- Current version
  - *Faster than omc translating **some** models…*

Better than OpenModelica?

```
julia>     include("perf.jl")
  0.001865 seconds (5.23 k allocations: 263.023 KiB)
Flat Model:
class HelloWorld
  Real x(fixed = true, start = 1.0);
  parameter Real a = 1.0;
equation
  der(x) = -a * x;
end HelloWorld;

(No Functions)
```

- For small models, the Julia variant of the frontend perform better
- Conclusion is OM.jl better than OpenModelica?
  - Not really…

```
"
record SimulationResult
    resultFile = "/mnt/c/Users/johti17/OneDrive - Linköpings univ
    simulationOptions = "startTime = 0.0, stopTime = 1.0, numberO
    messages = "LOG_SUCCESS       | info    | The initialization
LOG_SUCCESS       | info    | The simulation finished successfull
",
    timeFrontend = 0.002198113,
    timeBackend = 0.012460822,
    timeSimCode = 0.0009588540000000001,
    timeTemplates = 0.042675787,
    timeCompile = 1.314570023,
    timeSimulation = 0.036142027,
    timeTotal = 1.411066526
end SimulationResult;
```

Memory Patterns for HelloWorld

```
model HelloWorld
  Real x( start = 1, fixed = true );
  parameter Real a = 1;
equation
  der(x) = - a * x;
end HelloWorld;
```

Experience: For small models such as the HelloWorld model and others, OpenModelica.jl > OpenModelica

```
julia>     include("perf.jl")
  0.001865 seconds (5.23 k allocations: 263.023 KiB)
Flat Model:
class HelloWorld
  Real x(fixed = true, start = 1.0);
  parameter Real a = 1.0;
equation
  der(x) = -a * x;
end HelloWorld;

(No Functions)
```

- Here, OM.jl uses less memory even during instantiation…
- As we will see next it also uses less memory for other operations…
  - However, the Julia Modelica Compiler *is cheating….*
  - *Note, OM.jl requires less memory here than omc requires for just one phase,*

OpenModelica:
Notification: Performance of NFInst.instantiate(HelloWorld): time 0.00136/0.001485, allocations: **366.4 kB** / 14.4 MB, free: 220 kB / **13.93 MB**

- A rough comparison of memory and speed for the **Engine1a** model in the Multibody library

- Here, OpenModelica consumes more memory *in total* but is around 3 times faster.
  - Certain phases are cheaper in OpenModelica
  - *More on that later..*

