Model Disambiguation Technology in MWORKS.Sysplorer

Zhipeng Chen[1] Zhichao Huang[1] Chong Zhou[1] Yinqi Chen[1] Qi Liu[1] Fanli Zhou[1] Liping Chen[2]

[1]Suzhou Tongyuan Software & Control Technology Co., Ltd., Suzhou, China , {chenzhipeng, huangzc, zhouc, chenyq, liuq, zhoufl}@tongyuan.cc
[2]Huazhong University of Science and Technology, Wuhan, China , chenlp@hust.edu.cn

Abstract

Modelica models exhibit excellent cross-platform compatibility (they can be compiled and simulated on any platform supporting Modelica). However, experiments have revealed that simulation results of the same Modelica model may vary across different platforms (under identical simulation algorithm configurations). The root cause of such discrepancies lies in model translation uncertainty introduced by improper modeling practices, such as insufficient initial constraints or ambiguous state variables selection. Different Modelica tools may have different translation strategies. Therefore, model disambiguation should be performed to ensure consistent simulation results. It can be addressed by three parties: through language improvements, such as the proposal of relevant annotations, by vendor tools; and by the modelers through manual intervention. This paper presents a model disambiguation technology in MWORKS.Sysplorer that enables modelers to automatically correct model text based on translation information, eliminating uncertainties and ensuring model portability across Modelica platforms.

Keywords: Model Disambiguation, Cross-platform, Modelica, MWORKS.Sysplorer

1 Introduction

Modelica's declarative and object-oriented modeling approach lowers the barrier to modeling complex systems. Specialized model library vendors, research teams, and open-source communities provide foundational model components, which users then leverage to build their own research objects. These foundational components often come with default configurations, such as default start values and state variables selection (Mattsson and Söderlind 1993). Modelers must adjust these settings according to their specific application scenarios, including designing necessary initial constraint equations to ensure their Modelica model satisfies well-defined mathematical constraints.

Although Modelica platforms can still translate and simulate models based on default configurations, such models may yield different simulation results across different platforms. These discrepancies can arise from platforms selecting different state variables or applying non-equivalent initial constraint conditions. Additionally, most modern Modelica platforms support nonlinear equation tearing techniques, but implementation differences may lead to the selection of distinct nonlinear iteration variables. When nonlinear equations have multiple solutions, variations in the chosen iteration variables and their initial estimates can further affect simulation outcomes.

To ensure Modelica model portability across platforms, potential translation ambiguities must be resolved during the modeling phase. However, most modelers have limited knowledge of Modelica's translation and solving theory, making it difficult to construct ideal models. Moreover, for complex models (e.g., those with hundreds or thousands of state variables), relying on modelers to eliminate uncertainties is neither reliable nor cost-effective.

The paper is organized as follows. Section 2 uses a series of extremely simple examples to systematically dissect the sources of model translation uncertainty. Section 3 presents MWORKS.Sysplorer's model disambiguation technology and proposes standardized Translation annotation semantics for Modelica specifications. Section 4 empirically demonstrates the practical effects of model disambiguation through experimental validation.

2 Model Translation Uncertainty

Model translation uncertainty primarily stems from three aspects: lack of initial equations, selection of state variables, and nonlinear tearing. We will provide a detailed explanation using a series of extremely simple examples (though lacking engineering significance) in separate subsections, demonstrating how it leads to inconsistent simulation results.

2.1 Lack of Initial Equations

An initial system in Modelica remains valid even with missing but consistent equations (Pantelides 1988), as tools automatically supplement them. Modelica specifies that start values should be used to supplement initial values. If no start value is available, default values are applied (0 for Real and Integer types, false for Boolean types). When selecting m equations from n candidates with $0 < m < n$, uncertainty arises. Different tools have significant discretion in how they perform this completion, and the underlying mechanisms they employ are generally not the same. Vendors should report supplemental information to alert modelers.

2.2 Start Value Recommended Priority

As "Start Value Recommended Priority"(see §8.6.2 (Modelica Association 2023)) states, the start values from models closer to the top level are considered more reliable. However, some vendor tools may not follow this recommendation for various reasons. Subsequent tool upgrades adopted this specification-recommended strategy for selecting start values, resulting in incompatibility with older versions.

