Discretizing Time or States?
A Comparative Study between DASSL and QSS

Xenofon Floros[1] Francois E. Cellier[1] Ernesto Kofman[2]

[1] Department of Computer Science, ETH Zurich, Switzerland

{xenofon.floros, francois.cellier}@inf.ethz.ch

[2] Laboratorio de Sistemas Dinámicos, FCEIA, Universidad Nacional de Rosario, Argentina

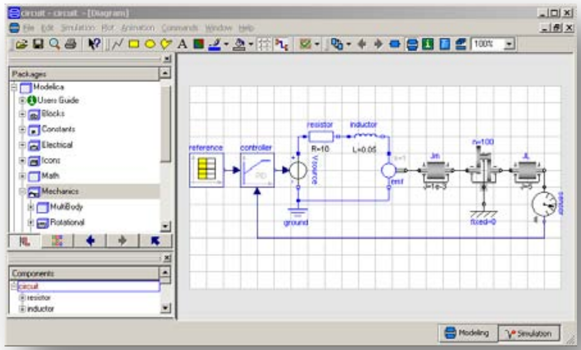
kofman@fceia.unr.edu.ar

Outline

- Modelica Language
- QSS methods and PowerDEVS
- Non-Discontinuous Modelica Models Simulation with QSS and DEVS
- OpenModelica to PowerDEVS (OMPD) Interface
- Simulation Results

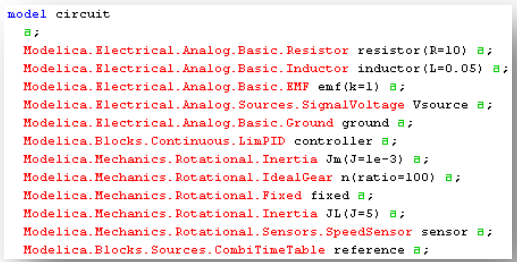
Modelica – The next generation modeling language

Graphical editor
for Modelica models



Modelica modeling environment
(free or commercial)

Textual description

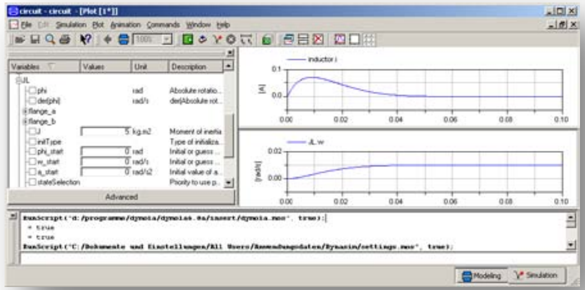


```
model circuit
  a;
  Modelica.Electrical.Analog.Basic.Resistor resistor(R=10) a;
  Modelica.Electrical.Analog.Basic.Inductor inductor(L=0.05) a;
  Modelica.Electrical.Analog.Basic.EMF emf(k=1) a;
  Modelica.Electrical.Analog.Sources.SignalVoltage Vsource a;
  Modelica.Electrical.Analog.Basic.Ground ground a;
  Modelica.Blocks.Continuous.LimPID controller a;
  Modelica.Mechanics.Rotational.Inertia Jm(J=1e-3) a;
  Modelica.Mechanics.Rotational.IdealGear n(ratio=100) a;
  Modelica.Mechanics.Rotational.Fixed fixed a;
  Modelica.Mechanics.Rotational.Inertia JL(J=5) a;
  Modelica.Mechanics.Rotational.Sensors.SpeedSensor sensor a;
  Modelica.Blocks.Sources.CombiTimeTable reference a;
```

Free Modelica language

MODELICA

Translation of Modelica models in C-Code and **Simulation**



```
RunScript("d:/programme/dymola/dymola6.0a/insert/dymola.mos", true);
 = true
 = true
RunScript("C:/Dokumente und Einstellungen/All Users/Anwendungsdaten/Dynasim/settings.mos", true);
```

Modelica simulation environment
(free or commercial)

QSS methods

Simulation of continuous systems by a digital computer requires discretization.

- Classical methods (e.g. Euler, Runge-Kutta etc.), that are implemented in Modelica environments, are based on **discretization of time**.
- On the other hand, the **Discrete Event System Specification (DEVS)** formalism, introduced by Zeigler in the 90s, enables the **discretization of states**.
- The **Quantized-State Systems (QSS)** methods, introduced by Kofman in 2001, improved the original quantized-state approach of Zeigler.
- **PowerDEVS** is the environment where QSS methods have been implemented for the simulation of systems described in DEVS.

QSS methods

Let’s assume that the OpenModelica compiler has performed all the preprocessing steps and reached a reduced ODE of the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t) \qquad \text{or} \qquad \begin{aligned} \dot{x}_1 &= f_1(x_1, \ldots, x_n, t) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, \ldots, x_n, t) \end{aligned}$$

Then the simulation environments (usually) provide solvers that evaluate the right-hand side of the above equation at discrete time steps t_k in order to compute the next value of the state variable x_{k+1} .

QSS methods

Contrary, in QSS methods the state variables have to be quantized in order to obtain piecewise constant trajectories.

Let $q_i(t)$ be piecewise constant approximations of $x_i(t)$. Then, we can write:

$$\begin{aligned} \dot{x}_1 &= f_1(x_1, \ldots, x_n, t) \\ &\vdots \\ \dot{x}_n &= f_n(x_1, \ldots, x_n, t) \end{aligned} \quad \Longrightarrow \quad \begin{aligned} \dot{x}_1 &= f_1(q_1, \ldots, q_n, t) \\ &\vdots \\ \dot{x}_n &= f_n(q_1, \ldots, q_n, t) \end{aligned}$$

and considering a single component we can define the "simple" DEVS models:

