Modeling, Identification and Control, Vol. 41, No. 4, 2020, pp. 241–285, ISSN 1890–1328

The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development

Peter Fritzson [1] Adrian Pop [1] Karim Abdelhak [2] Adeel Asghar [1]
Bernhard Bachmann [2] Willi Braun [2] Daniel Bouskela [9] Robert Braun [1]
Lena Buffoni [1] Francesco Casella [3] Rodrigo Castro [5] Rüdiger Franke [6]
Dag Fritzson [1] Mahder Gebremedhin [1] Andreas Heuermann [1] Bernt Lie [7]
Alachew Mengist [1] Lars Mikelsons [1] Kannan Moudgalya [4] Lennart Ochel [1]
Arunkumar Palanisamy [1] Vitalij Ruge [2] Wladimir Schamai [8] Martin Sjölund [1]
Bernhard Thiele [1] John Tinnerholm [1] Per Östlund [1]

[1] *PELAB – Programming Environment Lab, Dept. of Computer and Information Science Linköping University, SE-581 83 Linköping, Sweden. E-mail:* {peter.fritszon,adrian.pop}@liu.se

[2] *FH Bielefeld, Bielefeld, Germany*

[3] *Dip. Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy*

[4] *Dept. of Chemical Engineering, IIT Bombay, Mumbai, India*

[5] *Dept. Computer Science, Universidad de Buenos Aires, Argentina*

[6] *ABB AG, DE-68309 Mannheim, Germany*

[7] *University of South-Eastern Norway, Porsgrunn, Norway*

[8] *Danfoss Power Solutions GmbH & Co. OHG, Offenbach, Germany*

[9] *Electricité de France, (EDF Lab), Chatou, France*

Abstract

OpenModelica is a unique large-scale integrated open-source Modelica- and FMI-based modeling, simulation, optimization, model-based analysis and development environment. Moreover, the OpenModelica environment provides a number of facilities such as debugging; optimization; visualization and 3D animation; web-based model editing and simulation; scripting from Modelica, Python, Julia, and Matlab; efficient simulation and co-simulation of FMI-based models; compilation for embedded systems; Modelica-UML integration; requirement verification; and generation of parallel code for multi-core architectures. The environment is based on the equation-based object-oriented Modelica language and currently uses the MetaModelica extended version of Modelica for its model compiler implementation. This overview paper gives an up-to-date description of the capabilities of the system, short overviews of used open source symbolic and numeric algorithms with pointers to published literature, tool integration aspects, some lessons learned, and the main vision behind its development.

Keywords: Modelica, OpenModelica, MetaModelica, FMI, modeling, simulation, optimization, development, environment, numeric, symbolic, compilation, embedded system, real-time

doi:10.4173/mic.2020.4.1

1 Introduction

The OpenModelica environment was the first open source Modelica environment supporting equation-based object-oriented modeling and simulation using the Modelica modeling language (Fritzson and Engelson, 1998; Modelica Association, 2017; Fritzson, 2014). Its development started in 1997 resulting in the release of a flattening frontend for a core subset of Modelica 1.0 in 1998 (Fritzson and Kågedal, 1998). After a pause of four years, the open source development resumed in 2002. A very early version of OpenModelica is described in (Fritzson et al., 2005). Since then the capabilities of OpenModelica have expanded enormously. The Open Source Modelica Consortium which supports the long-term development of OpenModelica was created in 2007, initially with seven founding organizations. The scope and intensity of the open source development has gradually increased. At the time of this writing the consortium has more than fifty supporting organizational members. The long-term vision for OpenModelica is an integrated and modular modeling, simulation, model-based development environment with additional capabilities such as optimization, sensitivity analysis, requirement verification, etc., which are described in the rest of this paper. Fritzson et al. (2005, 2018c) are two less detailed and now partly out of date overview papers about OpenModelica.

The current overview paper gives an up-to-date greatly expanded description of the capabilities of the system, short overviews of used open source symbolic and numeric algorithms with pointers to published scientific literature, tool integration aspects, some lessons learned, and the main vision behind its development.

This paper is organized as follows. Section 2 presents the idea of integrated environment, Section 3 details the goals for OpenModelica, Section 4.1 presents a detailed overview of the OpenModelica environment, Section 5 describes selected open source applications, Section 6 presents related work, and Section 7 the conclusions.

2 Integrated Interactive Modeling and Simulation Environments

An integrated interactive modeling and simulation environment is a special case of programming environments with applications in modeling and simulation. Thus, it should fulfill the requirements both from general integrated interactive environments and from the application area of modeling and simulation mentioned in the previous section.

The main idea of an integrated programming environment in general is that a number of programming support functions should be available within the same tool in a well-integrated way. This means that the functions should operate on the same data and program representations, exchange information when necessary, resulting in an environment that is both powerful and easy to use. An environment is interactive and incremental if it gives quick feedback, e.g., without recomputing everything from scratch, and maintains a dialogue with the user, including preserving the state of previous interactions with the user. Interactive environments are typically both more productive and more fun to use than non-interactive ones.

There are many things that one wants a programming environment to do for the programmer or modeler, particularly if it is interactive. Comprehensive software development environments are expected to provide support for the major development phases, such as:

- Requirements analysis
- Design
- Implementation
- Maintenance

A pure programming environment can be somewhat more restrictive and need not necessarily support early phases such as requirements analysis, but it is an advantage if such facilities are also included. The main point is to provide as much computer support as possible for different aspects of systems development, to free the developer from mundane tasks so that more time and effort can be spent on the essential issues.

Our vision for an integrated interactive modeling and simulation environment is to fulfill essentially all the requirements for general integrated interactive environments combined with the specific needs for modeling and simulation environments, e.g.:

- Specification of requirements, expressed as documentation and/or mathematics
- Design of the mathematical model
- Symbolic transformations of the mathematical model
- A uniform general language for model design, mathematics, and transformations
- Automatic generation of efficient simulation code
- Execution of simulations
- Debugging of models
- Design optimization and parameter studies

- Export/import of models to/from other tools
- Evaluation and documentation of numerical experiments
- Graphical presentation
- Model and system structure parameterization
- Variant and version handling, traceability

3 Goals for OpenModelica

- Providing a complete open source Modelica-based industrial-strength implementation of the Modelica language, including modeling and simulation of equation-based models, system optimization, and additional facilities in the programming/modeling environment.
- Providing an interactive computational environment for the Modelica language. It turns out that with support of appropriate tools and libraries, Modelica is very well suited as a computational language for development and execution of numerical algorithms, e.g. for control system design and for solving nonlinear equation systems.

The research related goals and issues of the OpenModelica open source implementation of a Modelica environment include, but are not limited to, the following:

- Development of a complete formal specification and reference implementation of Modelica, including both static and dynamic semantics. Such a specification can be used to assist current and future Modelica implementers by providing a semantic reference, as a kind of reference implementation.
- Language design, e.g. to further extend the scope of the language, e.g. for use in diagnosis, structural analysis, system identification, integrated product development with requirement verification, etc., as well as modeling problems that require partial differential equations.
- Language design to improve abstract properties such as expressiveness, orthogonality, declarativity, reuse, configurability, architectural properties, etc.
- Improved implementation techniques, e.g. to enhance the performance of compiled Modelica code by generating code for parallel hardware.
- Improved debugging support for equation-based languages such as Modelica, to make them even easier to use.
- Improved optimization support, with integrated optimization and modeling/simulation. Two kinds: parameter-sweep optimization based on multiple simulations; direct dynamic optimization of a goal function without lots of simulations, e.g., using collocation or multiple shooting.
- Easy-to-use specialized high-level (graphical) user interfaces for certain application domains.
- Visualization and animation techniques for interpretation and presentation of results.
- Integrated requirement modeling and verification support. This includes the ability to enter requirements formalized in a kind of Modelica style, and to verify that the requirements are fulfilled for selected models under certain usage scenarios.
- High-performance simulation, e.g., of large-scale models, generating simulations to efficiently utilize multi-core computers for high performance.

3.1 History and System Architecture

The OpenModelica effort started by developing a rather complete formal specification of the Modelica language. This specification was developed in Operational Semantics, which still is the most popular and widely used semantics specification formalism in the programming language community. It was initially used as input for automatic generation of the Modelica translator implementations which are part of the OpenModelica environment. The RML compiler generation tool (our implementation of Operational Semantics) (Fritzson et al., 2009a) was used for this task.

However, inspired by our vision of integrated interactive environments with self-specification of programs and data, and integrated modeling and simulation environments), in 2005 we designed and implemented an extension to Modelica called MetaModelica (Pop et al., 2006; Fritzson et al., 2011, 2019), see also Section 4.1.3. This was done in order to support language modeling and specification (including modeling the language itself), in addition to the usual physical systems modeling applications, as well as applications requiring combined symbolic-numeric capabilities. Modeling the semantics in Modelica itself was also inspired by functional languages such as Standard ML (Milner et al., 1997), and OCaml (OCaml, 2018). Moreover, it was an investment into a future Modelica becoming a combined symbolic-numeric language such as Mathematica, but more efficient and statically strongly typed.

This language extension has a backwards compatible Modelica-style syntax but was initially implemented on top of the RML compiler kernel. The declarative specification language primitives in RML with single-assignment pattern equations, potentially recursive uniontypes of records and match expressions, fit well into Modelica since it is a declarative equation-based language. In 2006 our whole formal specification of Modelica static and translational semantics, at that time about 50 000 lines, was automatically translated into MetaModelica. After that, all further development of the symbolic processing parts of the OpenModelica compiler (the run-time parts were mainly written in C), was done in MetaModelica.

At the same time we embarked on an effort to completely integrate the MetaModelica language extension into the Modelica language and the OpenModelica compiler. This would enable us to support both Modelica and MetaModelica by the same compiler. This would allow modeling the Modelica tool and the OpenModelica compiler using its own language. This would get rid of the limitations of the RML compiler kernel and the need to support two compilers. Moreover, additional tools such as our Modelica debugger can be based on a single compiler.

Such an ability of a compiler to compile itself is called compiler bootstrapping. This development turned out to be more difficult and time-consuming than initially expected; moreover, developers were not available for a few years due resource limitations and other priorities. Finally, bootstrapping of the whole OpenModelica compiler was achieved in 2011. Two years later, in 2013, all our OpenModelica compiler development was shifted to the new bootstrapped compiler (Sjölund et al., 2014; Sjölund, 2015), after automatic memory reclamation (garbage collection), separate compilation, and a new efficient debugger had been achieved for our new compiler platform.

4 The OpenModelica Environment

At the time of this writing, the OpenModelica environment primarily consists of the following functionalities and subsystems:

- OMC – The OpenModelica Model Compiler
- The new OpenModelica Compiler frontend
- Symbolic Programming and Meta Modeling with MetaModelica
- Numeric-symbolic solver modules
- OMEdit – the OpenModelica Graphic Model Editor and Simulator GUI
- 3D Animation and Visualization
- Debugging and Performance Optimization
- Interactive Electronic Notebooks
- Interactive Scripting using Modelica, Python, Julia, and Matlab
- Audio-Video Tutorials
- FMI – Functional Mockup Interface
- Multi-Parameter Sensitivity Analysis
- Parameter System Identification
- Embedded System Support
- Model-based Control Design with Dynamic Optimization
- Model-based Fault and Dependability Analysis
- Data Reconciliation for Enhanced Sensor Data
- Using Artificial Neural Networks for Model Calibration
- Embedded System Support
- MDT Eclipse Plug-in
- ModelicaML UML Profile and Eclipse Plug-in
- Requirement Verification
- Design Optimization
- Parallelization and Multi-Core

The relationships between the main OpenModelica subsystems are briefly depicted above in Figure 1. Their functionality and selected applications are described in the rest of this article. An example of using OpenModelica to perform simulations and plotting simulation results is depicted in Figure 2 for the V6Engine model.

4.1 OMC – The OpenModelica Model Compiler

OMC is the OpenModelica compiler which translates Modelica models into simulation code, which is compiled and executed to perform simulations. The OpenModelica compiler is generated from formal specifications in RML (earlier) or MetaModelica (currently). At the time of this writing the OpenModelica compiler (OMC) is generated from a specification of about three hundred thousand lines of MetaModelica. Moreover, OMC is able to compile itself, i.e., it is bootstrapped (Sjölund et al., 2014). There is also a compilation mode to generate low-footprint code for embedded systems (Section 4.20).

