Object-Oriented Implementation of a Simulator for Linear Implicit Equilibrium Dynamics

Dirk Zimmer [1*]

[1]Institute of System Dynamics and Control, DLR, Münchener Straße 20, 82234 Weßling, Germany
[*]*dirk.zimmer@dlr.de*

Abstract. Models based on linear implicit equilibrium dynamics form a special but very useful sub-set of DAE systems that can be applied for the simulation of technical physical systems. Since these are based on differential-algebraic equations, a transformation into executable code must be performed for the purpose of model evaluation. One way to achieve this is the object-oriented formulation of the simulation code itself. To explore this path a dedicated simulator prototype has been implemented and is outlined here. The long-term goal is to define alternative compilation targets for a Modelica compilers that enable highly scalable simulation code for very large systems.

1 What is Linear Implicit Equilibrium Dynamics?

Linear Implicit Equilibrium Dynamics (LIED) is technically defined as a special class of Differential Algebraic Equation (DAE) Systems.

1.1 Formal Definition

A DAE system with potential state derivatives $\dot{\boldsymbol{x}}$, time t and algebraic variables $\mathbf{w}$

$$\mathbf{0} = \boldsymbol{F}(\dot{\boldsymbol{x}}, \boldsymbol{x}, \mathbf{w}, t)$$

is defined as LIED system when it can be transformed into the following form:

$$\begin{bmatrix} \mathbf{w}_E \\ \dot{\boldsymbol{x}}_E \end{bmatrix} = \mathrm{g}(\boldsymbol{x}_I, \boldsymbol{x}_E, t)$$

$$\mathbf{A}(\boldsymbol{x}_I, \boldsymbol{x}_E, \mathbf{w}_E) \begin{bmatrix} \mathbf{w}_I \\ \dot{\boldsymbol{x}}_I \end{bmatrix} = \mathrm{f}(\boldsymbol{x}_I, \boldsymbol{x}_E, \mathbf{w}_E, t)$$

We see that both the algebraic variables as well as the state derivatives can be split into a fully explicit part $(\dot{\boldsymbol{x}}_E;\ \mathbf{w}_E)$ and a part $(\dot{\boldsymbol{x}}_I;\ \mathbf{w}_I)$ with a linear system in implicit form expressed by the regular matrix $\mathbf{A}$. Furthermore, the following conditions shall hold true:

$$\dot{\boldsymbol{x}}_E \cap \dot{\boldsymbol{x}}_I \subseteq \dot{\boldsymbol{x}}$$
$$\mathbf{w}_E \cap \mathbf{w}_I \supseteq \mathbf{w}$$
$$\dot{\boldsymbol{x}}_E \cap \dot{\boldsymbol{x}}_I \cap \mathbf{w}_I \supseteq \dot{\boldsymbol{x}}$$
$$\dot{\boldsymbol{x}}_E, \dot{\boldsymbol{x}}_I, \mathbf{w}_E, \mathbf{w}_I \text{ are all disjoint}$$

These conditions essentially mean that it is allowed to perform certain symbolic mechanism of index reduction such as the dummy derivative method [5] originating from Pantelides [6]. Using this method, states variables of $\boldsymbol{x}$ can be transformed to algebraic variables in $\mathbf{w}_I$ and further derivatives may be added to $\mathbf{w}_I$ or $\mathbf{w}_E$. In practice, this is important because it means that the linear implicit dynamics can be expressed by far fewer states than suggested by the vector $\boldsymbol{x}$ of the original DAE formulation.

1.2 Informal Explanation

The formal definition above may be primarily perceived as a relatively strong restriction on the model equations and not many systems may be intuitively expected to fall into this category. Surprisingly, LIED can be applied successfully for the object-oriented modelling of complex thermofluid architectures [7],[8] or to mechanical systems with stiff contacts [9].

The idea is that the non-linear behaviour of the slow mode is explicitly expressed whereas the fast dynamics that typically is needed to uphold non-linear constraints is expressed by a linear implicit system that fulfils the constraint in its equilibrium. Hence the name: linear implicit equilibrium dynamics. The equilibrium dynamics is thereby often a replacement dynamic and only an approximation of reality (as all modelling is).