$$q_i = Q(x_i) = Q(\int \dot{x}_i \, dt)$$

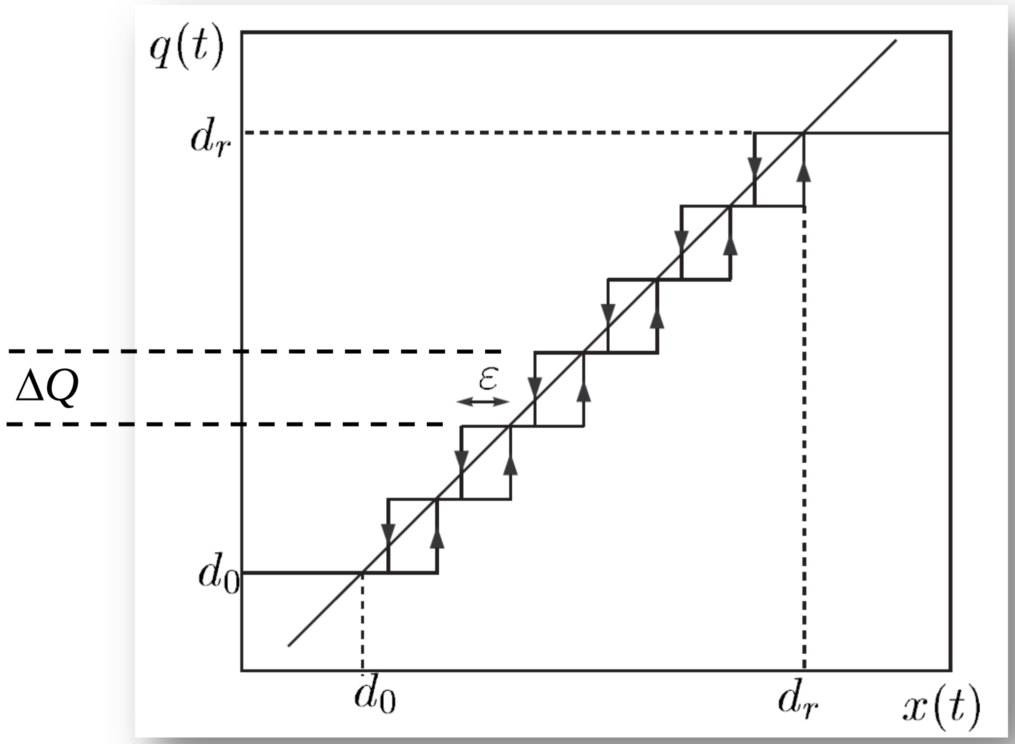
$$\dot{x}_i = f_i(q_1, \ldots, q_n, t)$$

Quantized Integrator

Static Function

QSS methods

In QSS methods function **Q** has been chosen to be a **Quantization Function with Hysteresis**.
Usually the discrete values Q_i are equidistant and $\Delta Q = |Q_k - Q_{k-1}|$ is called quantum.



QSS1 approximation

- **At each step only one (quantized) state variable that changes more than the quantum value ΔQ is updated producing a discrete event.**

QSS higher order methods

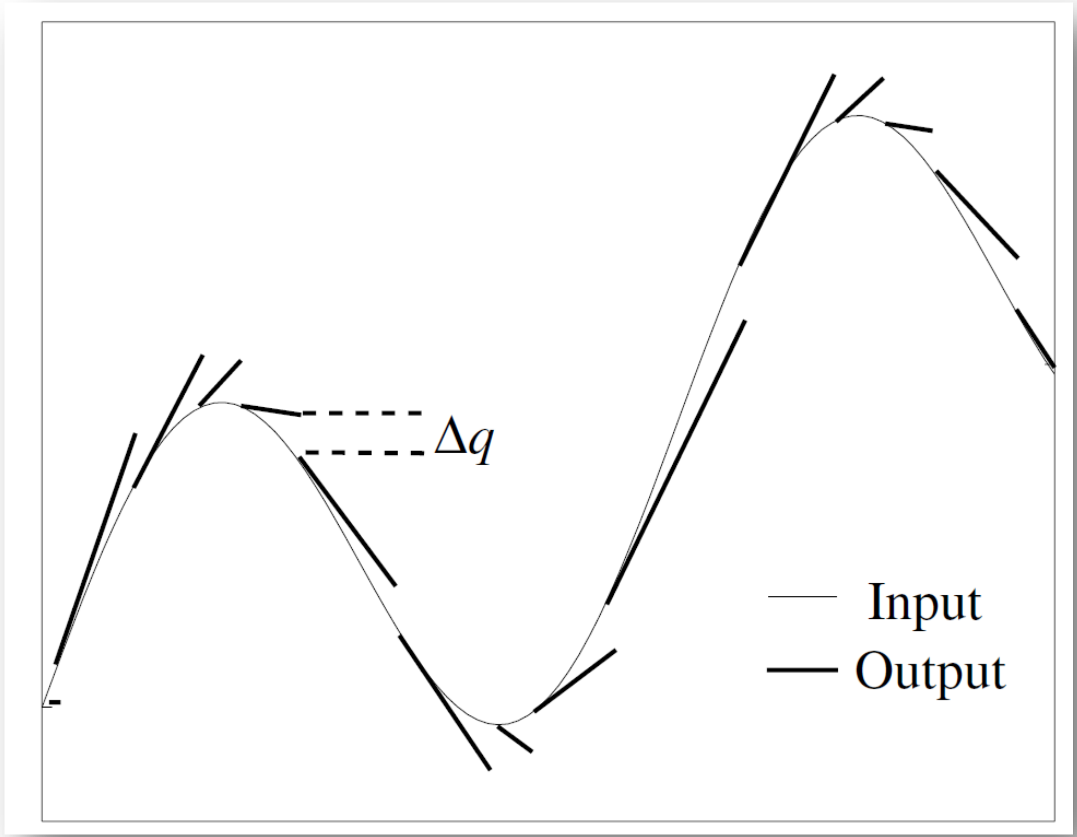
- **QSS is a first-order approximation of the real system trajectory.**

(In QSS quantized variables and state derivatives are piecewise constant, thus state variables piecewise linear).

- The problem is that both the approximation error and the time interval between successive events grows linearly with the quantum ΔQ . Thus reducing the error will cause a proportional increase of the computational time.
- There are also available **higher-order approximations (QSS2, QSS3)** that reduce the computation time.