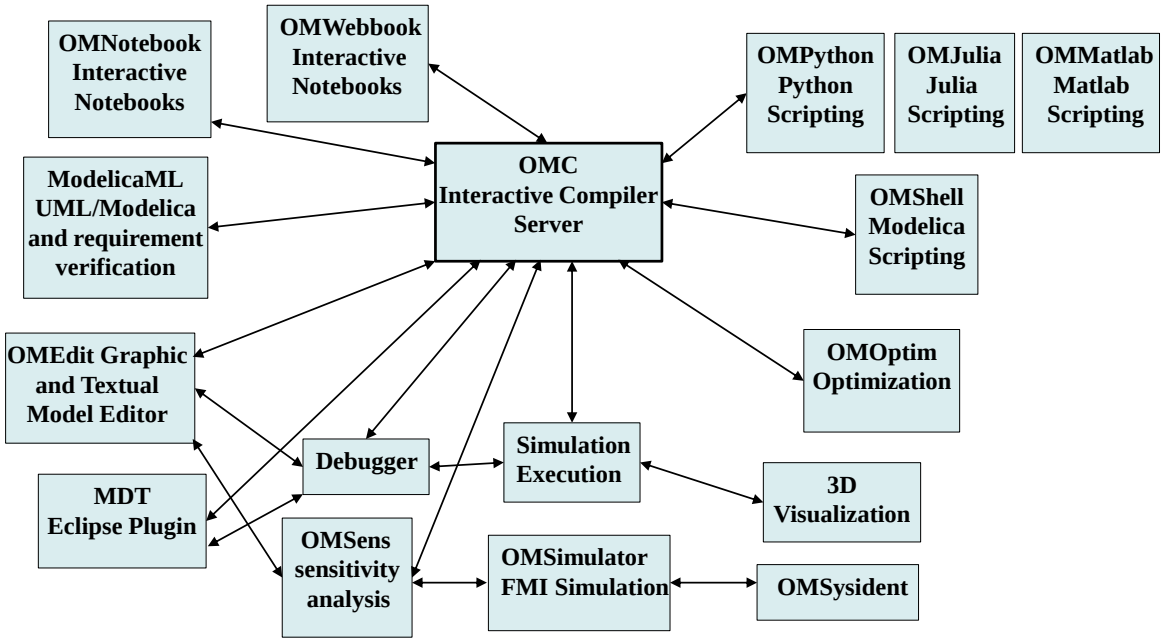


Figure 1: The architecture of the OpenModelica environment. Arrows denote data and control flow.

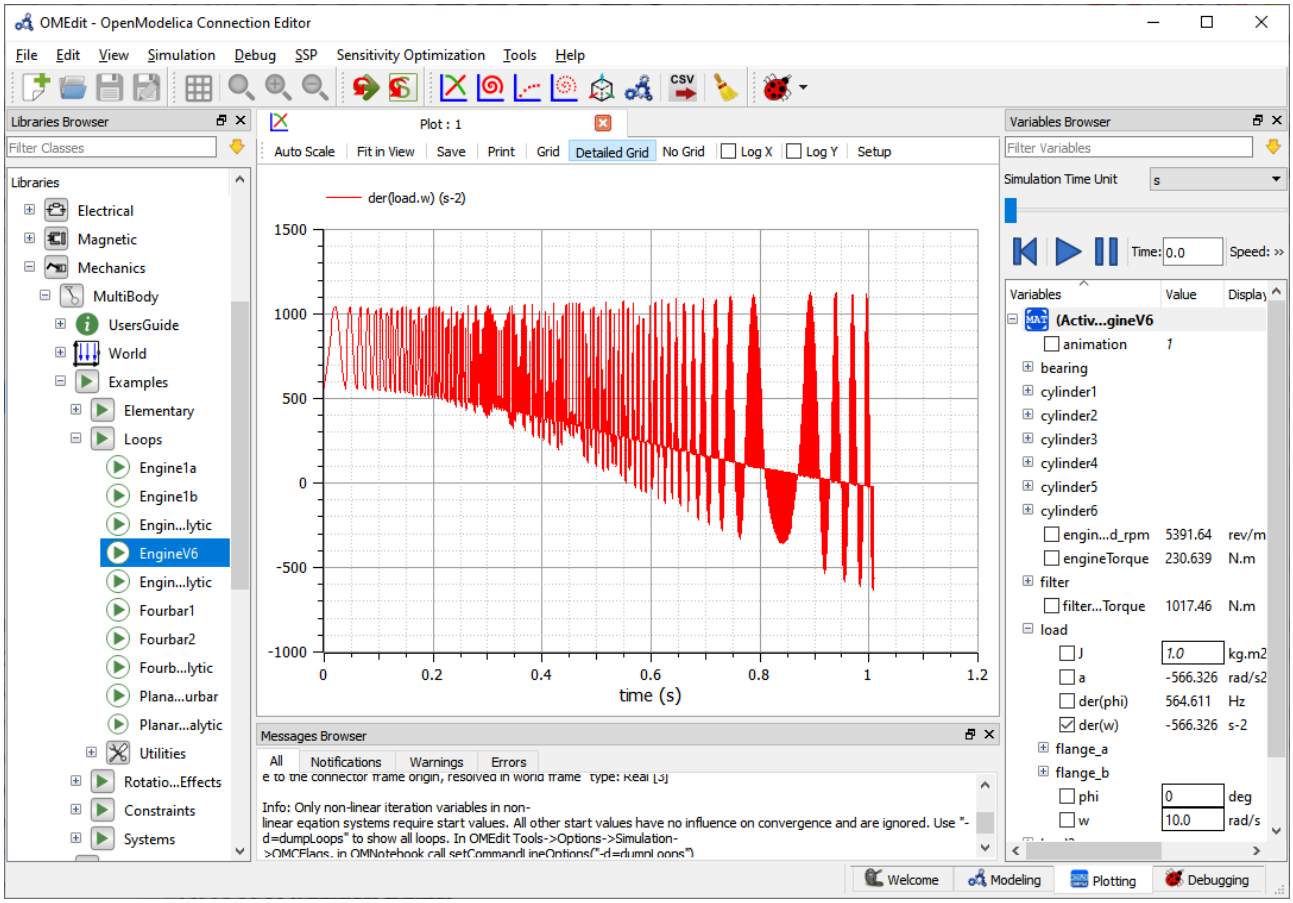


Figure 2: OpenModelica simulation of the V6Engine model with 11000 equations. Plotting simulation results using OMEdit. *Left*: Model browser. *Right*: Plot variable browser. Bottom: message browser window.

1. **Lexical Analysis**
   Keywords, operators and identifiers are extracted from the model.
2. **Parsing**
   An abstract syntax tree represented in MetaModelica is created from the operators and identifiers.
3. **Semantic Analysis**
   The abstract syntax tree gets tested for semantic errors.
4. **Intermediate Representation**
   Translation of the abstract syntax tree to an intermediate representation (IR) called SCode in MetaModelica. This is further processed by the frontend (Section 4.1.2) producing DAE IR code.
5. **Symbolic Optimization Backend**
   The intermediate representation gets optimized and preprocessed (Section 4.2).
6. **Code Generation**
   Executable code gets generated from the low level intermediate representation.

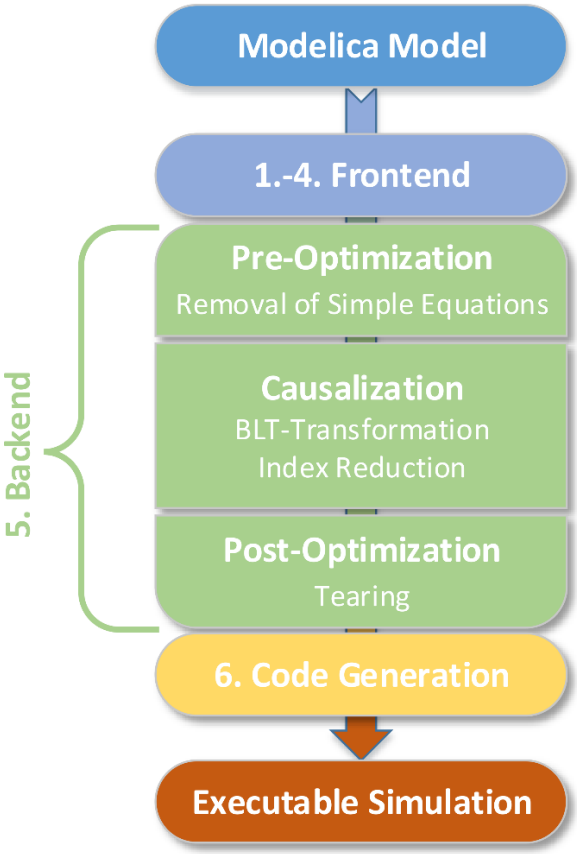


Figure 3: OpenModelica compiler workflow – from model to executable simulation code.

4.1.1 OMC Compiler Structure

The compilation of Modelica models with the OpenModelica Compiler (OMC) can be divided into six phases (Figure 3) to get an executable simulation. In a nutshell the Frontend performs lexical and semantic analysis and the Backend performs symbolic optimization on the provided DAE-model-representation. From the optimized MetaModelica intermediate representation an executable simulation program in a target language (C, C++ and some others) is generated and compiled.

4.1.2 New Compiler Frontend

As previously mentioned in Section 3.1, a new OpenModelica compiler frontend has been developed. This large effort has been made in order to provide complete language coverage as well as much faster compilation including efficient support for compilation of very large models. The first usable version was released in OpenModelica 1.14.0, in December 2019. The new frontend (Pop et al., 2019) uses model-centric and multiple phases design principles and is about 10 to 100 times faster than the old frontend. A few highlights:

- The new front-end was carefully designed with performance and scalability in mind.
- References (pointers) are used to link component references to their definition scope via lookup and usage scope via application.
- Constant evaluation and expression simplification are more restricted compared to the old frontend.
- Arrays of basic types and arrays of models are not expanded until the scalarization phase.
- Expansion of arrays is currently needed because the backend currently cannot handle all the cases of non-expanded arrays, but will be eliminated in the future (Section 4.2.8) to give increased performance for array computations.

One of the design principles of the new frontend has been to find ways to break dependencies between the various frontend phases. Instead of being component-focused like the old compiler frontend it has been designed to be model-focused, meaning that each frontend phase processes the whole model before the model is passed on to the next phase. The result is the design

seen in Figure 4, which shows the flow of the model through the different phases of the new frontend.

The symbolic instantiation phase builds the instance tree and constructs all the nodes, and the expression instantiation phase instantiates all expressions in that instance tree. This involves looking up the names used in expressions and associating them with the correct nodes in the instance tree. The lookup tree for a class is only constructed once and then reused for all instances of that particular class, unlike the old frontend where a new lookup tree is constructed for each instance.

The typing phase traverses the instance tree and determines the type of all variables and expressions. The flattening phase of the new frontend traverses the instance tree and flattens the tree into a flat model that consists of a list of variables, a list of equations, and a list of algorithms. It also expands connect-equations and for-equations into basic equations.

The new frontend is implemented in modern MetaModelica 3.0 which combines Modelica features with functional languages features. The implementation currently consists of 65 MetaModelica packages or uniontypes defining encapsulated data structures and functions that operate on the defined data.

4.1.3 MetaModelica for Symbolic Programming and Meta-Programming

The need for integrating system modeling with advanced tool capabilities is becoming increasingly pronounced. For example, a set of simulation experiments may give rise to new data that is used to systematically construct a series of new models, e.g. for further simulation and design optimization.

Such combined symbolic-numeric capabilities have been pioneered by dynamically typed interpreted languages such as Lisp (Teitelman, 1974) and Mathematica (Wolfram, 2003). Such capabilities are also relevant for advanced modeling and simulation applications but lacking in the standard Modelica language. Therefore, this is a topic of long-running design discussions in the Modelica Design group.

One contribution in this direction is the MetaModelica language extension (Pop and Fritzson, 2006; Fritzson et al., 2011, 2019) that has been developed to extend Modelica with symbolic operations and advanced data structures in a backwards-compatible way, while preserving safe engineering practices through static type checking and a compilation-based efficient implementation.

The MetaModelica language is an efficiently compiled language that provides symbolic programming using tree and list data structures. This is similar to what is provided by the rather young language Julia (Bezanson et al., 2017; Julialang, 2018) which has recently appeared, Julia 1.0 was released in August 2018. A comparison between MetaModelica and Julia is presented by Fritzson et al. (2019). MetaModelica is also used for modeling/specification of languages (including the Modelica language) and for Modelica-style programming of model transformations, where the OpenModelica compiler itself is the currently largest application.