- For other model's similar patterns can be observed

```
package Test

import Modelica.Electrical.Analog.Basic.Ground;
import Modelica.Mechanics.MultiBody.Examples.Loops.Engine1a;

model MBTest
Engine1a engine;
equation
end MBTest;

end Test;
```

```
Notification: Performance of prepare postOptimizeDAE: time 0.009321/3.412, allocations: 3.976 MB / 0.7221 GB, free: 136.9 MB / 395.4 MB
Notification: Performance of postOpt lateInlineFunction (simulation): time 0.001914/3.414, allocations: 0.6345 MB / 0.7228 GB, free: 136.9 MB / 395.4 MB
Notification: Performance of postOpt wrapFunctionCalls (simulation): time 0.034/3.448, allocations: 14.31 MB / 0.7367 GB, free: 126.7 MB / 395.4 MB
Notification: Performance of postOpt inlineArrayEqn (simulation): time 4.761e-05/3.448, allocations: 39.94 kB / 0.7368 GB, free: 126.6 MB / 395.4 MB
Notification: Performance of postOpt constantLinearSystem (simulation): time 4.364e-05/3.448, allocations: 20 kB / 0.7368 GB, free: 126.6 MB / 395.4 MB
Notification: Performance of postOpt simplifysemiLinear (simulation): time 4.682e-05/3.448, allocations: 14.77 kB / 0.7368 GB, free: 126.6 MB / 395.4 MB
Notification: Performance of postOpt removeSimpleEquations (simulation): time 0.05772/3.506, allocations: 29.53 MB / 0.7657 GB, free: 96.97 MB / 395.4 MB
Notification: Performance of postOpt simplifyComplexFunction (simulation): time 2.96e-05/3.506, allocations: 11.47 kB / 0.7657 GB, free: 96.96 MB / 395.4 MB
Notification: Performance of postOpt solveSimpleEquations (simulation): time 0.002412/3.508, allocations: 418.8 kB / 0.7661 GB, free: 96.58 MB / 395.4 MB
Notification: Performance of postOpt tearingSystem (simulation): time 0.02351/3.532, allocations: 7.19 MB / 0.7731 GB, free: 89.45 MB / 395.4 MB
Notification: Performance of postOpt inputDerivativesUsed (simulation): time 0.0006618/3.533, allocations: 62.69 kB / 0.7731 GB, free: 89.39 MB / 395.4 MB
Notification: Performance of postOpt calculateStrongComponentJacobians (simulation): time 0.07976/3.612, allocations: 38.42 MB / 0.8107 GB, free: 50.89 MB / 395.4 MB
Notification: Performance of postOpt calculateStateSetsJacobians (simulation): time 1.49e-05/3.612, allocations: 4.547 kB / 0.8107 GB, free: 50.89 MB / 395.4 MB
Notification: Performance of postOpt symbolicJacobian (simulation): time 0.05104/3.663, allocations: 20.43 MB / 0.8306 GB, free: 30.58 MB / 395.4 MB
Notification: Performance of postOpt removeConstants (simulation): time 0.002354/3.666, allocations: 0.7332 MB / 0.8313 GB, free: 29.83 MB / 395.4 MB
Notification: Performance of postOpt simplifyTimeIndepFuncCalls (simulation): time 0.002407/3.668, allocations: 60 kB / 0.8314 GB, free: 29.77 MB / 395.4 MB
Notification: Performance of postOpt simplifyAllExpressions (simulation): time 0.007889/3.676, allocations: 303.5 kB / 0.8317 GB, free: 29.47 MB / 395.4 MB
Notification: Performance of postOpt findZeroCrossings (simulation): time 0.001987/3.678, allocations: 244.4 kB / 0.8319 GB, free: 29.24 MB / 395.4 MB
Notification: Performance of postOpt collapseArrayExpressions (simulation): time 0.0008735/3.679, allocations: 95.17 kB / 0.832 GB, free: 29.15 MB / 395.4 MB
Notification: Performance of sorting global known variables: time 0.015/3.694, allocations: 4.887 MB / 0.8368 GB, free: 24.29 MB / 395.4 MB
Notification: Performance of sort global known variables: time 7.41e-07/3.694, allocations: 0 / 0.8368 GB, free: 24.29 MB / 395.4 MB
Notification: Performance of remove unused functions: time 0.01041/3.705, allocations: 2.134 MB / 0.8389 GB, free: 22.16 MB / 395.4 MB
Notification: Model statistics after passing the back-end for simulation:
```

```
julia>     include("perf.jl");
[ Info: Loading MSL Version: MSL:4.0.0
  0.000003 seconds
[ Info: Loaded MSL successfully
Attempting to instantiate...Test.MBTest
  7.353518 seconds (7.66 M allocations: 336.631 MiB, 4.91% gc time)
```

```
FrontEnd: time 2.415e-06/2.294, allocations: 0 / 402.6 MB, free: 14.48 MB / 331.4 MB
```

```
Notification: Performance of NFInst.instantiate(Test.MBTest): time 2.161/2.161, allocations: 312.4 MB / 326.4 MB, free: 10.83 MB / 251.4 MB
```

Reason: Issue with certain MetaModelica practices

```
""" #= Performs an AVL right rotation on the given tree. =#"""
function rotateRight(inNode::Tree)::Tree
  local outNode::Tree = inNode
   outNode = begin
    local node::Tree
    local child::Tree
    @match outNode begin
      NODE(left = child && NODE(__)) => begin
        node = setTreeLeftRight(outNode, left = child.right, right = outNode.right)
        setTreeLeftRight(child, right = node, left = child.left)
      end

      NODE(left = child && LEAF(__)) => begin
        node = setTreeLeftRight(outNode, left = EMPTY(), right = outNode.right)
        setTreeLeftRight(child, right = node, left = EMPTY())
      end
      _ => begin
        inNode
      end
    end
  end
  return outNode
end
```