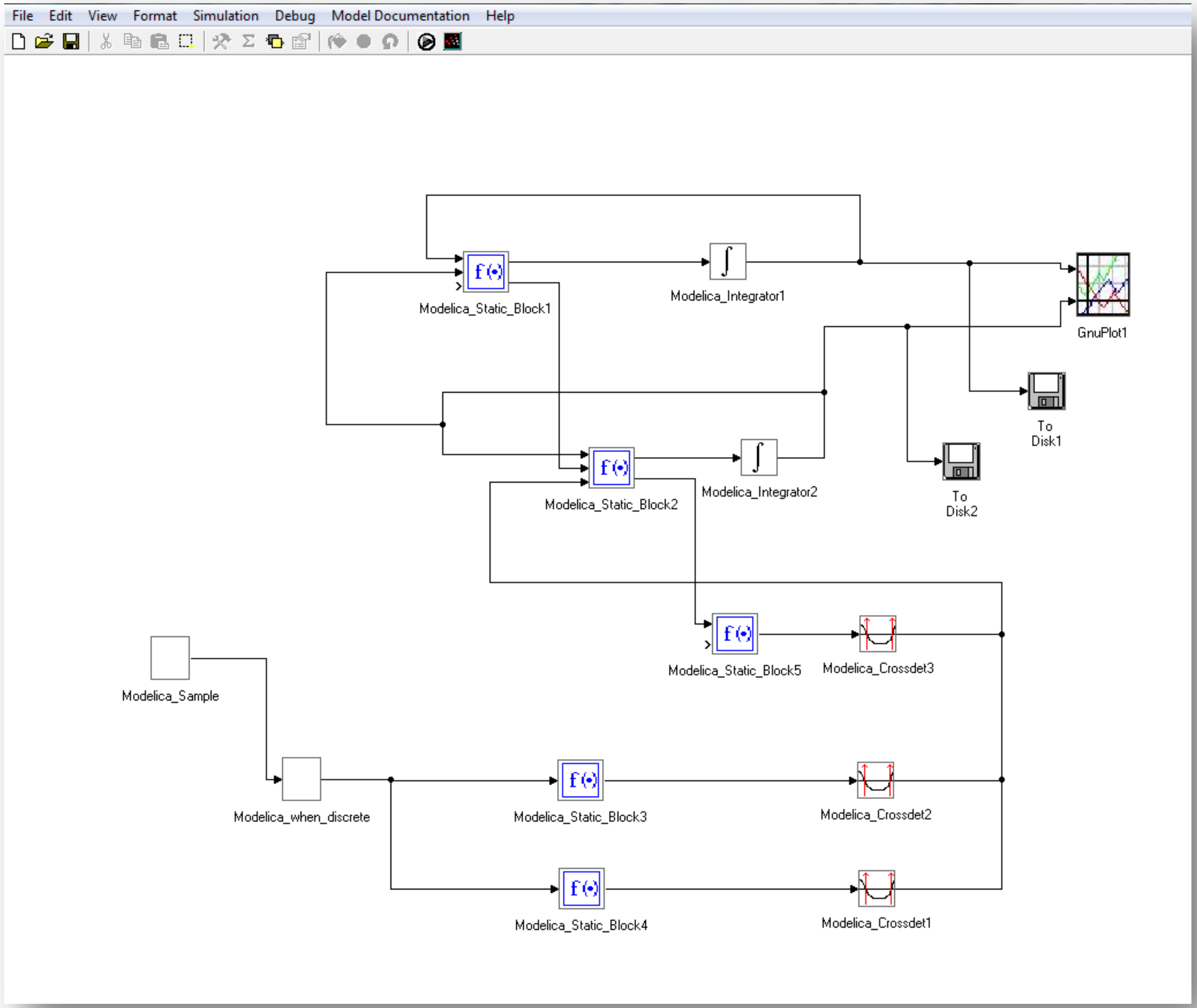
QSS higher order methods

- For example, in QSS2 the quantized variables have piecewise linear trajectories thus (in the linear case) the state derivatives are also piecewise linear and the state variables piecewise parabolic



QSS2 approximation

PowerDEVS



- Specify system structure (using DEVS formalism)
- Block implementation hidden (C++ code)
- Integrators implement the QSS methods
- Simulation using hierarchical master-slave structure and message passing
- Allows for real-time simulation

http://sourceforge.net/projects/powerdevs/

Simulate Modelica Models with QSS and DEVS

Thus, in absence of discontinuities, we can simulate our Modelica model using a coupled DEVS model consisting of coupling, n Quantized Integrators and n Static Functions, e.g.

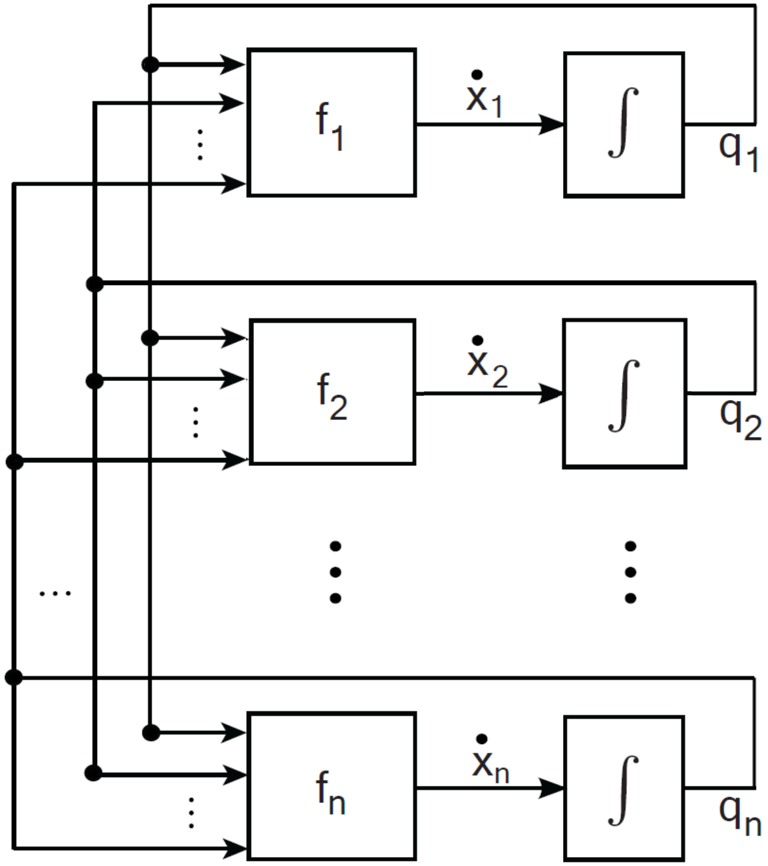
$$\dot{x}_1 = f_1(q_1, \ldots, q_n, t)$$

$$\vdots$$

$$\dot{x}_n = f_n(q_1, \ldots, q_n, t)$$

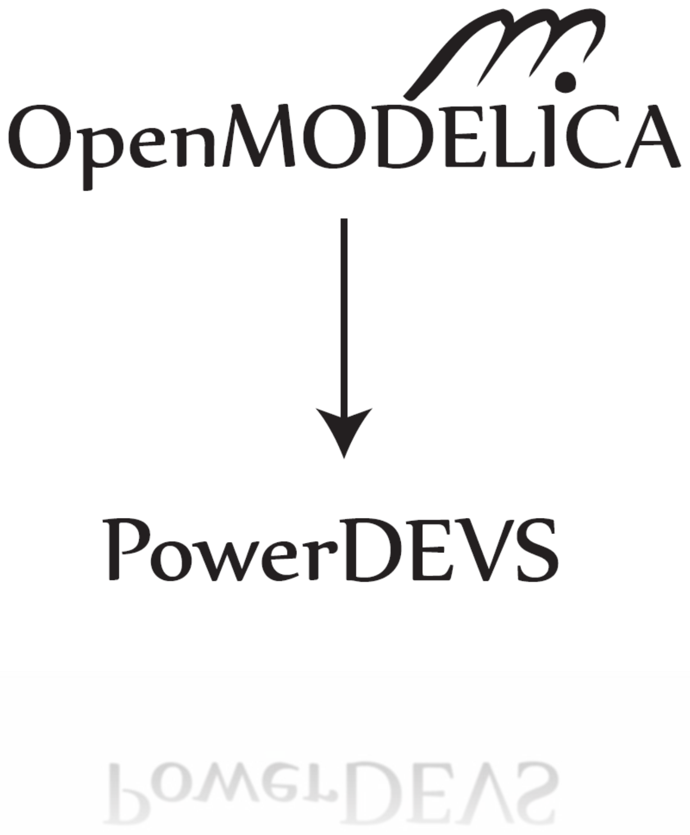
$$\dot{x}_i = f_i(q_1, \ldots, q_n, t)$$