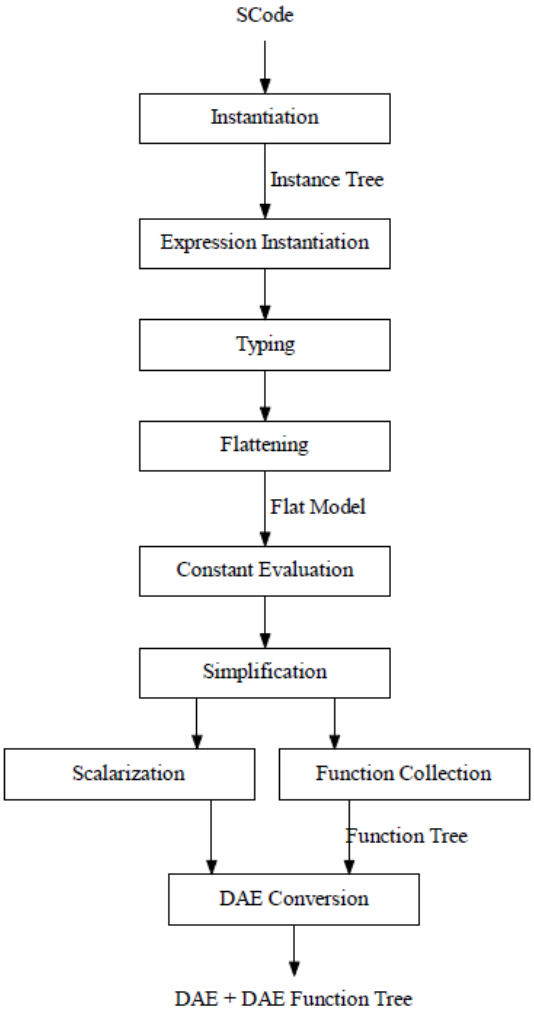


Figure 4: The OMC new frontend phases.

The research contributions of MetaModelica are not about inventing new language constructs since they have already been well proven in several other languages. However, in the context of Modelica there are contributions on integrating such constructs into the Modelica language including the Modelica type system in a backwards compatible way. The following is a very brief overview of the most important language extensions:

- Overloading of user-defined operators and functions. Note: overloading is called multiple dis-

patch in Julia.

- Uniontype construct to define unions of possibly recursive record types. This is used to create tree data structures. Syntax example:

```
uniontype Exp
  record RCONST
    Real rval;
  end RCONST;
  record INTconst
    Integer exp1;
  end INTconst;
end Exp;
```

  Uniontypes are also present in Julia.

- The predefined type `Any` is a supertype of any other MetaModelica type, i.e., all other MetaModelica types are subtypes of `Any`. Used e.g. in replaceable,

```
replaceable type TypeParam =
    Any constrainedby Any;
```

- The predefined `Option` uniontype provides a type safe way of representing optional values.

- Built-in list and tuple data types. List of integers: `list(2,3,4)` is a list of integers. A tuple: `(a,b,"cc")`

- Match-expressions for traversing and transforming complex data structures. This supports pattern matching specified by pattern expressions, and building data structures such as trees, lists, etc.

- Exception handling using

```
try
  // ...
else
  // ...
end try;
```

  Also a `fail()` function to cause an exception.

The following recent enhancements available in MetaModelica 3.0 were found to be quite useful in the implementation of the new frontend:

- Flexible pattern matching specified by `()`, that does not require verbose listing of all record fields (or named field access) of the record in the pattern matching, e.g., `UNTYPED_BINDING()`.

- Record field access via dot notation inside the case, e.g., `binding.bindingExp`.

- Definition of functions inside uniontypes.

- Definition and usage of parameterized union datatypes such as trees using redeclare/replaceable types.

4.1.4 Experimental Just-in-Time Compilation

Just-in-Time Compilation (JIT) allows compilation and executing code during runtime. Such a facility opens up new flexible strategies for handling the compilation and execution of Modelica code and even going beyond Modelica to general variable structure systems. The following work is currently ongoing related to OpenModelica.

The OpenModelica LLVM backend (OMLB)
The OpenModelica LLVM backend (OMLB) is an experimental OpenModelica prototype backend to investigate just-in-time compilation using the LLVM compiler framework (Tinnerholm, 2019). The goal was to investigate the advantages and disadvantages of having OMC target LLVM instead of C. The investigation was also performed to examine if targeting LLVM would be a viable option to achieve efficient Just-in-time compilation (JIT). Another approach with similar goals was conducted by (Agosta et al., 2019). While OMLB currently is not complete enough for bootstrapping, it demonstrates the benefits of having an LLVM based backend and JIT. OMLB is presently able to compile the algorithmic subsets of MetaModelica and Modelica interactively. Inspired by the design goals of the Julia programming language and the successful use of Julia for equation-based modeling as done by Elmqvist et al. (2017), an investigation was conducted in 2018 comparing MetaModelica and Julia (Fritzson et al., 2019).

This investigation highlighted the similarities and differences between the two languages, both in terms of design goals and programming paradigm. The conclusions were that there are similarities both with regards to the indented audience and the design goals of the two. These similarities prompted another investigation (Tinnerholm et al., 2019) regarding the possibility of automatically translating the existing OpenModelica frontend into Julia. Such an OpenModelica frontend in Julia could provide a framework for experimentation with variable structured systems while at the same time adhering to the Modelica standard.

An Experimental Julia-based Modelica Compiler Prototype
To empirically investigate the advantages, disadvantages, and challenges of providing a Modelica compiler in Julia, an OpenModelica to Julia translator

was developed together with an extension of the MetaModelica runtime initially described in (Fritzson et al., 2019). From our preliminary experiments in (Tinnerholm et al., 2019) we observed that automatically generated Julia code may outperform hand-written MetaModelica code in some cases. However, the compilation time was overall slower compared to OMLB, due to OMLB making use of precompiled runtime functions in contrast with the overhead imposed by the Julia compiler due to type specialization.

Prototyping a Standards Compliant Modelica Compiler with Run-time Just-in-Time Compilation
Regarding just-in-time compilation (JIT), the status in the fall of 2019 was that there are still two options to provide a JIT in the OpenModelica compiler environment. One is via OMCompiler.jl – an experimental standards compliant prototype subset Modelica compiler in Julia, the other is to increase the scope of OMLB with its associated JIT. However, since the MetaModelica to Julia translator is capable of translating the existing OMC frontend, it is also capable of converting the OMLB code-generator into Julia. Thus, further development of OMCompiler.jl will not invalidate the possibility of having LLVM as a final backend target for OMC.

4.1.5 Template-Based Code Generation

The OMC code generation uses a text-template based approach. The Susan text template language (Fritzson et al., 2009b) based on MetaModelica was developed for this purpose. It facilitates generation of code for multiple target platforms from the low-level intermediate code in and enables writing concise code generation specifications. Several alternative regular code generators are available to produce the simulation code as C or C++ code (or Java or C# code using experimental code generators), which is compiled and executed to perform simulations or to export FMUs.

4.2 OMC Backend with Numeric-Symbolic Solver Modules

In the following we briefly present four of the most important numeric-symbolic modules inside the OMC Backend that perform symbolic optimization (Figure 3).

4.2.1 Removal of Simple Equations

Some variables in the equation system correlate, because they are connected by so-called simple equations. The most elementary equation is equality, e.g.: $x = y$. In this equation it is possible to declare either x or y as an alias variable and replace it in every equation it occurs with the corresponding other variable. The equation can be removed from the system and is later used to reconstruct the value of the removed alias variable if necessary. Even more complex, but still simple equations can be extracted such that the resulting system will be much smaller (e.g. any linear equation connecting two variables). More information for this process regarding a specific model can be gained using the compiler flag `-d=debugAlias`.

4.2.2 BLT-Transformation (Matching/Sorting)

The transformation of a system of differential-algebraic equations to Block-Lower-Triangular form is fundamental to the simulation. The first step is to assign every variable to an equation such that the equation can be solved (explicitly or implicitly) for the assigned variable. This step is called Matching and is unique if there are no algebraic loops in the system. Afterwards the equations are sorted into blocks, such that an evaluation sequence is achieved (Sorting). If a block contains more than one equation, it forms an algebraic loop, where all variables assigned to those equations have to be solved simultaneously. Further information on BLT-Transformation can be found in Duff et al. (2017, chapter 6). More information regarding a specific model can be gained using the compiler flag `-d=backenddaeinfo`.

4.2.3 Index Reduction

The *differential index* of a system of differential-algebraic equations is defined as the maximum number of differentiations of all equations such that all unknowns of the system can be solved by integrating an ordinary differential equation. Most solvers are designed to work with systems of index zero or one, so an efficient reduction is necessary. The equations that have to be differentiated and the corresponding number of differentiations can be obtained with Pantelides (1988) algorithm. The index reduction algorithm with dummy-states, described in Söderlind and Mattsson (1993), reduces the system to index one, so that it can be simulated with common solvers. Alternative methods to handle index reduction have been proposed in Qin et al. (2016, 2018). Simulation without index reduction is also possible, but less reliable. The process of index reduction identifies a set of state variables which are algebraically connected. Some of those states will be treated as regular algebraic variables (*dummy states*) to simulate the system correctly. One can influence this process of state selection by providing `stateSelect` attributes for states, e.g., `x(stateSelect=StateSelect.avoid)`, see Table 1. More information for this process regard-

Table 1: StateSelect Attributes

Attribute	Description	Strictness
always	Always pick as continuous state (never pick as dummy state)	Forced
prefer	Prefer to pick as continuous state	Suggestion
default	Default value, no special treatment	No Influence
avoid	Try to avoid picking this as a continuous state	Suggestion
never	Never pick as continuous state (always pick as dummy state)	Mostly Forced

Table 2: TearingSelect Annotation

Attribute	Description	Strictness
always	Always pick as tearing variable	Mostly Forced
prefer	Prefer to pick as tearing variable	Suggestion
default	Default value, no special treatment	No Influence
avoid	Try to avoid picking this as a tearing variable	Suggestion
never	Never pick as tearing variable	Forced

ing a specific model can be gained using the compiler flags `{d=bltdump` or `{d=stateselection` (extends `{d=backenddaeinfo`).

4.2.4 Tearing

For every algebraic loop some of the assigned variables are chosen as tearing-variables, such that all other variables can be evaluated explicitly on the basis of those variables. The goal is to efficiently find small sets of tearing-variables. Many algorithms are already implemented in the OpenModelica Compiler and published in Cellier and Kofman (2006). One can influence this process by providing `tearingSelect` annotations, similar to the `stateSelect` attribute. Since this is not part of the Modelica language and specific to OpenModelica, it must be provided as an annotation (e.g. `x annotation(tearingSelect = prefer)`; see Table 2. Discrete variables can never be tearing variables. More information for this process regarding a specific model can be gained using the compiler flags `-d=dumpLoops` or `-d=iterationVars`.

4.2.5 Simulation using Numeric Solvers

After code generation for specified target language and linking with the OpenModelica Simulation Runtime, the model can be simulated. For the simulation OpenModelica offers multiple numeric integration/solver methods for ODE systems as well as DAE-mode (Section 4.2.6) for direct solution of DAE systems. Mostly DASSL (Petzold, 1982) respectively IDA (Hindmarsh et al., 2005) are used to integrate the systems, but there are more solvers for specific problems (Table 3). For models containing algebraic loops there are multiple linear (Table 4) and non-linear (Table 5) algebraic solvers to choose from. There are general purpose solvers like LAPACK for linear problems and a combination of a Newton method with the total pivot method as fallback.

4.2.6 DAEMode

A recent extension of the numeric solver module is the DAEMode which is used for solving very large models. DAE-mode can be accessed using the compiler flag `{daeMode`. This is part of an emerging trend in Modelica tools of handling large-scale models, with hundreds of thousands or possibly millions of equations, (Casella, 2015). OpenModelica has pioneered this field by introducing sparse solvers in the solution chain: KLU for linear algebraic equations, Kinsol for nonlinear algebraic equations, and IDA for causalized differential equations. It also introduced the direct use of IDA as differential-algebraic equation solver, skipping the traditional causalization step, which is computationally more efficient for certain classes of systems. The largest system handled so far is an electro-mechanical power system model with about 600 000 differential-algebraic equations (Braun et al., 2017).