2 What is LIED good for?

As the above references demonstrate, LIED has been applied using Modelica [1] for the object-oriented modeling of thermofluid or mechanical systems.

To this end, it is necessary to use triplets as interface of the model components that consist in a signal for the explicit non-linear part and a pair of potential as presented in Table 1.

Domain	Signal	Potential	Flow
trans. mechanics	position: r [m]	velocity: v [m/s]	force: f [N]
rotational mechanics	angle: φ [rad]	angular velocity: ω [rad/s]	torque: τ [Nm]
thermofluid streams	Thermo-dynamic state: $\hat{\Theta}$	inertial pressure r [Pa]	mass-flow rate: $\dot{m}$ [kg/s]

Table 1: Connection triplets for the object-oriented modelling of LIED Systems.

More background on the derivation of these triplets can be found in [10]. It goes beyond the scope of this paper how the equations are formulated in detail but the two Modelica model diagrams shown in Figure 1 and 2 may illustrate the practical usefulness.

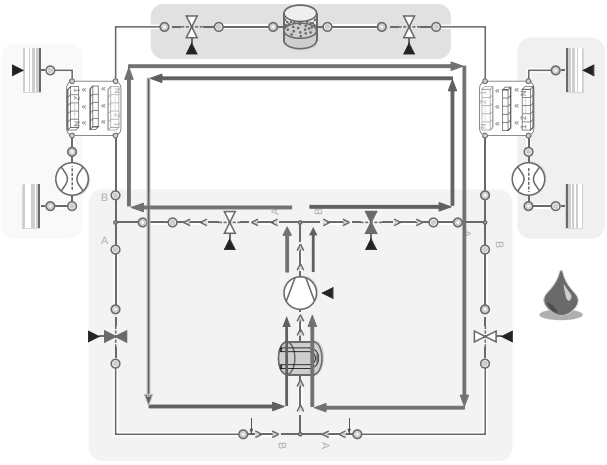

Figure 1: Model diagram of a reversible heat pump systems using the ThermoFluid Stream Library [8].

Especially the ThemoFluid Stream Library has meanwhile become a popular OpenSource library both by academia [4] and by industry [7].

LIED systems have very benevolent characteristics for object-oriented modelling. Following simple connection rules, the resulting matrix $\mathbf{A}(\boldsymbol{x}_I, \boldsymbol{x}_E, \mathbf{w}_E)$ will be regular and an a-priori statement on solvability can be given [7]. This makes this class of modeling very robust and prevents many computational simulation errors.

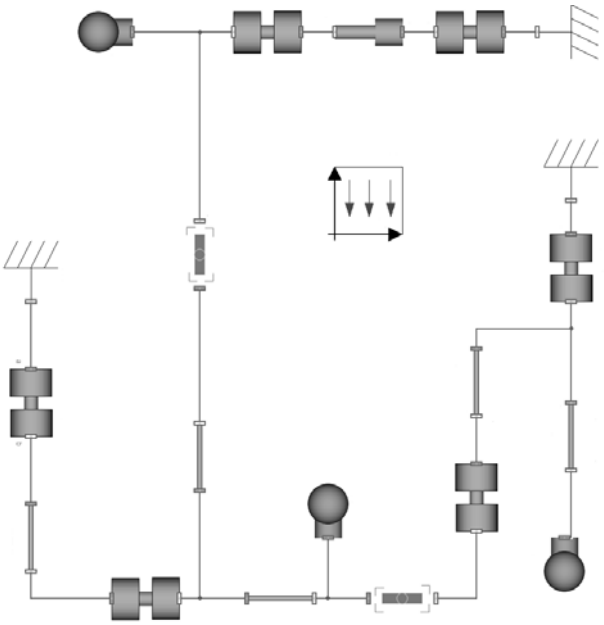

Figure 2: Model diagram of a mechanical system for a kinematic using the Dialectic Mechanics library [9].

3 A dedicated simulator

3.1 Why?

The libraries are currently developed using Modelica and since LIED systems are a subset of DAE systems and Modelica can deal with DAEs in general, any Modelica compiler can be used to generate simulation code out of LIED systems. A dedicated simulator is thus not needed. Existing and mature simulation software can be used.