$$q_i = Q(x_i) = Q\left(\int \dot{x}_i \, dt\right)$$



Goal

Design and implement an interface between OpenModelica and PowerDEVS

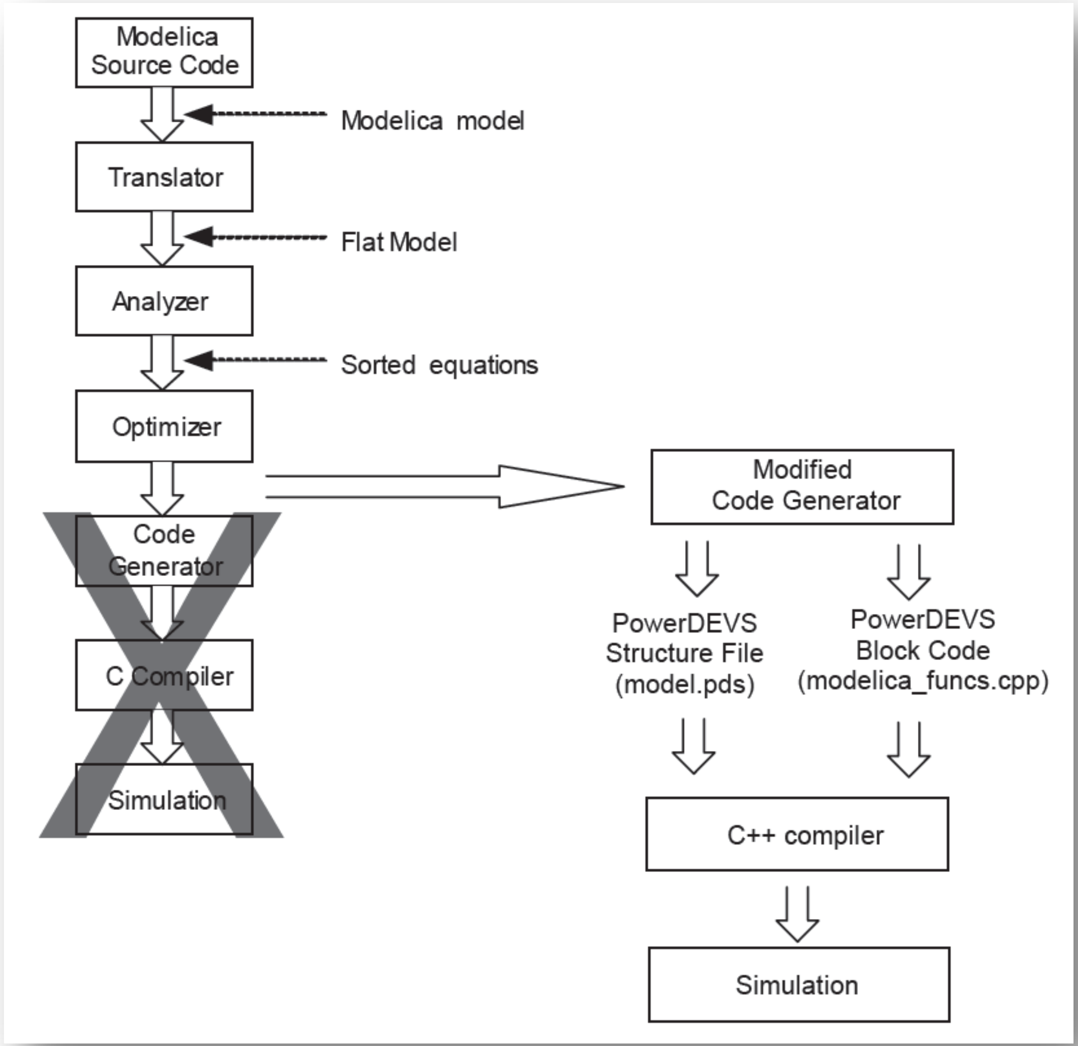


Goal

Interfacing OpenModelica and PowerDEVS we take advantage of

- The powerful modeling tools and market share offered by Modelica
  - Users can still define their models using the Modelica language or their favorite graphical interface.
  - No prior knowledge of DEVS and QSS methods is needed.
- The superior performance of quantization-based techniques in some particular problem instances
  - QSS methods allow for asynchronous variable updates, which potentially speeds up the computations for real-world sparse systems.
  - QSS methods do not need to iterate backwards to handle **discontinuities**, they rather predict them, enabling **real-time simulation**.

Interfacing OpenModelica and PowerDEVS (OMPD interface)



OMPD Interface (I)

1. **Equation splitting**

   The interface extracts the indices of the equations needed in order to compute the derivative of each state variable.

2. **Mapping split equations to BLT blocks**

   The splitted equations are mapped back to BLT blocks of equations. This is needed in order to pass this information to the OMC module that is responsible for solving algebraic loops.

3. **Mapping split equations to DEVS blocks**

   Since the state variables and the equations needed to compute them have been identified, they are assigned sequentially to static blocks in the DEVS structure.

4. **Generating DEVS structure**

   In order to correctly identify the DEVS structure of the model, the **dependencies** between the state variables that are computed in each static block have to be resolved. This is achieved employing the incidence matrix and the mapping of equations to DEVS blocks.

OMPD Interface (II)

5. **Generating the .pds structure file**

   Based on the DEVS structure it is straightforward to generate the .pds structure file.

6. **Generating static blocks code**

   In this step the functionality of each static block is defined. Each static block needs to know its inputs and outputs, supplied by the DEVS structure, as well as the BLT blocks needed to compute the corresponding state derivatives. Then the existing code generation OMC module is employed to provide the actual simulation code for each static block.

7. **Generating the .cpp code file**

   The code for the static blocks is ouput in the .cpp code file along with other information needed by PowerDEVS.