Listing 1. An example of model translation uncertainty caused by lack of initial equations with multiple start value

```
model Case1
  model M
    Real x(start = 1);
  equation
    der(x) = 1;
  end M;
  M m;
  Real y(start = 2) = m.x;
end Case1;
```

In Listing 1, tools adopting Modelica's recommended start value selection strategy will set the initial value of x to 2. However, older versions may initialize x with 1 instead. Considering that not all modeling tools support this feature uniformly, MWORKS.Sysplorer's will elevate start values to initial conditions by adding "fixed=true" attributes when performing model disambiguation.

2.2.1 Lack of initial equations for state variables

Listing 2. An example of model translation uncertainty caused by lack of initial equations for state variables

```
model Case2_1
  Real x;
  Real y;
equation
  x + y = 1;
  der(x) - der(y) = 0.5;
end Case2_1;
```

Noted that not all differential variables will be identified as state variables. In Listing 2, both x and y are differential variables, and they are subject to the algebraic constraint $x+y=1$. As a result, the system has only one degree of freedom, meaning only one of them can be selected as a state variable. The system does not provide additional information to determine which variable x or y is the better choice as the state variable.

Listing 3. A mathematically equivalent example to Listing 2

```
model Case2_2
  Real x;
  Real y;
equation
  y + x = 1;
  der(x) - der(y) = 0.5;
end Case2_2;
```

Changing $x+y=1$ to $y+x=1$ in Listing 2 yields Listing 3, which are mathematically equivalent, but get different result in Dymola 2024x trivial version. This is because Dymola selects y as the state variable in Listing 2, while choosing x as the state variable in Listing 3. As a result, the initial value $(x,y)=(1,0)$ in Listing 2, while $(x,y)=(0,1)$ in Listing 3. The root cause is that the original model lacks an initial condition, and the selection of state variables affects how default conditions are supplemented.

Listing 4. An alias example to Listing 2

```
model Case2_3
  Real x;
  Real u;
equation
  x + u = 1;
  der(x) - der(u) = 0.5;
end Case2_3;
```

Similarly, by renaming y to u in Listing 2, we obtain Listing 4. When simulating with MWORKS.Sysplorer, y is selected as the state variable in Listing 2, while x becomes the state variable in Listing 4. The default values of the state variables are used as the initial conditions, resulting in completely different outcomes.

To eliminate this uncertainty, modelers can: (1) Assign a start value to either x or y in Listing 2 to explicitly determine the state variables selection. However, if both are given start values, they return to an equally weighted condition, providing no additional information to prioritize one over the other. (2) Use the StateSelect attribute (see §4.8.7.1 (Modelica Association 2023)), such as StateSelect.prefer, to explicitly specify the preferred state variable. But if both variables are marked with this attribute, the ambiguity persists.

Since the fixed mechanism alone cannot reliably resolve such conflicts (particularly when multiple variables carry additional constraints), Sysplorer adopts the following strategy when performing model disambiguation: (1) Automatically applies "fixed=true" to elevate start values to initial conditions. (2) Uses "StateSelect.always" to enforce unambiguous state variables selection. Additional note: In cases of dynamic state variables selection, MWORKS.Sysplorer will only automatically process a subset of differential variables with "fixed=true" when performing model disambiguation.

2.2.2 Lack of initial equations for discrete variables

Listing 5. An example of model translation uncertainty caused by lack of initial equations for discrete variables

```
model Case3
  Integer x;
  Integer y;
equation
  when sample(0, 0.1) then
    x = pre(x) + 1;
    y = pre(y) + x;
  end when;
initial equation
  y = x + 1;
end Case3;
```


DOI
10.3384/ecp21839

In Listing 5, both x and y are discrete variables with only one initial equation provided, requiring an additional initial condition to be supplemented. The key issue lies in whether to set $x = 0$ or $y = 0$ as the supplemental condition. Although mathematically equivalent, when the initial equation $y = x + 1$ is reformulated as $x = y - 1$, both MWORKS.Sysplorer and Dymola2024x exhibit changes in their automatically supplemented conditions. This ultimately leads to inconsistent simulation results.

To eliminate this uncertainty, modelers can similarly assign start values to either x or y in Listing 5, which explicitly uses the start value as the initial condition. However, when both variables have start values specified, the system returns to an equally weighted condition where no additional information exists to determine which start value should take precedence as the initial condition. The fixed attribute mechanism alone cannot reliably resolve this conflict when both variables carry additional constraints. To address this, MWORKS.Sysplorer automatically applies "fixed=true" to elevate the start values to initial conditions.