However, Modelica compilers are very complex and algorithms for the processing of general DAEs involve drawbacks: all Modelica compilers create a “flat” model, where all equations are collected in a single set. For very large systems, this poses problems primarily because large amounts of code are generated.

Out of practical experience, many LIED systems have very benevolent structural characteristics that enable a compilation already on the component level. A compiler for standard Modelica cannot exploit these benefits, albeit they may greatly increase the ability to deal with large systems or with variable structure systems.

To explore alternative compile targets dedicated for LIED systems, a prototypical simulator has been developed with the name `zimsim`.

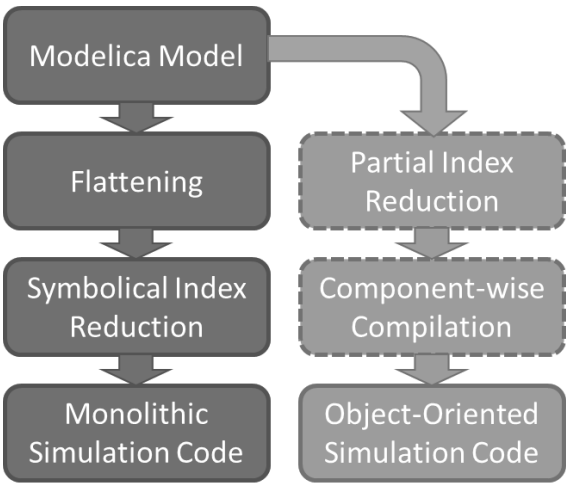

Figure 3: Illustration of different pathways for the compilation of simulation code. The blue boxes on the left illustrate the classic compilation of Modelica where all equations are collected in a single set. The green boxes on the right illustrate a potential pathway for LIED systems. This paper focuses on the compile target which is the object-oriented simulation code.

3.2 Computational structure and implementation of blocks

Figure 5 presents a simple planar mechanical system of a crane crab (a pendulum attached to a slider). To each component of the system 3 computational blocks are assigned of different color: blue, green, and orange. The blue and green blocks form thereby a computational sequence directed from the root whereas the orange signal leads to the root. For this particular domain of LIED systems, the blue signal contains the positional state and undergoes non-linear transformations. The transformation of the green and orange signal forms a linear response which can be used to solve the linear equations system that spans across the components.

```
class RevoluteJoint: public Component
  Signal flangeTo, flangeOn
  double phi, phi_der, w, z, residualTorque;
  ContinuousState position{ phi,phi_der };
  ContinuousState velocity{ w,z };
  Tearing zeroTorque{ phi_der, residualTorque };
  void evalState();
  void evalKinetic();
  void evalImpulse();
  virtual void metainfo(Meta& meta)
```

Figure 4: C++ class diagram of a revolute component.

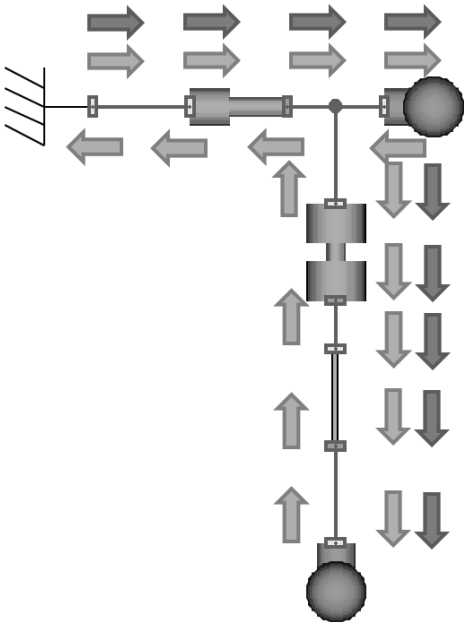

Figure 5: Crane crab modelling diagram

In practice this means that the code for each component can be represented by dedicated C++ class and the computational blocks can be expressed by C++ member functions. This is illustrated in Figure 4 with matching colors.

3.3 Handling of meta information

It is necessary to register the state-variables and their derivatives in the system. This is done by member objects. In similar vein, tearing variables and residuals are registered. The tearing variable belongs to the green signal and used to probe the system whose linear response can be assessed by the residual belonging to the orange signal flow. Furthermore, for each block (or member function) the dependence on the input and output signals needs to be registered.