- Not apparent at first glance, but due to the heavy recursion used for this function, the named arguments here create a significant number of allocations
- Removing the keyword arguments here saved 5% of memory
  - ➢ Small models where lookup processing is a significant part of the translation

✓ Changing this in the omc could possibly save some memory as well

Challenges

High Memory Consumption in the Frontend for some Models

- Currently, the frontend consumes too much memory for operations, around 10X that of OpenModelica
  - Also, the OpenModelica Frontend has improved future in terms of memory efficiency since I started this work
- Memory patterns vary from model to model
- The backend (**OMBackend.jl**) could use some efficiency improvements during translation.
- Not yet 100% coverage of the Standard Library in the Frontend
  - *Some issues with Fluids*

The Silver Lining

- For small models, OM.jl typically consumes less memory than OM
  - No need to reparse libraries
  - Faster instantiations for some models
  - Libraries such as MSL are loaded upon compilation…..
  - ✓ The translation performance of the frontend for the Julia Modelica Compiler is now decent at large
- While we do not fully cover the MSL. A significant amount of the Modelica Standard Library is handled by the frontend
- The MTK and the wider Julia ecosystem provide a wide array of solvers and libraries
  - Better simulation time than OpenModelica[1]

[1]For some models, and also via new language features such as DOCC and the THETA operator

Using OpenModelica + OM.jl

Flat Modelica Capabilities

- Using tools such as **OMJulia** it is possible to directly interface the rest of the OpenModelica environment
- For models currently not handled by the Julia OpenModelica Compiler it is possible to output flat Modelica and then feed to the OpenModelica.jl environment

Consequences

- One can imagine a combination of OMJulia.jl and OpenModelica.jl to use advanced libraries such as the Buildings Library within the Julia Environment
  - ***Backend can be reused with the frontend***

Conclusions

- OM.jl is moving closer to an initial real release
- Due to improvements of, Julia I now believe that one can feasibly implement a full-fledged compiler in it that not only works but also has decent performance
  - Some details…
- OM.jl provides a way of working with the Modelica ecosystem in the Julia Environment
- The capabilities of Julia allow one to quickly implement and prototype new language features

References

Tinnerholm, J., Sjölund, M., & Pop, A. (2019, November). Towards introducing just-in-time compilation in a modelica compiler. In Proceedings of the 9th International Workshop on Equation-based Object-oriented Modeling Languages and Tools (pp. 11-19).

Tinnerholm, J., Pop, A., Sjölund, M., Heuermann, A., & Abdelhak, K. (2020).

Tinnerholm J. et al. Towards an Open-Source Modelica Compiler in Julia.

Tinnerholm, J., Pop, A., Heuermann, A., & Sjölund, M. (2021, September). OpenModelica. jl: A modular and extensible Modelica compiler framework in Julia targeting ModelingToolkit. jl. In Modelica Conferences (pp. 109-117).

Tinnerholm, J. (2022). A Composable and Extensible Environment for Equation-based Modeling and Simulation of Variable Structured Systems in Modelica (Licentiate dissertation, Linköping University Electronic Press).

Tinnerholm, J., Casella, F., & Pop, A. (2022, November). Towards Modeling and Simulation of Dynamic Overconstrained Connectors in Modelica. In *Modelica Conferences* (pp. 35-44).

Tinnerholm, J., Casella, F., & Pop, A. (2023, December) Supporting Infinitely Fast Processes in Continuous System Modeling in the Proceedings of the 15th International Modelica Conference

Tinnerholm, J., Zapatero, M., Pop, A., Fritzson, P., & Castro, R. (2023). Automatic Translator from System Dynamics to Modelica with Application to Socio-Bio-Physical Systems. *Scandinavian Simulation Society*, 302-309.

Questions

Visualization of OpenModelica.jl by Chat GPT

LINKÖPING UNIVERSITY

Bonus Slide I did one for OpenModelica as well

Visualization of **OpenModelica** by Chat GPT

LINKÖPING UNIVERSITY