2.3 Necessity of Start Values for Nonlinear System

The uncertainty in model solving arises from the existence of multiple solutions in nonlinear systems. Currently, the specifications do not provide clear recommendations for the selection of nonlinear iteration variables (i.e., nonlinear tearing), which may lead to inconsistencies in nonlinear tearing across different versions. Consequently, the initial iteration values may vary, resulting in different solutions or even cases where one version succeeds in finding a solution while another fails.

Listing 6. An example of necessity of start values for nonlinear system

```
model Case4_1
  Real x;
  Real y = x - 2;
equation
  (x - 0.78) * (x - 1.57) * (x - 3.14) = 0;
end Case4_1;
```

In Listing 6, the direct simulation yields a result of 0.78. However, when x's start value is set near 1.5, the simulation result becomes 1.57 and when x's start value is set near 3, the result changes to 3.14.

This demonstrates that the tool cannot autonomously determine which solution the modeler intends to obtain, and consequently performs iterations based on the provided start value. This observation underscores the critical necessity of specifying start values as initial estimates for nonlinear systems.

2.4 Nonlinear Tearing

Listing 7. An example of nonlinear iteration variables selection

```
model Case4_2
  Real x;
  Real y;
equation
  x - y = 2;
  (x - 0.78) * (x - 1.57) * (y - 1.14) = 0;
end Case4_2;
```

Listing 6 and Listing 7 are mathematically equivalent, but yield different simulation results in MWORKS.Sysplorer: x=0.78 in Listing 6 versus x=1.57 in Listing 7. This discrepancy occurs because x is explicitly determined as the nonlinear iteration variables in Listing 6 while both x and y qualify as potential nonlinear iteration variables, and MWORKS.Sysplorer defaults to selecting y In Listing 7.

Providing a start value for x forces its selection as the iteration variables. However, when both x and y have start values, the system reverts to the original ambiguous state. Nonlinear iteration variables has no *StateSelect* attributes like *state variables*. So MWORKS.Sysplorer innovatively employs annotation markup to explicitly specify which variable should serve as the nonlinear iteration variables. This annotation-based approach effectively eliminates the ambiguity, and the entire process can be automatically handled by the model disambiguation technology in MWORKS.Sysplorer.

2.5 More Words about Nonlinear Tearing

Note that, it is possible that some variables appear in both the nonlinear system and the initial nonlinear system. For instance, variable y in Listing 8 falls into this category

Listing 8. An example of nonlinear iteration variables selection

```
model Case5
  Real x;
  Real y;
  Real z;
equation
  der(x) = sin(time);
  x^2 + y^2 = z^2;
  z = x * sin(y) + y * cos(x);
initial equation
  x + y + z = 1;
end Case5;
```

2.6 More Words about State Variables Selection

Even when the initial equations have sufficient initial conditions, the selection of state variables can still introduce uncertainty. In Listing 9, if y is chosen as the state variable, then x and z form a nonlinear system. However, if x is selected as the state variable, no nonlinearity exists. This discrepancy leads to non-negligible differences in simulation results between platforms that choose x as the state variable and those that select y.

Listing 9. An example of model translation uncertainty caused by state variables selection

```
model Case6
  Real x(start = 0);
  Real y(start = 0.5);
```

```
  Real z;
equation
  der(x) + der(y) = 1;
  z = sin(x)^2 + 1;
  z = x + 2 * y + time;
initial equation
  x = 0;
end Case6;
```

In Listing 9, if the "stateSelect=StateSelect.always" attribute is added to x and y separately, and simulations are performed for each case, the results are as shown in Figure 1.

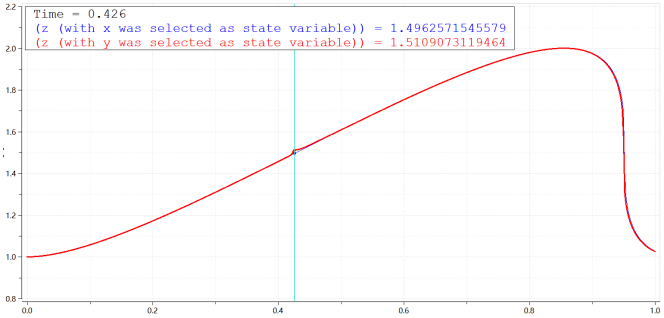

Figure 1. An example of inconsistency caused by State Variable Selection

2.7 A graphic example about State Variables Selection

We provide a simplified circuit model (which may lack physical significance) to demonstrate that even when using a drag-and-drop component assembly method, inconsistent selection of state variables can lead to divergent simulation results.