4.2.7 Homotopy-based Initialization

In many cases, solving the initialization problem of Modelica models requires solving nonlinear system by means of iterative methods, whose convergence may be critical if the provided initial guesses are not close enough to the solution. To mitigate this problem,

Table 3: Available numeric solver methods

Integrator	Method	Explicit or Implicit	Step Size	Order
euler	Forward Euler method	Explicit	Fixed	1
impeuler	Backward Euler method	Implicit	Fixed	1
irksco	Own developed Runge-Kutta solver	Implicit	Variable	1-2
heun	Heun's method	Explicit	Fixed	2
trapezoid	Trapezoid rule	Implicit	Fixed	2
rungekutta	Classic Runge-Kutta method	Explicit	Fixed	4
imprungekutta	Runge-Kutta methods based on Radau and Lobatto IIA-method	Implicit	Variable	1-6
rungekuttaSsc	Runge-Kutta based on Novikov	Explicit	Variable	4-5
Dassl (default)	BDF method	Implicit	Variable	1-5
ida	BDF method with sparse linear solver	Implicit	Variable	1-5
symSolver	Symbolic inline solver	-	Fixed	1
symSolverSsc	Symbolic implicit Euler	-	Variable	1
qss	Quantized state systems method (Migoni et al., 2011)	Implicit	Variable	1
dassl + daeMode	Solves the DAE system instead of ODE system	Implicit	Variable	1-5
ida + daeMode	Solves the DAE system instead of ODE system	Implicit	Variable	1-5
optimization	Special solver for dynamic optimization	-	-	-

Table 4: Available linear solvers for algebraic loops

Solver	Method	
default	Lapack with totalpivot as fallback	(Anderson et al., 1999)
lapack	Non-Sparse LU factorization using LAPACK	(Anderson et al., 1999)
lis	Iterative linear solver	(Nishida, 2010)
klu	Sparse LU factorization	(Natarajan, 2005)
umfpack	Sparse unsymmetric multifrontal LU factorization	(Davis, 2004)
totalpivot	Total pivoting LU factorization for underdetermined systems	

Table 5: Available non-linear solvers for algebraic loops

Solver	Method	
hybrid	Modified Powell hybrid method from MINPACK	(Dennis Jr. and Schnabel, 1996)
kinsol	Combination of Newton-Krylov, Picard and fixed-point solver	(Taylor and Hindmarsh, 1998)
newton	Newton-Raphson method	(Cellier and Kofman, 2006)
mixed	Homotopy with hybrid as fallback	(Keller, 1978; Bachmann et al., 2015)
homotopy	Damped Newton solver with fixed-point solver and Newton homotopy solver as fallbacks	

OpenModelica implements the `homotopy()` operator of the language, which allows to replace some key expressions in model equations with simplified counterparts, to make the initialization problem less sensitive to an accurate choice of initial guesses. Once the solution of the simplified problem has been found, a homotopy transformation is performed from the simplified to the actual formulation of the expression in the homotopy operators. If the simplified expression is chosen appropriately, the homotopy path followed by the solution is continuous and allows to reliably reach the solution of the actual initialization problem (Sielemann et al., 2011; Bachmann et al., 2015; Keller, 1978). See also Casella et al. (2011b) for an application.

4.2.8 New OMC Backend

The current OMC backend is lacking in modularity, efficiency, and does not support non-expanded arrays in a general way. The latter functionality is needed to support compilation and simulation of large-scale models with large arrays. Therefore an effort has been started spring of 2020 of re-designing and re-implementing the backend to improve modularization and enable efficient handling of general non-expanded arrays.

4.3 OMEdit – the OpenModelica Graphic Model Editor and Simulator GUI

OMedit is the OpenModelica graphical model editor (Asghar et al., 2011) for component-based model design by connecting instances of Modelica classes. The editor also provides text editing. Moreover, the OMEdit GUI provides a graphical user interface to simulation and plotting (Figure 2). Also, it also provides browsing, parameter update, 3D animation (Section 4.4), debugging and performance analysis (Section 4.5), and FMI composite editing (Section 4.10).

Figure 5 depicts the connection editing view of OMEdit in the center. The model browsing window is to the left and a model documentation window is shown at the upper right.

A typical usage of OMEdit is to first create a model using the connection editor, then simulate, and finally plot by selecting which variables should be plotted in the variable plot selection window (Figure 5, lower right).

A model can be created by opening a new empty model and dragging/dropping model components from the model browsing window to the left into the central connection editing area and creating a new model by connecting those components. Alternatively an existing model can be opened by double clicking the model in the model browser window to the left. A model can also be created textually by clicking the text button and typing in Modelica text.

A simulation is performed by clicking on the green right-arrow at the top. After a successful simulation the plot selection window will appear at the right. One rather unusual example of how a plot can appear is visible in Figure 2). There are also variants of the simulation green arrows at the top that combine simulation with debugging or 3D visualization.

4.4 3D Animation and Visualization

The OpenModelica 3D animation and visualization is a built-in feature of OMEdit to animate based on 3D shapes defined by the MSL Multi-Body library. It provides visualization of simulation results and animation of geometric primitives and CAD-files. OpenModelica generates a scene description XML-file which assigns model variables to visualization shape attributes. The scene description file can also be used to generate a visualization controlled by an FMU either in OMEdit or in an external visualization tool as Unity 3D (Waurich and Weber, 2017). In combination with the Modelica_DeviceDrivers Library, interactive simulations with visual feedback and 3D-interactions can be implemented for training, development and testing purposes.

4.5 Debugging and Performance Analysis

4.5.1 The Algorithm Debugger

The OpenModelica algorithm debugger (Figure 7), (Pop, 2008; Sjölund, 2015) is available for use either from OMEdit or from the MDT Eclipse plug-in. The debugger provides traditional debugging of the algorithmic part of Modelica, such as setting breakpoints, starting and stopping execution, single-stepping, inspecting and changing variables, inspecting all kinds of standard Modelica data structures as well as MetaModelica data structures such as trees and lists.

4.5.2 The Equation Model Debugger

The OpenModelica equation model debugger (Figure 8) (Pop et al., 2014; Sjölund, 2015) is available for use from OMEdit. It provides capabilities for debugging equation-based models, such as showing and explaining the symbolic transformations performed on selected equations on the way to executable simulation code. It can locate the source code position of an equation causing a problem such as a run-time error, traced backwards via the symbolic transformations.

In February 2020, new functionality was demonstrated to perform "backward" trace of which variables

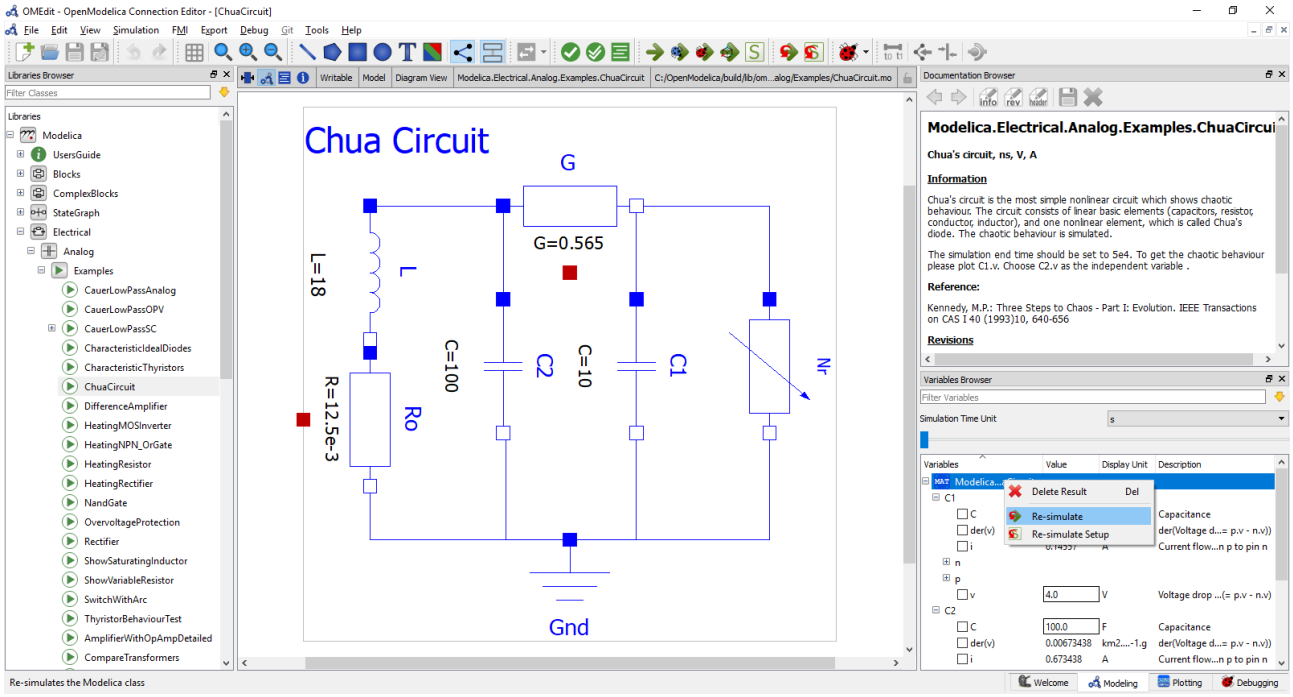


Figure 5: OMEdit on the Modelica.Electrical.Analog.Examples.ChuaCircuit model. *Center*: Model connection diagram. *Upper right*: Information window. *Lower right*: Plot variable browser with a small popup re-simulate menu on top.

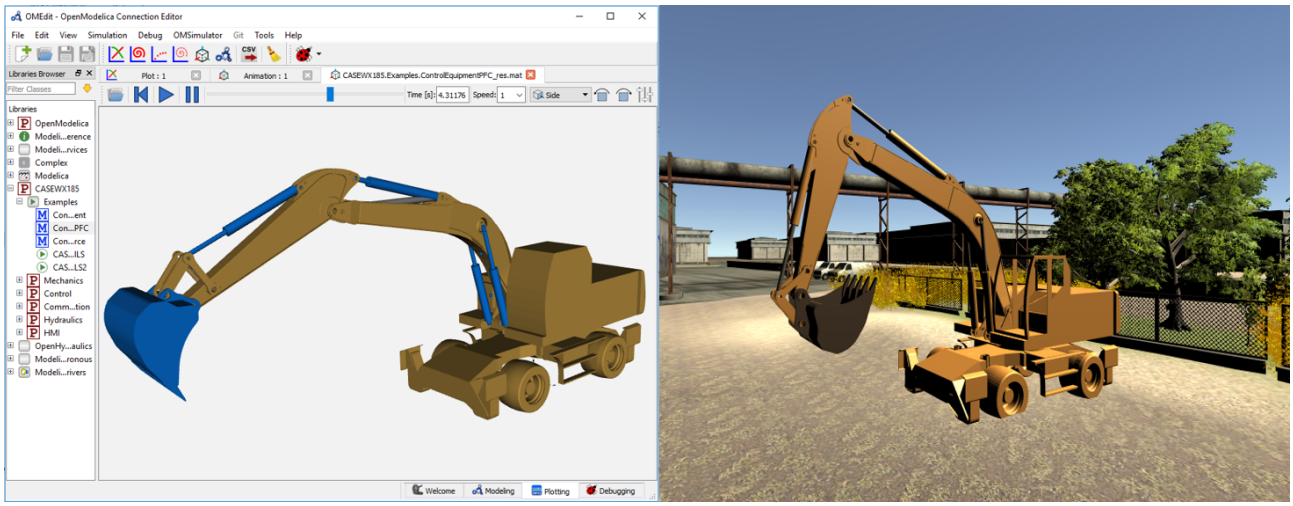


Figure 6: OpenModelica 3D animation of a simulated excavator in OMEdit and in unity 3D.