OMPD Interface - Example Model

- Various tests have been performed to ensure that the implemented OMPD interface is working correctly.
- Here we present the results of processing the following second-order model through the implemented interface:

```
model M1
  Real x1;
  Real x2;
  Real u2;
equation
  der(x1)=x2-x1/10;
  der(x2)=u2-x1;
  u2=(1-u2-x2)^3;
end M1;
```

OMPD Interface - Example Model

```
model M1
  Real x1;
  Real x2;
  Real u2;
equation
  der(x1)=x2-x1/10;
  der(x2)=u2-x1;
  u2=(1-u2-x2)^3;
end M1;
```

Information extracted by OMC compiler :

```
Variables (3)
=========
1:  $u2:VARIABLE sys3, Real type: Real ...
2:  $x2:STATE sys3, Real type: Real  ...
3:  $x1:STATE sys3, Real type: Real ...

Equations (3)
=========
1 : $u2 = (1.0 - $u2 - $x2) ^ 3.0
2 : $DER$x1 = $x2 - $x1 / 10.0
3 : $DER$x2 = $u2 - $x1

BLT blocks
2,
1,
3,
```

In other words OMC has identified that model M1 has:

- 2 state variables
- 1 algebraic variable

and 3 equations:

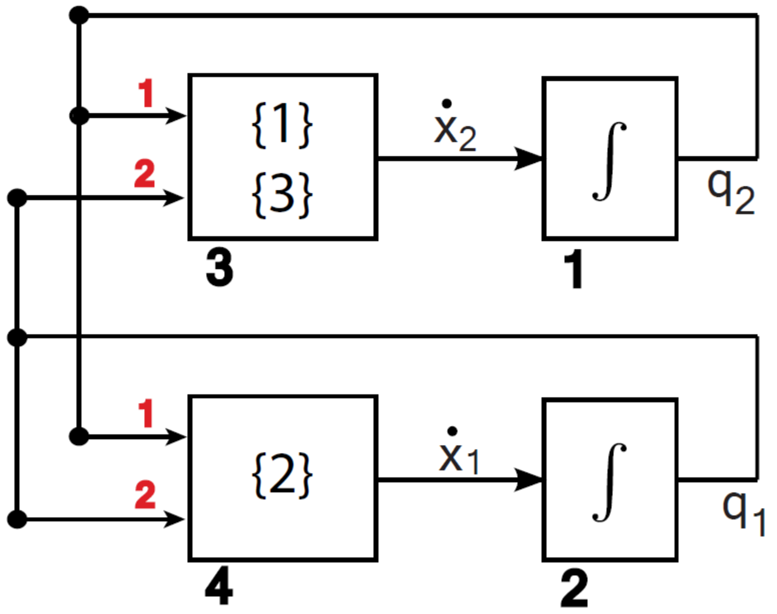
$$u_2 = (1 - u_2 - x_2)^3 \qquad \{1\}$$
$$\dot{x}_1 = x_2 - \frac{x_1}{10} \qquad \{2\}$$
$$\dot{x}_2 = u_2 - x_1 \qquad \{3\}$$

organized in 3 BLT blocks.

OMPD Interface - Example Model

```
model M1
  Real x1;
  Real x2;
  Real u2;
equation
  der(x1)=x2-x1/10;
  der(x2)=u2-x1;
  u2=(1-u2-x2)^3;
end M1;
```

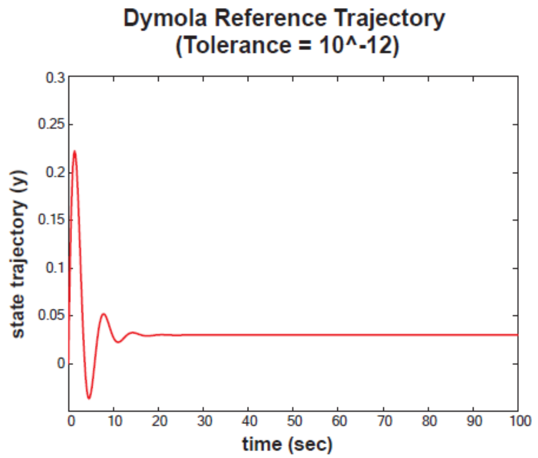
Automatically generated DEVS structure



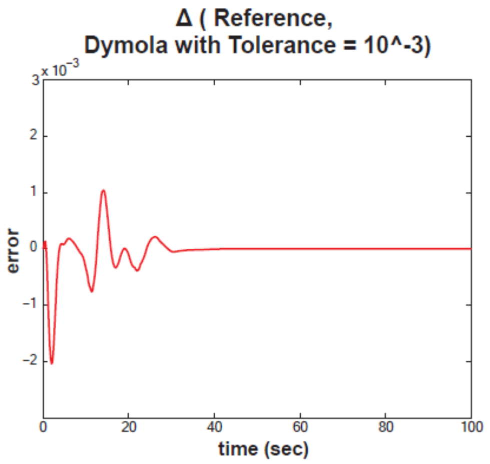
$$u_2 = (1 - u_2 - x_2)^3 \qquad \{1\}$$
$$\dot{x}_1 = x_2 - \frac{x_1}{10} \qquad \{2\}$$
$$\dot{x}_2 = u_2 - x_1 \qquad \{3\}$$

OMPD Interface - Example Model

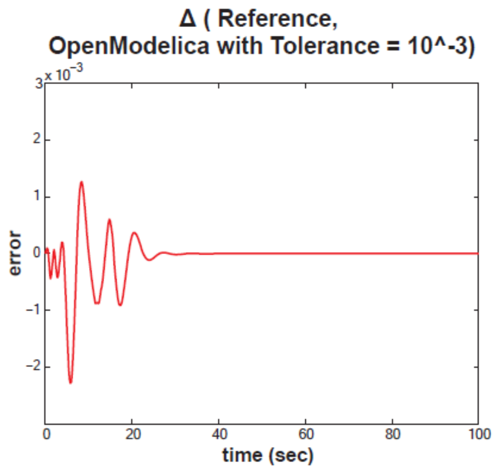
- Simulation Results for model M1.