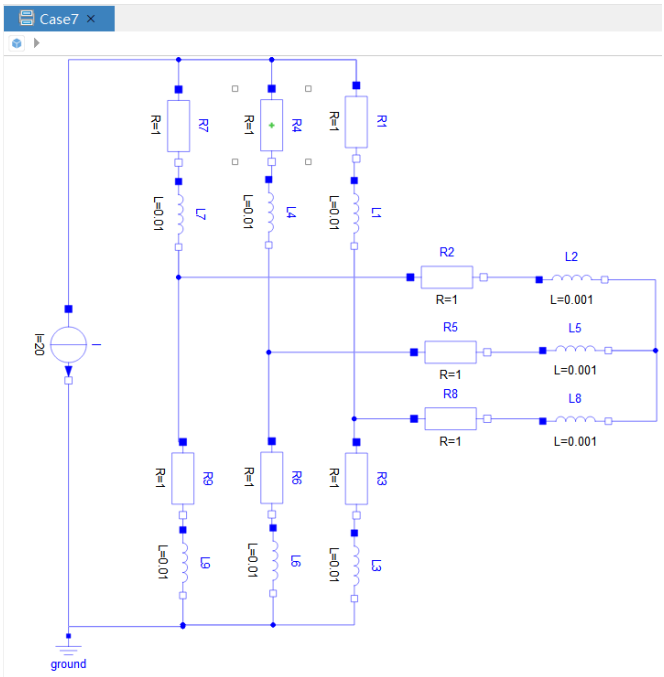

Figure 2. A graphic example about State Variables Selection

The circuit model Figure 2 contains 9 inductors with algebraic constraints, but only 4 of them can have their currents (which must be linearly independent) selected as state variables. MWORKS.Sysplorer selected L6.i, L7.i, L8.i, and L9.i as state variables, while OpenModelica chose L1.i, L4.i, L5.i, and L8.i. This difference in selection leads to inconsistent initial conditions between the two platforms, for instance, L1.i has an initial value of 20A in MWORKS.Sysplorer, but 0A in OpenModelica.

3 Model Disambiguation Technology

MWORKS.Sysplorer utilizes the results of the first simulation to perform the following actions during model disambiguation in the second translation:

1. Add the "fixed=true" attribute to variables that were supplemented with initial values in the report.

2. Add the "stateSelect=StateSelect.always" attribute to state variables listed in the report.

3. Record vendor-customized annotations for nonlinear iteration variables and initial nonlinear iteration variables, and assign them corresponding start values.

4. If the above modifications involve protected variables, vendor annotation could be used, otherwise, they would conflict with Chapter 4.1 of (Modelica Association 2023).

The following model Listing 10 represents the disambiguated version of all models mentioned in the previous section. Specifically, Case2_1_protected, Case4_2_protected, and Case5_protected correspond to modified versions of Case2_1, Case4_2, and Case5 respectively, where all model variables have been set as protected variables.

Listing 10. An example of disambiguated model

```
model DisambiguationModel
  Case1 m1(m(x(stateSelect=StateSelect.
      always,fixed=true)));
  Case2_1 m2(y(stateSelect=StateSelect.
      always,fixed=true));
  Case2_1_protected m2_p;
  Case3 m3(x(fixed=true));
  Case4_1 m4(x(start=0.78));
  Case4_2_protected m4_p;
  Case5 m5(x(stateSelect=StateSelect.always
      ),y(start=0.349110989362788),z(start
      =0.4190691332521359));
  Case5_protected m5_p;
  Case6 m6(y(stateSelect=StateSelect.always
      ),x(start=0));
  Case7 m7(L7(i(stateSelect=StateSelect.
      always,fixed=true)),L9(i(stateSelect=
      StateSelect.always,fixed=true)),L6(i(
      stateSelect=StateSelect.always,fixed=
      true)),L8(i(stateSelect=StateSelect.
      always,fixed=true)));
```


DOI
10.3384/ecp21839

```
  annotation(__MWORKS(Translation(m2_p(y(
      stateSelect=StateSelect.always,fixed=
      true)),m5_p(x(stateSelect=StateSelect
      .always),y(start=0.349110989362788),z
      (start=0.4190691332521359)),m4_p(y(
      start=-0.43000000000000005)),
      INonLinearVariables={m5.y, m5.z, m5_p
      .y, m5_p.z},MNonLinearVariables={m4.x
      , m4_p.y, m6.x, m5.y, m5_p.y})));
end DisambiguationModel;
```

The following are keynotes when doing model disambiguation:

1. If the original model is read-only, you can create a new model that references it, then perform disambiguation in the new model.