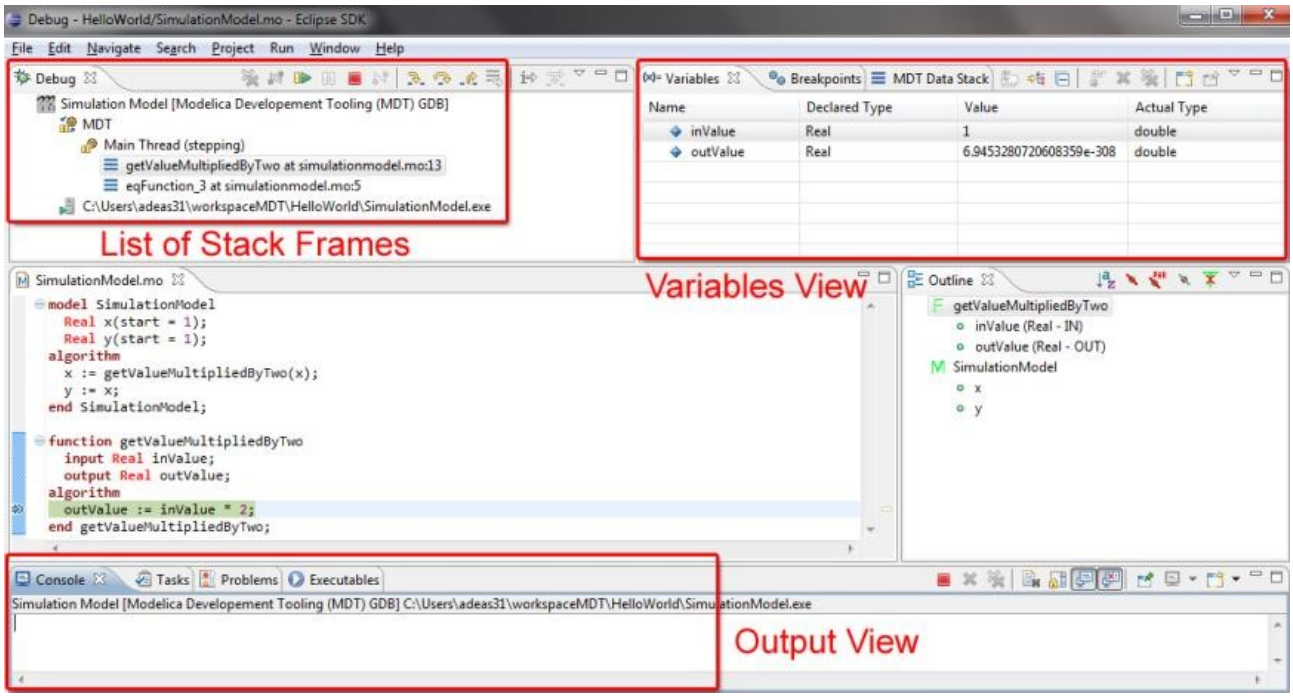


Figure 7: The OpenModelica algorithmic code debugger viewed from the MDT Eclipse plug-in. The OMEdit version of the debugger looks about the same. A breakpoint has been set in the function which is called from the small model called SimulationModel.

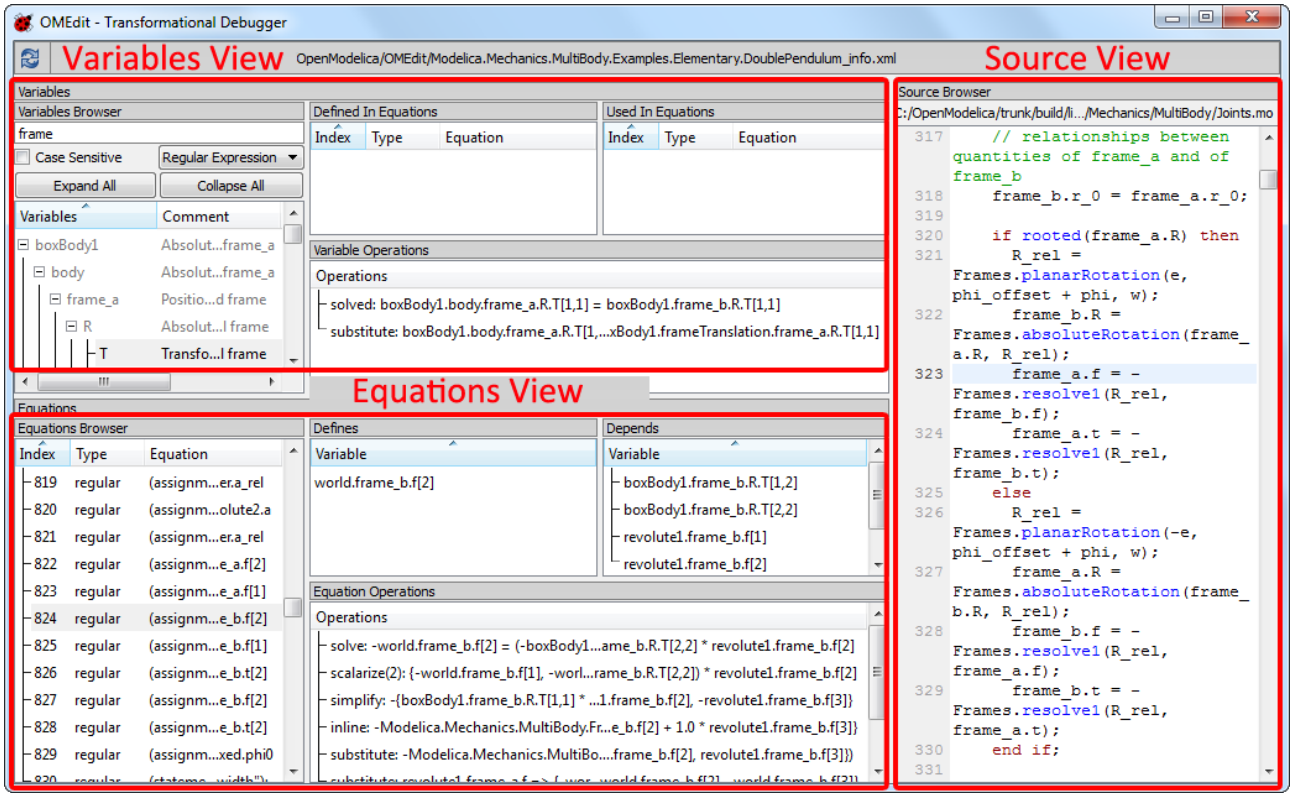
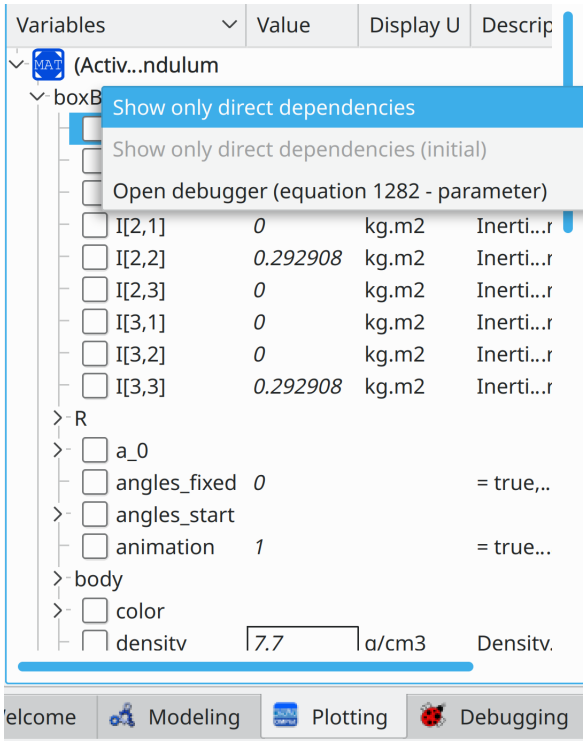
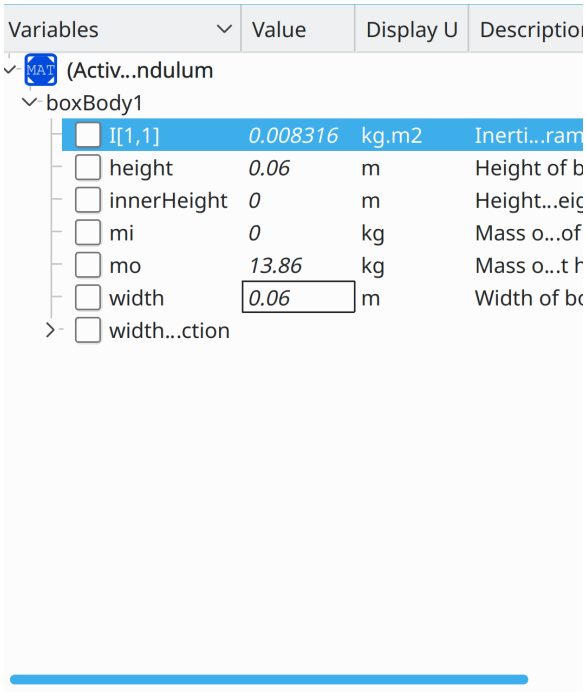


Figure 8: The OpenModelica equation model debugger. *Left*: Equations view where equations and symbolic transformations can be viewed. *Right*: Source view where the erroneous equation is pointed out.



(a) *Left*: Turn on the menu choice.



(b) *Right*: List of variables directly influencing the chosen variable.

Figure 9: Debugger tracing variables or equations influence of a variable.

or equations that directly influence a chosen variable (Figure 9). This can be useful to understand the dependencies causing a faulty variable value.

4.5.3 The Performance Profiler/Analyzer

By using performance profiling analysis it is possible to detect which equations or functions cause low performance during a simulation.

The OpenModelica profiler (Sjölund, 2015) uses compiler-assisted source code instrumentation. There is one call to a real-time clock before executing the equation block or function call and one call to the clock after execution of the block. Associated with each call is a counter that keeps track of how many times this function was triggered for the given time step. Similarly, each call is associated with clock data – one variable for the total time spent in the block for all time steps and one variable for the total time spent in the block for the current time step. The time measurement uses the best real-time clock available on the used platform.

With profiling enabled only for equation blocks (strongly connected equation sets) and functions, the overhead cost is low compared to the cost of solving most nonlinear systems of equations. The profiler is integrated with the equation debugger, which enables the tool to directly point out the equations using a large fraction of the simulation time (Figure 10).

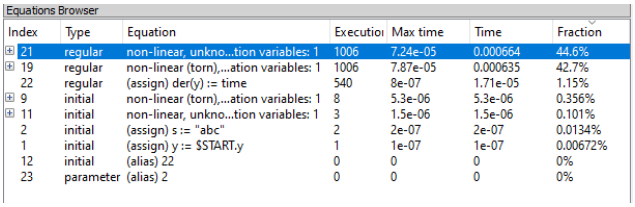
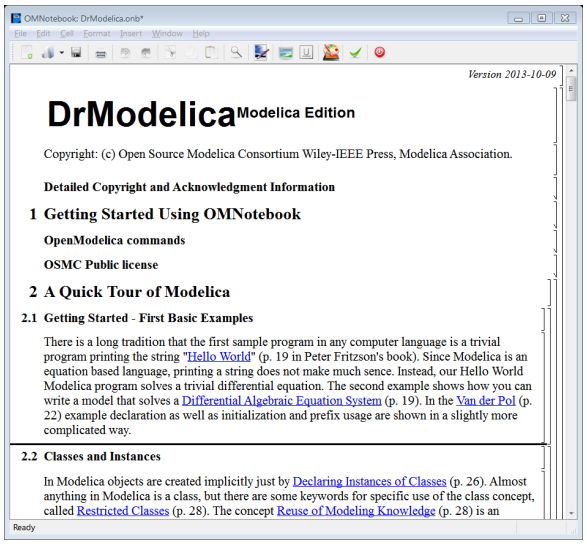


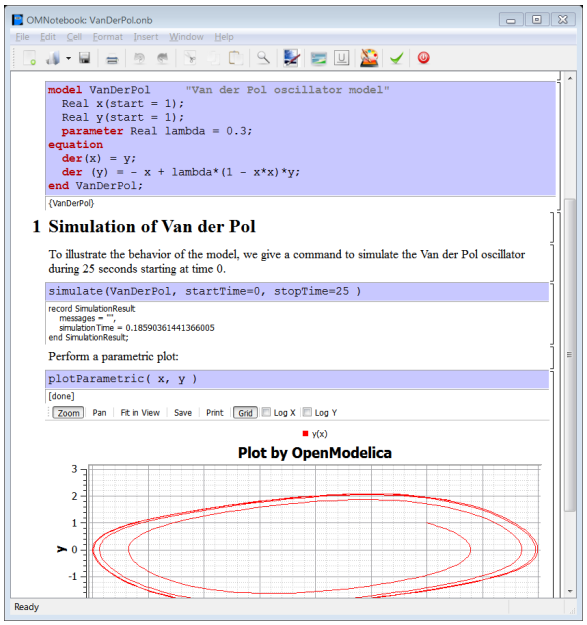
Figure 10: The OpenModelica performance profiler showing which sets of equations use the biggest fraction of the simulation time.

4.6 Teaching with Interactive Electronic Notebooks

Electronic notebooks provide an executable electronic book facility supporting chapters, sections, execution of simulation models, plotting. The first versions of OpenModelica used the proprietary Mathematica notebook facility. Later versions include a simple open source implementation called OMNotebook described below. More recently, a web-based notebook has been developed as well as a plug-in to the Jupyter notebook facility.