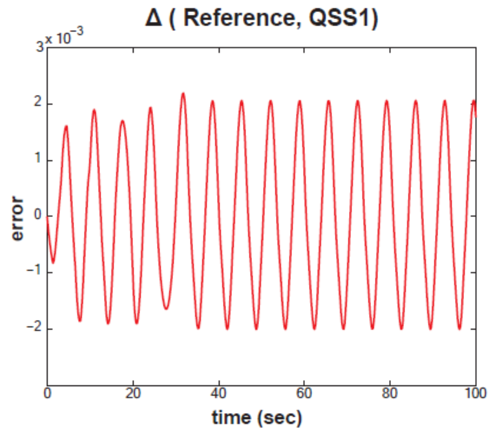
(a)



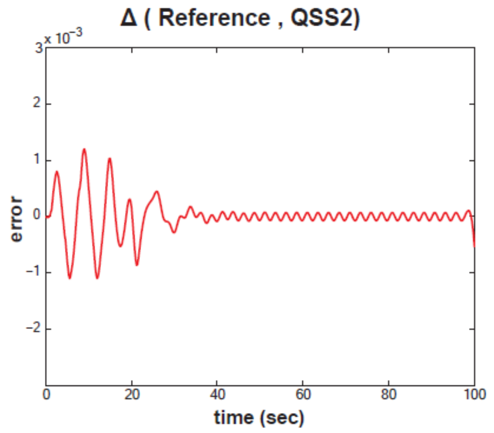
(b)



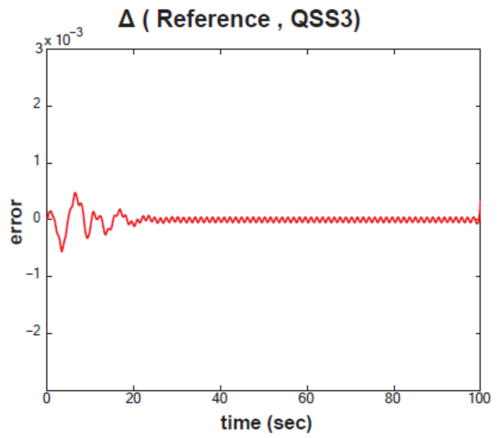
(c)



(d)



(e)



(f)

Simulation Results – Benchmark Framework

- We want to compare the performance of the standard DASSL solver of OpenModelica with the QSS3 method implemented in PowerDEVS.
- More specifically we would like to investigate the performance of both algorithms for **non-stiff models of different size and sparsity without discontinuities**.
- To achieve that we generate linear test models of the form:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} \qquad \text{where } \mathbf{x} \in \mathbb{R}^n \text{ is the vector of state variables}$$

In order to control the eigenvalues of the system matrix A is constructed as follows:

1. Generate real-valued random eigenvalues drawn from a Gaussian distribution : $eig \sim -\mathcal{N}(5,2)$
2. Create a diagonal matrix $\mathbf{D} = diag(eig)$
3. Create a random orthogonal matrix $\mathbf{M}$
4. Then, matrix $\mathbf{A} = \mathbf{M} \cdot \mathbf{D} \cdot \mathbf{M}^T$ has the desired eigenvalues .

- **Sparsity s** is defined as the number of connections to each state variable.

  To achieve a certain sparsity level **s**, we set the n-s absolute smallest elements of each row of **A** to zero.

Simulation Results – Benchmark Framework

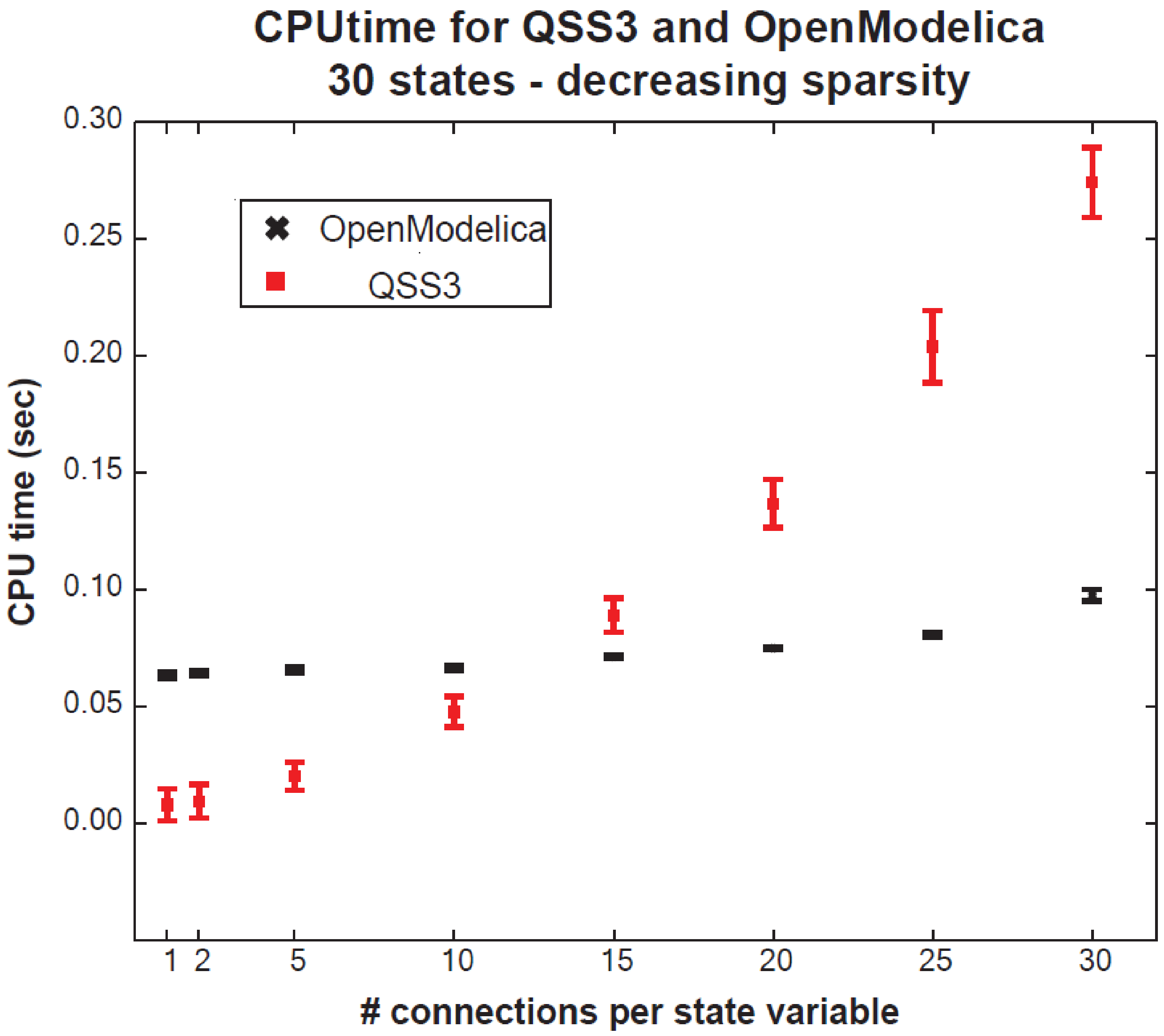
For the comparison simulations the following settings were used:

- OpenModelica 1.5.0 with DASSL as the standard solver
- PowerDEVS with QSS3
- Tolerance was set to 10^{-3} for both environments.
- Simulation End-Time was set to 3 sec.
- The output file generation was disabled.