2. If the model fails to solve in the first place, disambiguating its state becomes rather meaningless, and the modelers need to modify the model themselves.

3. After Performing model disambiguation, modelers can still manually fine-tune the model by editing the text. Regardless of whether the original model is modifiable, it is recommended to create a new model for disambiguation to clearly track all modifications.

4. After Performing model disambiguation, providing explicit start values will accelerate the solving of the initial value system.

5. MWORKS.Sysplorer uses vendor-specific annotations to explicitly define the selection of nonlinear variables.

6. A variable may appear in both the nonlinear system and the initial nonlinear system, so MNonLinearVariables and INonLinearVariables are used to control them separately. MNonLinearVariables means nonlinear iteration variables during model simulation, while INonLinearVariables means nonlinear iteration variables during Initialization.

7. Unlike Modelon (Kari 2022), which considers both iteration variables and residual equations to resolve ambiguity caused by nonlinear tearing, we focus solely on iteration variables for the following reasons: configuring annotation is difficult when equations aren't at the simulation model's top level; different tools process iteration and residual equations maybe inconsistently; and our practical experience demonstrates that restricting iteration variables alone sufficiently eliminates model ambiguity.

8. If the modeler modifies the model after model disambiguation, causing variables in the annotations to become invalid, a warning message must be issued. There are two typical handling methods: directly ignoring the annotations or striving to satisfy the annotations. The tool may offer an option to let the modeler choose the preferred approach.

9. While other platforms don't recognize MWORKS.Sysplorer's vendor annotations, the model disambiguation provides deterministic initial values and relatively accurate compatible initial guesses for nonlinear iterations - making it highly probable to maintain result consistency across other platforms.

3.1 Annotation of Translation

Since the minimal tearing problem is NP-hard (Karp 2009) (Elmqvist and Otter 1994), most simulation platforms prioritize translation efficiency by employing greedy or heuristic algorithms to approximate minimal tearing. Even if vendors develop improved heuristic strategies, concerns over version compatibility often discourage modifications, hindering tool advancement.

Generally, better heuristics should facilitate nonlinear system solving. A typical scenario where models solvable in the original tool fail in new tools occurs because when initial modeling in the original tool, users would supplement appropriate start values as nonlinear iteration initial guesses if missing values caused solving failures, after switching to new tools with different tearing strategies, some models may lack required start values due to altered variable selection, necessitating manual re-supplementation.

Therefore, we propose that the Modelica specification incorporate Translation Annotations to govern nonlinear/-linear tearing and initial-value system tearing, thereby eliminating strategy-induced inconsistencies for modelers. For tool vendors, the implementation priority should be:

1. First satisfy the tearing requirements specified in the annotations.

2. Then prioritize variables based on start value hierarchy (top-level variables take precedence).

3. Finally pursue minimal tearing under equal conditions.

This strategy ensures tearing consistency across all compliant platforms when users provide comprehensive annotation specifications. The benefits are threefold:

1. Better interoperability between different vendors' tools.

2. Enhanced control for modelers over system solving.

3. Optimal utilization of each tool's translation and solving capabilities.

Ultimately, this advancement moves Modelica closer to the ideal of "Write once, run on any Modelica tool" - significantly improving model portability and reducing maintenance overhead. However, even with identical model

descriptions and unified nonlinear/linear tearing through translation annotations, achieving simulation results that differ only by rounding errors across platforms under identical settings remains a significant challenge. Key persistent discrepancies include:

1. Inconsistent precision tolerance interpretations across platforms
2. Divergent implementations of nonlinear/linear solvers
3. Algorithmic variations in implicit integration methods
4. Discrepancies in variable-step size adjustment strategies
5. Interpolation method variations
6. Event handling policy differences
7. Integrator restart strategy implementations

While writing this paper, we observed that both OpenModelica (Fritzson et al. 2022) and Modelon (Kari 2022) employ vendor-specific annotations for tearing, which exhibit subtle differences not only between themselves but also when compared with MWORKS.Sysplorer. This phenomenon demonstrates that such requirements are relatively common. Therefore, we hope the Modelica Association could provide an official annotation standard to eliminate inconsistencies across different implementations.