(a) *Left*: The DrModelica document start page.



(b) *Right*: The VanDerPol sub-document showing a cell with a Modelica model, simulation commands, and plot results.

Figure 11: The OMNotebook electronic notebook showing part of the DrModelica document (course-material) for learning Modelica.

4.6.1 OMNotebook and DrModelica

OMNotebook (Figure 11) (Fernström et al., 2006) is a book-like interactive user interface to OpenModelica primarily intended for teaching and course material. It supports sections and subsections to any level, hiding and showing sections and cells, interactive evaluation and simulation of Modelica models as well as plotting results. The user can define his/her own books. This tool is useful for developing interactive course material. The DrModelica (Sandelin et al., 2003) interactive Modelica teaching course was the first main application, at that time based on Mathematica notebooks, later translated to use interactive Modelica scripting in OMNotebook.

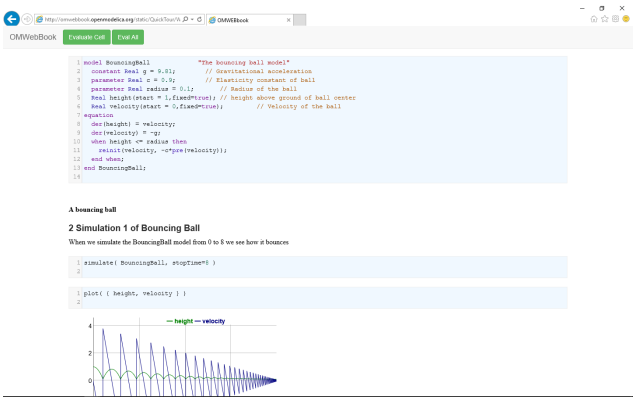


Figure 12: OMWebbook with editable models, simulations, and plots, here simulating the bouncing ball.

4.6.2 OMWebbook – Interactive Web-based Editable and Executable Book

OMWebbook (Figure 12) (Moudgalya et al., 2017; Fritzson et al., 2018b) is an interactive web-based electronic book. This is similar to OMNotebook, but textual model editing and simulation is performed in a web-browser. Simulation is performed by a dedicated simulation server. Thus, the user need not install OpenModelica on a computer. Editing and simulation can even be done from smartphones or tablets.

4.6.3 Jupyter Notebook for OpenModelica

More recently, the Python-based Jupyter notebook software (Project Jupyter, 2016) has appeared, supporting a number of scripting languages. Therefore, based on user demand, we have also developed a Jupyter notebook plug-in for OpenModelica (2020) supporting Modelica scripting. However, Python scripting together with the OMPython package was already available and used in the Jupyter notebooks

presented in Lie et al. (2016).

4.7 Self-Learning Audio-Video Tutorials

A number of interactive audio-video tutorials, called spoken tutorials, have been developed to provide step-by-step teaching about how to use OpenModelica and develop simple Modelica models (Moudgalya et al., 2017; FOSSEE-Modelica, 2020). The audio parts of the tutorials are dubbed in many languages and are suitable for on-line usage (Moudgalya et al., 2017). A total of 14 short Spoken Tutorials of 10 minutes each are available, and a total of 10,000 students have been trained using these tutorials at the time of writing this article.

4.8 Text-book Companions for Teaching and Prototyping

Most open source software projects rely on contributions from the user community. Students form a substantial fraction of this community. One of the shortcomings of Free and Open Source Software is inadequate documentation, and the lack of contributions to it by students aggravate this problem: Students often lack motivation to document or are not capable of creating good documents. The converse is often true: Students are much better coders and they often enjoy coding. We addressed the above mentioned problem by solving the inverse problem: Ask students to write code for existing documents. For students, documents are textbooks.

A textbook companion comprises a collection of code for all relevant solved problems in a textbook (Moudgalya, 2018). A student who has understood the concepts in a textbook may be motivated to learn an appropriate open source software and code the solved examples, verifying the correctness at every step. There is no document containing the code and hence there is very little chance of copyright violations.

The student community has already created a large number of textbook companions for OpenModelica (Moudgalya, 2018; FOSSEE-OM-Textbook, 2020). At the time of this writing, we have OpenModelica textbook companions for 56 books. Textbook companions are extremely useful as documents. Anyone who needs to know the syntax and semantics for a command could locate a solved example that has the underlying calculation and the corresponding code. A textbook companion can also be used by course instructors to carry out what-if studies on solved examples. Finally, if a large number of textbook companions are created, a course instructor can use a database of such documents to set problems.

4.9 Interactive Scripting APIs using Modelica, Python, Julia, and Matlab

Interactive scripting APIs (Application Programming Interfaces) are provided for several scripting languages using interactive read-eval-print loops.

There is an interactive session handler, OMShell, that parses and interactively interprets commands and expressions in Modelica for evaluation, thus providing Modelica scripting. The session handler also contains simple history facilities, and completion of file names and certain identifiers in commands.

Interactive sessions handlers with scripting APIs to OpenModelica are also provided for the languages Python (Python Software Foundation, 2018), Julia (Julialang, 2018), and Matlab (MathWorks, 2018), through the subsystems OMPython (Lie et al., 2016), OMJulia (Lie et al., 2019) and OMMatlab (OpenModelica, 2020). This gives the user the possibility to use Modelica together with the rich set of facilities and libraries in these languages, e.g. for tasks such as control design and post processing of simulation results.

More precisely, the scripting language APIs (OMPython, OMJulia, OMMatlab) provide methods for

(i) establishing objects of Modelica code within the scripting language,

(ii) getting and setting Modelica model parameters,

(iii) getting and setting piece-wise constant inputs over a time interval,

(iv) getting and setting simulation options,

(v) carrying out simulation and getting solutions,

and more. Initial states can be set via parameters.

4.10 FMI – Functional Mockup Interface

4.10.1 FMI Import and Export

The FMI (Functional Mockup Interface) standard describes a way of describing and packaging causal models in either binary or source-code form. Many tools (including Modelica tools) support exporting models from their native modeling representation into FMI form. The standard is widely used in industry, especially the automotive industry which initially drove the development. Today, the Modelica Association is maintaining the standard and continuously developing it further. A model or simulation unit is called FMU (Functional Mockup Unit) according to the standard. Regarding export from Modelica tools, compared to a Modelica model which is usually acausal, an exported model in FMU form is less general since it is causal

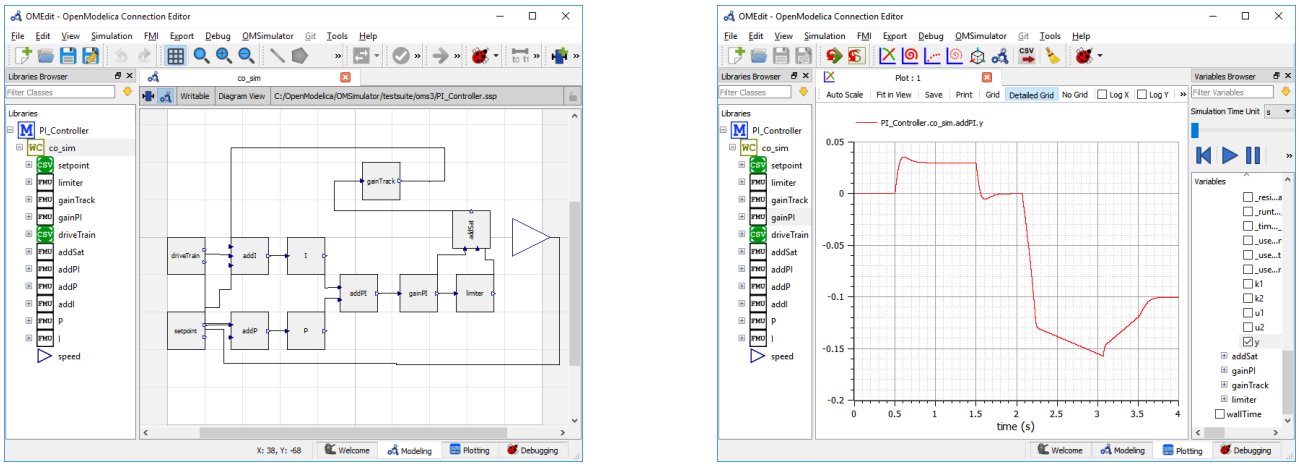


Figure 13: The OpenModelica OMSimulator composite model editor (left) and simulator right.

– the causality of ports has to be fixed. The OpenModelica toolset can be used to both export any given Modelica model as an FMU and import FMUs to create a composite model.

4.10.2 OMSimulator – FMI and TLM-based Simulation/Co-simulation and Composite Model Editor

Simulation according to the FMI standard can be done using model-exchange FMUs (exported models without a solver), co-simulation FMUs (exported models including an embedded solver), or tool-to-tool co-simulation. Standard Modelica simulation uses the same solver for all included model components, which is the approach used for model-exchange FMUs. Co-simulation mechanisms that synchronize several solvers have to be used for co-simulation FMUs, which sometimes may cause numerical stability issues.

OMSimulator (2020) is an OpenModelica subsystem that provides efficient simulation and co-simulation of FMUs. Thus, models from non-Modelica tools compiled into FMUs can also be utilized and simulated. Furthermore, models that cannot be exported as FMUs can be integrated in a simulation using tool-to-tool co-simulation. This is provided via wrappers to models in tools such as ADAMS (MSCSoftware, 2020), Beast (Fritzson et al., 2014; Fritzson, 2018), Simulink (MathWorks, 2019a), Hopsan (Axin et al., 2010), or co-simulation of FMUs with embedded solvers. The system can optionally be used with TLM (Transmission Line Modeling) connectors, which provide numerically more stable co-simulation.

The previous version, OMSimulator 1.0 (2017) was already made available in OpenModelica 1.12.0 (Fritzson et al., 2018a). However, it was strictly restricted to TLM-connections between components.

OMSimulator is provided together with a composite model editor integrated in OMEdit (Figure 13), that allows combining external models (e.g. FMUs for both model-exchanged and co-simulation) into new composite models, simulating them and in some cases (for the TLM version) perform 3D animation. Composite models can be imported and exported by using the SSP standard (Systems and Structure Parameterization) standard (Modelica Association, 2018; OpenModelica, 2020).

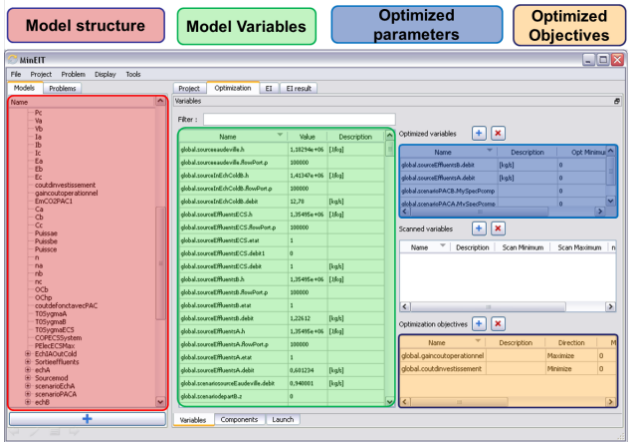
4.11 Parameter System Identification

OMSysIdent (2020) is a system parameter identification module built on top of the OMSimulator (2020) API. For estimating the sought parameter values, a system model needs to be provided as FMU, as well as respective measurement data of the system. The API of OMSysIdent is integrated with the scripting interfaces of OMSimulator and OpenModelica (using Lua or Python scripting). Internally, the module uses the Ceres Solver (Agarwal et al., 2018) library for the optimization task.

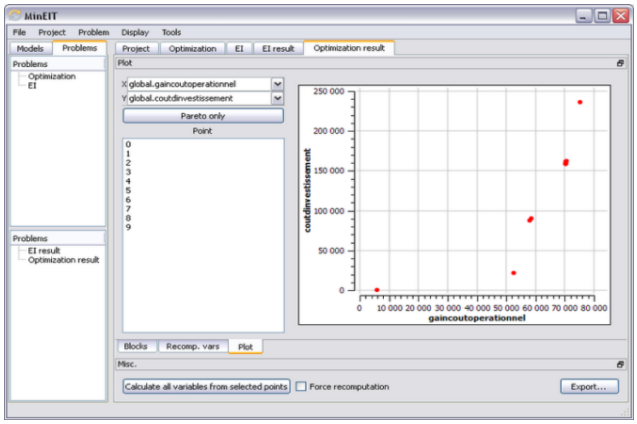
4.12 Parameter Search-Based Design Optimization

An important use for modeling and simulation is to improve a system design, usually before it is physically realized and manufactured. In this process it is customary to perform a number of simulations for different values of the design parameters, until a design has been obtained that best fulfills a given set of design criteria.