For these purposes each component class must contain a virtual member function called `metainfo`. This function takes a crawler object by reference and depending on the implementation of the crawler different metainformation may be extracted.

3.4 Ordering of blocks, compacting and extracting of sub-blocks.

When instantiating a component class, all structural relevant meta data are collected. With this information, it is now possible to put all calls to member functions of all components in correct order. First a partial order is created based on the signal dependence, then the linear systems (of different dimension) marked by tearing and

residual variables are compacted and in the last stage the dynamic part is compacted (meaning that the all constant evaluations are placed upfront and all calls not needed for derivative evaluation are put last). With this ordering the linear subsystems can be constructed and solved and the overall model can be evaluated.

3.5 ODE simulation

Runge-Kutta solvers with fixed step size of order 1 to 4 as well as Backward Euler and ESDIRK23 [3] with variable step-size control have been implemented as numerical ODE solvers.

3.6 Overall Simulator Software Design

The major classes of the simulator are:

- `ModelEvaluation` instantiates the model, collects structural meta-information, orders the computational blocks and offer model evaluation
- `Simulator` implements numerical ODE solver
- `Recorder` collects references to desired outputs and generates output either to file, memory or TCP port.

4 First scaling results

4.1 The scaling experiments

The crane crab model of Figure 5 has been used to perform a simple scaling experiment. Using a logarithmic grid of factor 4, 1 to 16384 crane crabs have been instantiated and simulated using `zimsim`. The same exercise has been performed within a commercial Modelica tool. Here, this generates models ranging from 206 equations up to 3.1 million equations. In Figures 6 and 7, we use the equation number since this quantity is more familiar to Modelica users.

4.2 Scaling results in terms of memory usage

Peak memory usage for model translation, compilation and simulation is orders of magnitudes lower using `zimsim`. The likely reasons are:

- Avoidance of flattening
- On demand generation of meta-information
- More variables on stack than on heap

For small models, comparing the memory usage hardly makes sense because we compare the memory usage of a dedicated simulator (LIED) with the one of a whole modeling and simulation environment (Modelica Tool). The two orders of magnitude for small models is thus not surprising. What is surprising that this gap does not significantly close for larger models. Peak memory consumption seems to appear in the models at model translation. Zimsim has been written with efficient use of memory in mind: Meta information is generated only on-demand and also the object-oriented formulation enables to do more computation on the steak than on the heap.