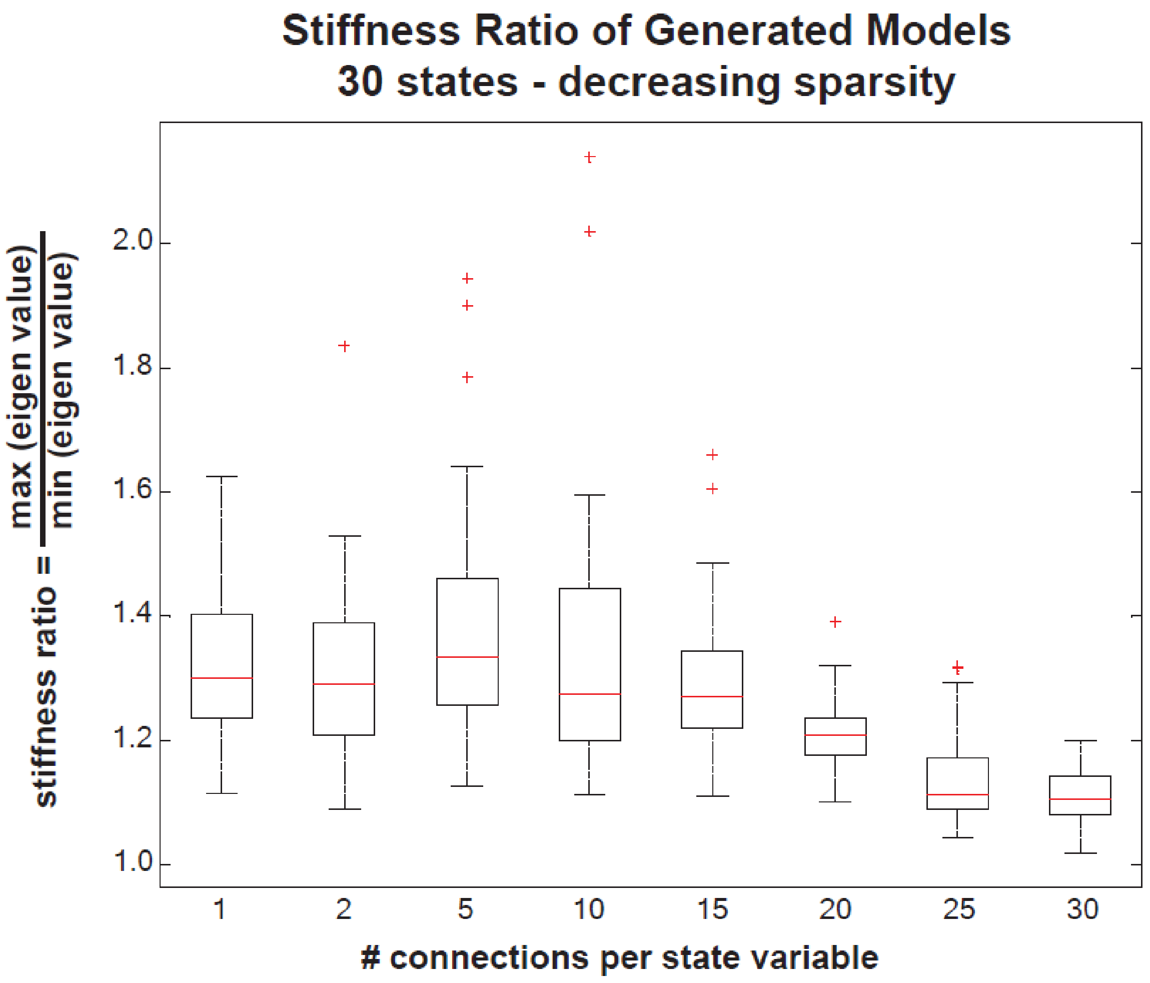
For each parameter configuration (n,s)

- 100 Modelica models were randomly generated and given as input to the standard OMC and via the OMPD interface to PowerDEVS.
- The CPU time needed for the simulation was measured for each generated executable.
- To obtain more reliable results, each simulation was repeated 10 times and the median was considered as the measured CPU time.
- Finally, the mean of the measurements is reported along with +/- 1 standard deviation.

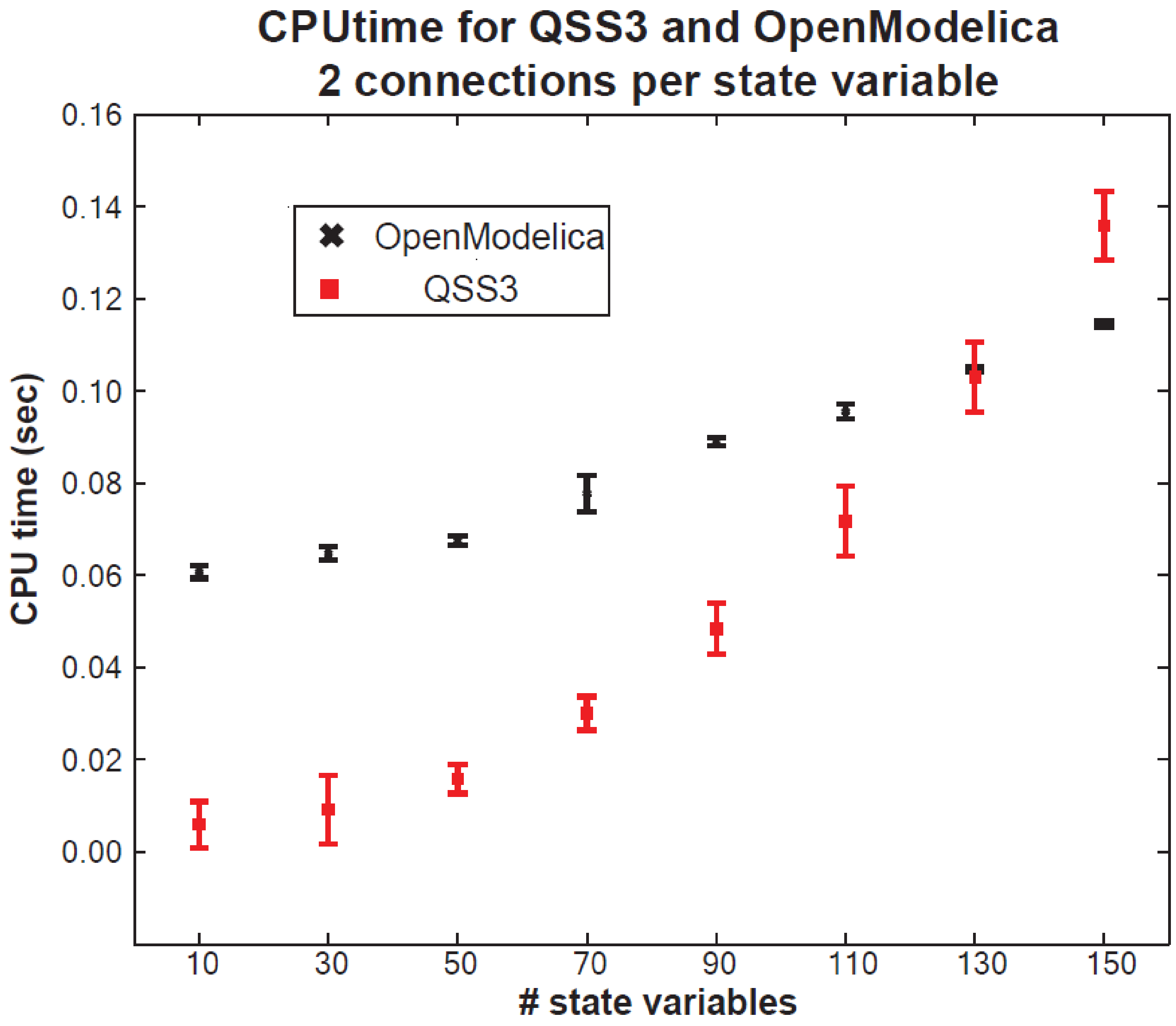
Simulation Results – Fixed Number of States