The traditional parameter sweep based design optimization performs many simulation runs while sweeping, i.e., performing a linear search, of the desired pa-



(a) Selecting variables, objectives, parameters.



(b) A result plot with a Pareto optimization of two goal functions.

Figure 14: The OpenModelica OMOptim tool for parameter sweep optimization.

rameters over an interval in order to find an optimal value of a goal function or goal variable. The drawback is the very large number of simulations that might be required. For example, three parameters each with an interval that is subdivided into 100 steps would require one million simulations to cover all combinations for these parameters.

4.12.1 The OMOptim Tool with Genetic Algorithms for Parameter Search

The OMOptim OpenModelica tool (Figure 14), (Thieriot et al., 2011) provides a GUI and uses genetic algorithms (simulated annealing) during parameter exploration as a search heuristic to find an optimal parameter setting.

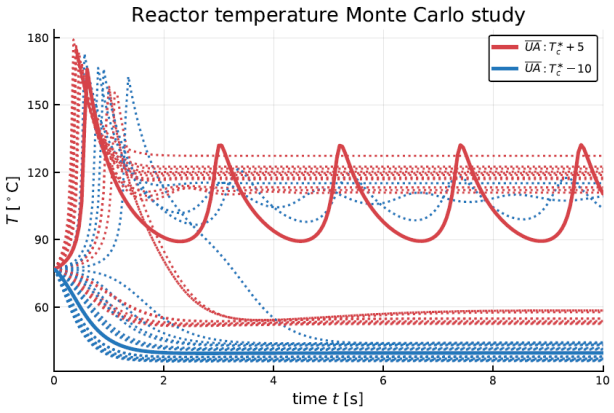


Figure 15: Sensitivity of reactor temperature to randomly varied heat transfer coefficient UA. Nominal parameters (solid) for increase in cooling temperature Tc (red) and decrease in cooling temperature Tc (blue).

4.12.2 Parameter Sweep Optimization based on Python, Julia, or Matlab Scripting

With a simulation model expressed in Modelica, and a cost function either expressed in Modelica or the scripting language, for-loops in a scripting language, Section 4.9, such as Python, Julia, Matlab, and to some extent Modelica can be used to compute how the cost function varies with changing values of parameters. Typical scripting language code for carrying out this operation is as follows (Julia syntax):

```
# mod -- model object
# sweep over np parameter values
np = 20
# parameter vector
par = range(1.0, 10.0, length =np)
# vector for storing resulting cost
cost = zeros(np)
for i in par
 setParameters(mod ,"p = $(par [i])")
 simulate(mod)
 cost[i] = getSolutions(" cost ")[end]
end
```

This idea can trivially be extended to multi parameter problems, including parameterization of inputs. To find parameters which, say, minimize the cost, one can then simply search for minimal point in the cost array, or by fitting the cost array data to some analytic parametric function, and then find the minimum of the analytic function.

To illustrate the idea of parameter sweep, Figure 15 shows how the solution of a reactor model (Seborg et al., 2011) changes for a randomly drawn heat transfer coefficient within a range.

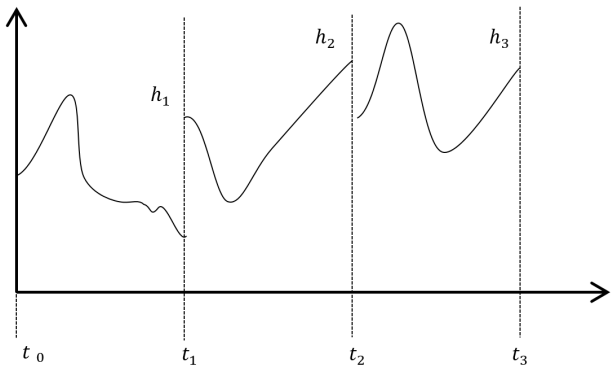


Figure 16: Dynamic optimization formulates the whole trajectory in terms of trajectory segments whose shapes are adjusted during optimization.

4.13 Dynamic Optimization Using Collocation

Another approach, dynamic optimization using collocation (Figure 16), (Bachmann et al., 2012; Åkesson, 2008; Ruge et al., 2014; Houska et al., 2011), avoids the combinatorial explosion of multiple simulation runs since only a single model evaluation is required to find an optimal trajectory. This is at the cost of being very sensitive to the model – the methods may typically not converge except for small models, i.e., not at all robust.

A collocation method formulates an optimization problem directly on a whole trajectory which is divided into trajectory segments (Figure 16) whose shapes are determined by coefficients which are initially not determined.

During the optimization process these coefficients are gradually assigned values which make the trajectory segments adjust shape and join into a single trajectory with a shape that optimizes the goal function under the constraints of fulfilling the model equations.

The systems to be optimized are typically described using differential-algebraic equations (DAEs), which can be conveniently formulated in Modelica. The corresponding optimization problem can be expressed using graphical or textual formulation based on annotations.

Solution algorithms based on collocation methods are highly suitable for discretizing the underlying dynamic model formulation. Thereafter, the corresponding discretized optimization problem can be solved, e.g. by the interior-point optimizer Ipopt (Wächter and Biegler, 2006). The performance of the optimizer heavily depends on the availability of derivative information for the underlying optimization problem. Typically, the gradient of the objective function, the Jacobian of the DAEs as well as the Hessian matrix of the corresponding Lagrangian formulation need to be determined. If only some or none of these derivatives are provided, usually numerical approximations are used. The generation of symbolic Jacobian is already available in OpenModelica (Braun et al., 2012; Shitahun et al., 2013) and the generation of symbolic Hessian is currently under development.

The main symbolic transformation steps during compile time and the dynamic optimization tool chain for OpenModelica with Ipopt are visualized in Figure 17.

The optimization can be called via a batch process using the following commands:

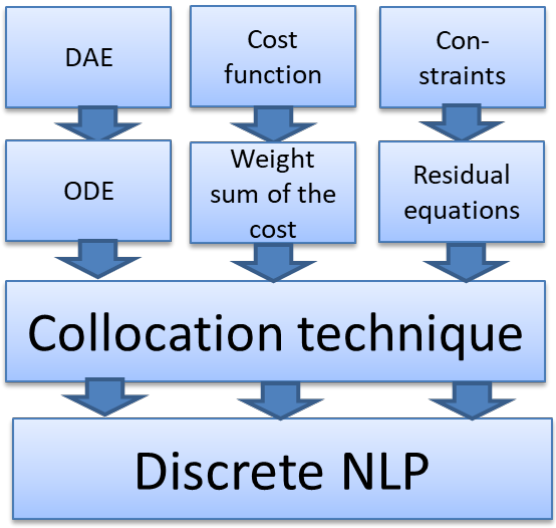
```
setCommandLineOptions("+gDynOpt");
loadFile("...");
optimize(nmpcProblem,
    numberOfIntervals=20,
          tolerance=1e-8);
```

The implementation has been tested with several applications and is demonstrated in the following using a combined cycle power plant model, see Figure 18. The model contains equation-based implementations of the thermodynamic functions for water and steam, which in turn are used in the components corresponding to pipes and the boiler. The model also contains components for the economizer, the super heater, as well as the gas and steam turbines. The model has one input, 10 states, and 131 equations. Additional details on the model are presented in (Casella et al., 2011a).

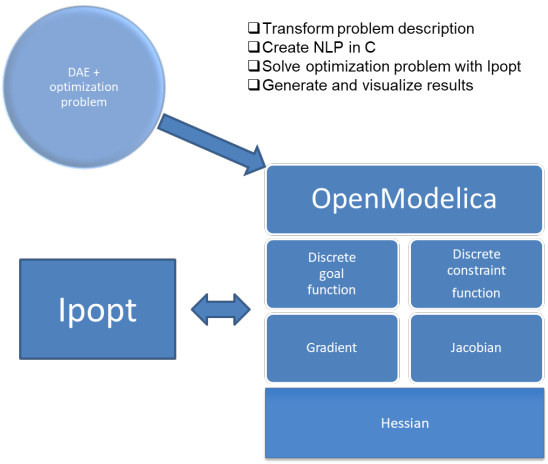
The optimization problem is set up to use 50 collocation points that result in 1651 variables for the nonlinear optimal control problem and was solved on a PC with a 3.2GHz Intel(R) Core(TM) i7. The algorithm requires an initial trajectory of all problem variables, which is provided by a simulation where the rate of change of the gas turbine load is set to a constant value. The optimization results are shown in Figure 18 and correspond with the results that are discussed in detail in (Casella et al., 2011a). Here, the trajectories are smoother, and the performance has been improved substantially.

4.14 Parameter Sensitivity Analysis Based on Optimization

The sensitivity of non-linear models in the form of ordinary differential equations is understood as the tendency to undergo qualitatively noticeable changes in response to shifts in the parameters used for the model setup (Khalil, 2002). Given a nonlinear model there exists an interest in automatically and efficiently detecting small sets of parameters that can produce strong changes in state variables when perturbed within ranges smaller than the uncertainty bounds.

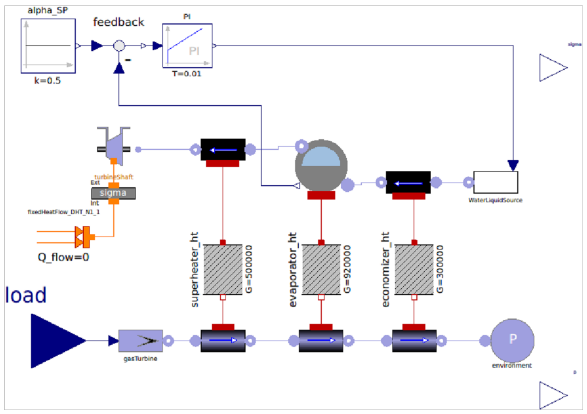


(a) Symbolic preprocessing and transformation.

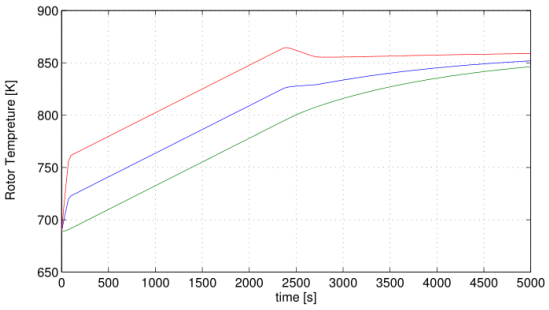


(b) Optimization tool chain.

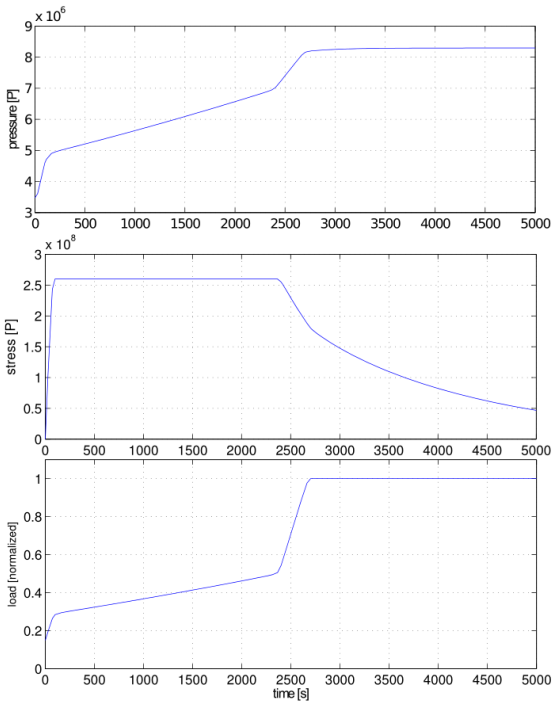
Figure 17: OpenModelica with Ipopt



(a) Combined cycle power plant displayed with OMEdit



(b) Optimal start-up trajectories. The upper curve shows the live steam temperature, the middle and low curves show the turbine rotor surface and mean temperatures.



(c) Optimal start-up trajectories. The upper curve shows the pressure in the evaporator, the middle curve shows the thermal stress in the steam turbine shaft and the lower curve shows the control input represented by the load.