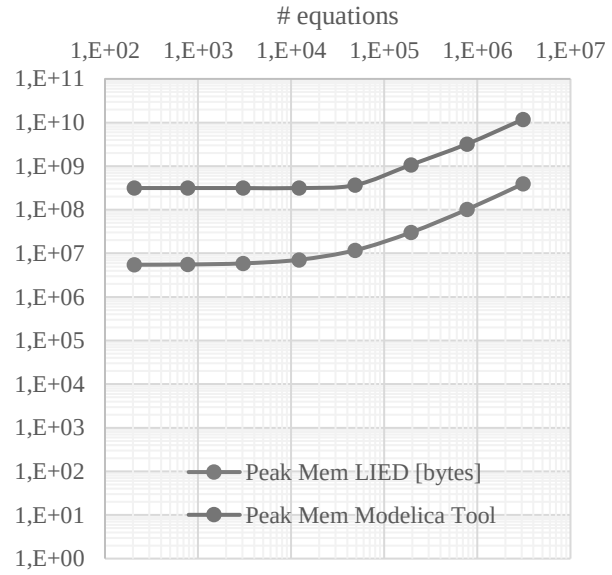

Figure 6: Memory usage in bytes

4.3 Scaling results in terms of performance

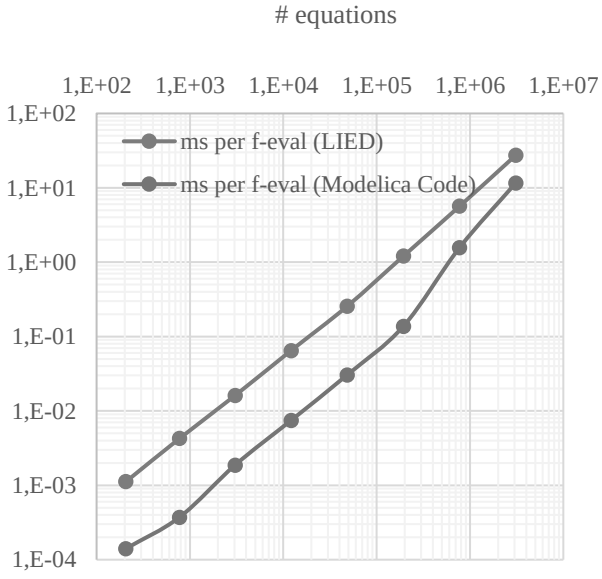

Figure 7: Time for single model evaluation in ms.

Simulation speed is up to an order of magnitude slower using `zimsim`. The likely reasons are:

- More overhead in `zimsim` due to interface variables and many function calls.
- Numerical solution of linear equation system in `zimsim` instead of symbolical transformation.

Pure simulation time is compared here. Time for translation, compilation and instantiation is ignored. Generation of output has been reduced to a negligible amount.
One has to consider that the code generation of the Modelica tool underwent decades of optimization whereas zimsim is still prototypical. A reduction of 50% in computational effort might be achievable for `zimsim`, however a certain gap will always remain.

We can observe that for larger systems, the performance penalty is smaller, the reason is unknown but is probably due the higher memory efficiency in `zimsim`.

A final remark: results for even larger systems could not be attained for `zimsim` as well as for the Modelica tool, however for completely different reasons. The Modelica tool started to hit the limits of memory leading to excessive translation time due to disk swapping. `zimsim` had no such problems but the instantiation has been implemented with a low-performing algorithm, requiring too much time. This is however a pure implementation issue and will be improved.

5 Outlook

The prototypical simulator demonstrates that LIED system can actually be coded directly in C++ and executed with acceptable efficiency and while using comparably little memory. The complexity of the software is thereby significantly lower than of any Modelica environment. The scaling capabilities are promising and can be improved by going to code dedicated for GPUs.

Although the modeling in C++ turned out to be much more natural than originally conceived, it is still not a desirable target. Instead, the existing C++ modelling libraries shall be used as an inspiration for Modelica compilers. Especially open compilers such as the OMC [2] could be modified to compile from Modelica libraries into a set of pre-compiled components.

The main, albeit preliminary, conclusion is that LIED systems not only form an interesting class for robust modelling but also an interesting class for large scale system simulation, worthy of further investigation.

References

[1] Fritzson, P., *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3*, 2nd ed., IEEE Press, Piscataway, New Jersey, 2014, pp. 1256.

[2] Fritzson, Peter A. et al. (2019) "The OpenModelica Integrated Modeling, Simulation, and Optimization Environment." *Proceedings of The American Modelica Conference 2018*, October 9-10, USA

[3] Jørgensen, J.B., Kristensen M. R. and Grove, P. (2018) A Family of ESDIRK Integration Methods. *arXiv Numerical Analysis* eprint :1803.01613

[4] Junglas, P. (2023) Implementing Thermodynamic Cyclic Processes Using the DLR Thermofluid Stream Library. *Simulation News Europe* E 33(4)

[5] Mattsson, S.E., Gustaf Söderlind (1993). "Index Reduction in Differential-Algebraic Equations Using Dummy Derivatives" In: *SIAM Journal on Scientific Computing* 1993 14:3, 677-692

[6] Pantelides, C. (1988), The consistent initialization of differential-algebraic systems, SIAM J. Sci. Statist. Comput., 9, 213–231

[7] Zimmer, D. (2020), Robust Object-Oriented Formulation of Directed Thermofluid Stream Networks . *Mathematical and Computer Modelling of Dynamic Systems*, Vol 26, Issue 3.

[8] Zimmer, D., N. Weber, M. Meißner (2022) The DLR ThermoFluid Stream Library. *MDPI Electronics - Special Issue*.

[9] Zimmer, D., C. Oldemeyer (2023). "Introducing Dialectic Mechanics". *Proceedings of the 15th International Modelica Conference*, Aachen.

[10] Zimmer (2024) Object-Oriented Modeling of Classic Physical Systems using Linear Implicit Equilibrium Dynamics *Preprints* 2024, 2024031139