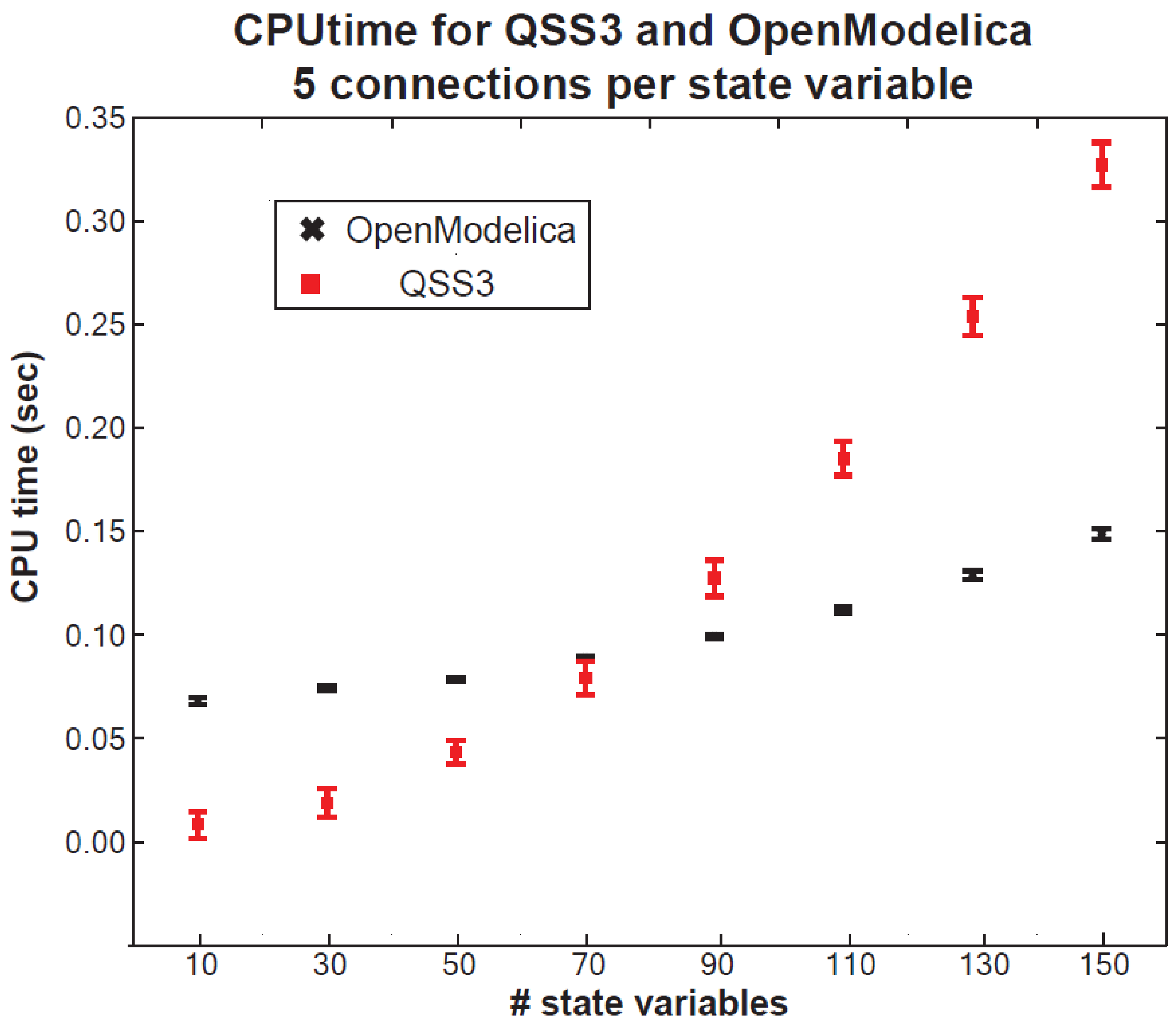
Simulation Results – Fixed Number of States



Simulation Results – Fixed Sparsity



Simulation Results – Fixed Sparsity



Summary

- **Goal:**
  - Interface OpenModelica and PowerDEVS.
  - Compare QSS methods with standard DAE solvers.
- **Motivation:**
  - Established and easy model definition in Modelica.
  - Exploit the superiority of QSS methods in certain classes of problems without the overhead of a manual conversion.
  - Give the user the freedom to simulate either in a discretized-time or discretized-state world.
- **Current status:**
  - All models without discontinuities are handled correctly by the interface.
  - Preliminary results demonstrate the superiority of QSS methods in sparse systems.
- **Future work:**
  - Extend the interface to cover cases with discontinuities.
  - Further investigate the performance of QSS methods compared to DASSL for various classes of large-scale, real-world problems.

Acknowledgements

To the PELAB group at Linköping University and in particular:

- Per Östlund
- Adrian Pop
- Martin Sjölund
- Prof. Peter Fritzson

QUESTIONS ?