4 Experimental Results

When simulating the model "LoadTestExpRecovery" from the model library OpenIPSL-3.0.1 (Vanfretti et al. 2016), it was found that the results from MWORKS.Sysplorer and OpenModelica were completely inconsistent, for example, the variable "pwLine2.P12". After investigation, the inconsistency was traced to differences in nonlinear tearing. Since it was hard to control OpenModelica's nonlinear selection at the modeling level, we instead used custom annotation in MWORKS.Sysplorer to control nonlinear tearing, as shown in the Listing 11.

Listing 11. An example of control nonlinear tearing using Translation annotation

```
model LoadTestExpRecovery_Disambiguated
  extends LoadTestExpRecovery;
  annotation (experiment(Interval=0.03,
      StartTime=0,StopTime=15), __MWORKS(
      Translation(INonLinearVariables={
      order3_Inputs_Outputs1.e1q,
      order3_Inputs_Outputs1.vd,
      order3_Inputs_Outputs1.vq,pwLine2.n.
      ii,pwLine2.n.ir,pwLine2.n.vi,pwLine2.
      n.vr,pwLine2.p.ii,pwLine2.p.ir,
      pwLine3.n.ii,pwLine3.n.ir,pwLine3.n.
      vi,pwLine3.n.vr,pwLine3.p.ii,pwLine3.
      p.ir,pwLine4.n.ii,pwLine4.n.ir,
      pwLine4.p.ii,pwLine4.p.ir},
      MNonLinearVariables={
      order3_Inputs_Outputs1.vd,
      order3_Inputs_Outputs1.vq,pwLine2.n.
      ii,pwLine2.n.ir,pwLine2.n.vi,pwLine2.
      n.vr,pwLine2.p.ii,pwLine2.p.ir,
      pwLine3.n.ii,pwLine3.n.ir,pwLine3.n.
      vi,pwLine3.n.vr,pwLine3.p.ii,pwLine3.
      p.ir,pwLine4.n.ii,pwLine4.n.ir,
      pwLine4.p.ii,pwLine4.p.ir})));
end LoadTestExpRecovery_Disambiguated;
```

The final simulation results of Listing 11 are consistent with OpenModelica, see Figure 3 below.

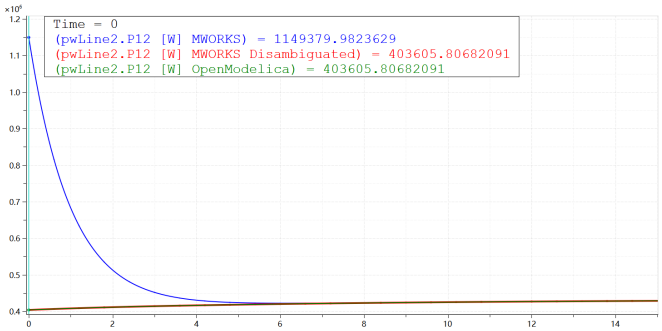

Figure 3. An example of inconsistency caused by State Variable Selection

References

Elmqvist, Hilding and Martin Otter (1994). "Methods for tearing systems of equations in object-oriented modeling". In: *Proceedings ESM*. Vol. 94, pp. 1–3.

Fritzson, Peter et al. (2022). "The OpenModelica integrated environment for modeling, simulation, and model-based development". In: Mic.

Kari, Oskar (2022). "Improving Tearing in a Modelica Compiler". In: *LU-CS-EX*.

Karp, Richard M (2009). "Reducibility among combinatorial problems". In: *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*. Springer, pp. 219–241.

Mattsson, Sven Erik and Gustaf Söderlind (1993). "Index reduction in differential-algebraic equations using dummy derivatives". In: *SIAM Journal on Scientific Computing* 14.3, pp. 677–692.

Modelica Association (2023-03). *Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification Version 3.6*. Tech. rep. Linköping: Modelica Association. URL: https://specification.modelica.org/maint/3.6/MLS.html.

Pantelides, Constantinos C. (1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231. DOI: 10.1137/0909014.

Vanfretti, Luigi et al. (2016). "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". In: *SoftwareX* 5, pp. 84–88.