Figure 18: Combined cycle power plant model in OpenModelica

Object-oriented modeling languages such as Modelica (Modelica Association, 2017; Fritzson, 2014) facilitate a systematic treatment of the problem by exposing a clear and unambiguous access to model parameters, state variables, and simulation configuration. This promotes the design of reusable frameworks that treat the models as black boxes (not excluding exploiting internal knowledge on the model structure) The OpenModelica (2020) tool also includes a Sundials/IDA solver that calculates parameter sensitivities using forward sensitivity analysis. Yet, this approach cannot be applied to models that are not fully differentiable. The option of picking several values within a parameter interval and sweep all possible combinations quickly leads to a combinatorial explosion that renders the approach unfeasible. In the simultaneous approach, after defining the parameters and their intervals, an algorithm (typically an optimization-based strategy) finds a vector of smallest perturbation values that produces the largest impact on the state variables. The OMSens OpenModelica (2020, ch. Paremeter Sensitivitites with OpenModelica) sub-system is a tool to assess the sensitivity of Modelica models (Danós et al., 2017). OMSens uses different methods for sensitivity analysis including robust, derivate-free non-linear optimization techniques based on the CURVI family (Dennis Jr. et al., 1991).

4.14.1 Sensitivity Analysis of Modelica models

Unlike most previous approaches, OMSens offers a wide choice of computational methods for sensitivity analysis. Elsheikh (2012) uses automatic differentiation to augment the model with the sensitivity equations. This is similar to the IDA Solver approach in OpenModelica (2020, ch. Paremeter Sensitivitites with OpenModelica) which is simpler to employ since it numerically computes sensitivities directly. Wolf et al. (2008) compares several methods including parameter-sweep and solver-based approaches using the DASPK solver (Petzold et al., 2006). Many optimization methods can be employed for sensitivity analysis. For example, Ipopt (Wächter and Biegler, 2006) is a well-known non-linear optimization routine. Other methods are mentioned in Section 4.12 and Section 4.13, some of which are time consuming or not at all robust.

4.14.2 Optimization-driven Sensitivity Analysis

Numerical aspects of the optimization algorithms need to be considered carefully, as they affect their efficiency, robustness and scope of applicability. A correct analysis should consider the combined, simultaneous effects of many perturbations of the parameters, something that is unmanageable due to the number of combinations and the impossibility of determining beforehand the size of those perturbations. Nonlinear optimization can be used to solve the problem by reformulating it as a model stability problem (Danós et al., 2017).

In the current version OMSens implements a derivative–free optimization algorithm named CURVI – curvilinear search method, Dennis Jr. et al. (1991) which is able to solve very difficult problems while allowing for custom interval constraints. There are three versions: CURVIF, CURVIG, CURVIH that use, respectively, function values, function values plus gradients, and the latter plus Hessians. All versions are globally convergent.

CURVIF is the flavor currently adopted in OMSens, and does not necessarily employ the least number of function evaluations. It can be seen as a trade-off between robustness and efficiency. Moreover, global optimization functionality is currently being added to OMSens.

4.14.3 OMSens Architecture

OMSens provides a flexible experimentation arena of different sensitivity analysis strategies for Modelica models. It provides modularity by being split into decoupled backend and frontend modules. It also provides flexibility since the backend is subdivided into modules that encapsulate responsibilities and expose clear invocation interfaces.

The OMSens modules can be divided in two groups: simultaneous sensitivity analysis and individual sensitivity analysis. In Figure 19 we find six main modules. In the simultaneous scenario, module 3 (Optimization) leads the workflow, invoking modules 1, 2 and 4 to perform an exploration of the parameter space. This needs successive simulations requested from module 2 (Modelica) depending on the simulation results of previous invocations, following a closed loop strategy. In the individual scenario modules 5 and 6 lead their own workflows, invoking single simulations with no dependency on the results of previous runs (open loop). Module 6 (Parameter sweeping) invokes simulations while sequentially picking values from a parameter space defined by the user.

A summary of the presented sensitivity analysis methods can be found in Table 6.

A sensitivity method measures the change of a chosen variable/state variable with respect to changes in one or more parameters. OpenModelica (2020, ch. Paremeter Sensitivitites with OpenModelica) can calculate sensitivities using the Sundials/IDA solver using the derivatives of each state variable with respect to each top-level parameter during a simulation (IDAsens method) defined for all values of time. OMSens can launch experiments using the IDASens method. OM-

Table 6: Summary of sensitivity methods for a generic state variable x with respect to the i-th parameter p_i.

Method	Formula	Input Type
IDAsense	$\dot{s}_i(t) = \frac{\partial x(t)}{\partial p_i}$	Single year
Rel	$s_{rel}(t) = \frac{\sigma(t)}{std(t)}$	Single year
RMS	$s_{rms}(t_0, t_f) = \sqrt{\frac{1}{n}(\sigma_0^2 + \cdots + \sigma_n^2)}$	Range of years

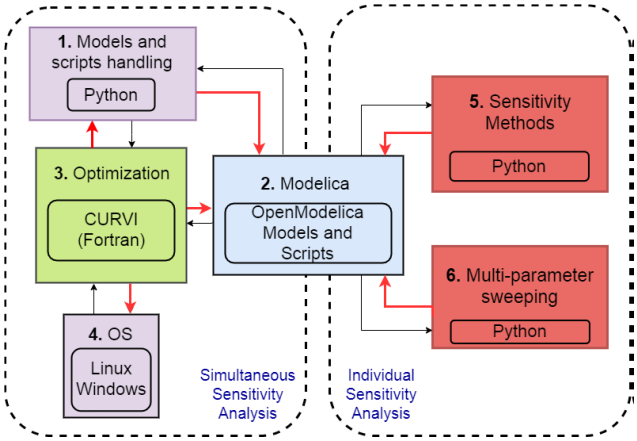


Figure 19: OMSens high level architectural view.

sens allows defining custom methods for comparisons of perturbed vs. unperturbed runs. For example, we use a Relative (Rel) method defined as

$$s_{rel}(t_k) = \frac{\sigma(t_k)}{x(t_k)} \quad \text{with} \quad \sigma = x_{per} - x,$$

i.e. the difference in a state variable with and without perturbation of a parameter (x_{per} vs. x).

It can be used to rank the parameters affecting a variable the most at a target year. We also define the Root Mean Square (RMS) method $s_{RMS}(t_0, t_f)$ that calculates the root mean square of the differences $\sigma(t_k)$ for integer years $t_0 \leq t_k \leq t_f$. It can be used to rank the most relevant parameters impacting a variable throughout a range of years.

4.14.4 Case Study – Sensitivity Analysis of a Complex Socio-Economic Model

World3 is a world-level socio-economic model available as a Modelica library (Cellier, 2008), here referred to as W3-Mod. It implements the World3 model as described in Meadows et al. (2004, Limits to Growth, 3rd edition) meant to forecast critical global aspects (population, pollution, natural resources) as far as year 2100. W3-Mod subdivides the model into 13 socio-economic sectors, with a total of 41 state variables, 245 algebraic variables, and 70 parameters (including bivariate table functions) representing many facets of human life and ecological factors.

In a separate earlier book by Meadows et al. (1974), simplistic sensitivity experiments were considered, but W3-Mod lacks a comprehensive sensitivity study. The model has long been characterized as strongly nonlinear and unstable with regards to parameter changes (Castro, 2012; Scolnik, 1979; Vermeulen and de Jongh, 1976).

Applying optimization-based analysis, we used OMSens to analyze the state variable Population of W3-Mod at year 2100, perturbing the top-12 most influencing parameters found using the Rel analysis method for that state variable. We allowed all parameters to change within a ±5% interval (conservative bounds for socio-economic indicators). CURVI found three non-intuitive perturbation values compared to the results obtained with the Rel method alone, not shown here.

Parameters `p_land_yield_fact_1` and `p_avg_life_ind_cap_1` (Default land yield factor and Default average life of industrial capital) were perturbed in opposite directions compared to what an individual parameter-based approach would indicate. With these differences the impact is noticeable. In Figure 20 we observe several simulations to interpret the results, all extended up to the year 2500 for the sake of readability. The black curve is the unperturbed run. The red curve is the run perturbed with the individual parameter-based strategy (Rel method) and the green curve represents perturbations found by CURVI. We can see that for the target year 2100 CURVI found a parameter combination that takes the population way up compared to what was achieved relying solely on the Rel method.

Verification with multi-parameter sweeping is available to automate the simulation and analysis of arbitrary combinations in a parameter space. The optimization-based method yields a substantial improvement compared to an individual parameter-based study. We now assess whether other perturbations offer extra insights. We create a space for both parameters

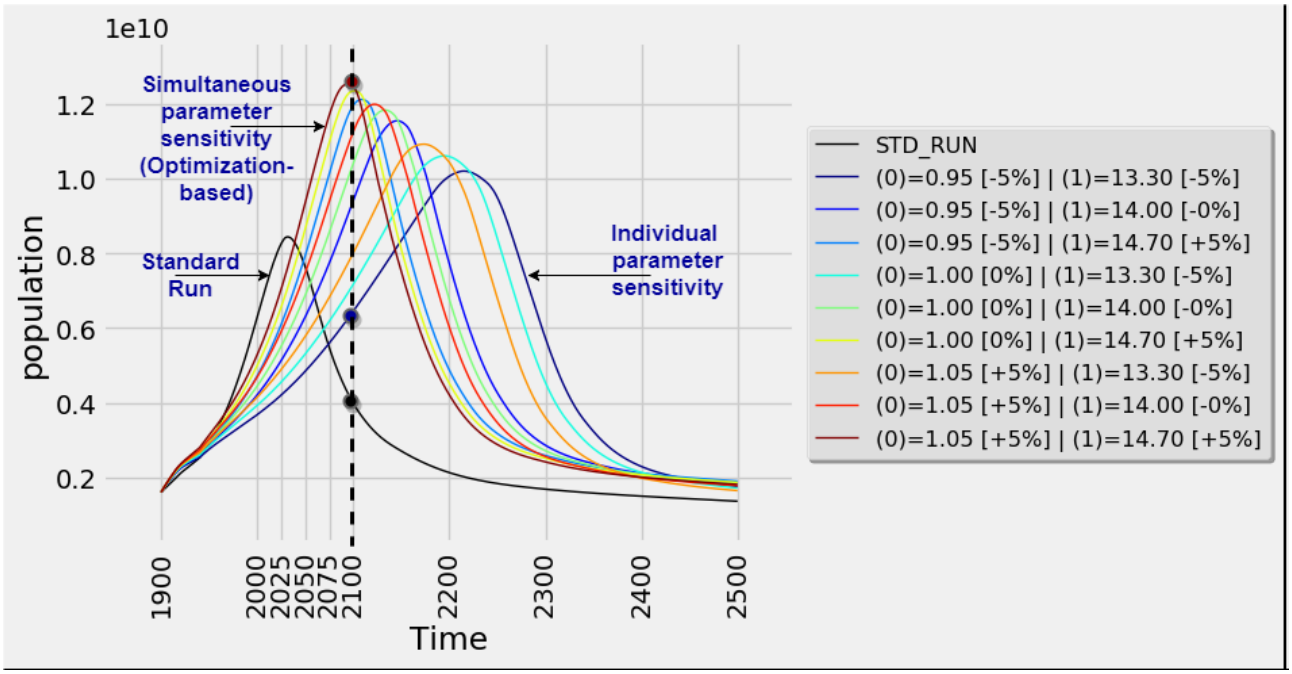


Figure 20: Population in W3-Mod.
Standard Run and OMSens perturbations: Rel method, multi-parameter sweep and CURVI.

using the same perturbation vector $[-5\%, 0\%, +5\%]$ and launch the analysis. The results in Figure 20 are denoted with (0) and (1) for values of land yield and industrial capital parameters. We observe that the population variable converges smoothly from the individual parameter-based to the simultaneous parameter-based study.

4.15 Model-based Control with Dynamic Optimization

Modelica has been applied to the formulation of dynamic optimization problems for complex physical systems since many years (Franke et al., 2003). The optimization methods of control vector parameterization and multiple shooting enable the efficient use of simulation models with numerical optimization solvers and the treatment of large-scale problems with parallel computing. Many successful industrial applications to model-based control of power systems underline the suitability. The Modelica technology and the implementation in OpenModelica evolved continuously.

FMI 2.0 for model exchange standardizes the solver interface to simulation models. It covers executable model code, an XML interface description, sparse model structures and analytic Jacobian matrices. Multiple FMI instances of one and the same model enable parallel optimization. The OpenModelica C++ runtime was developed with focus on the export of simulation models to real-time control applications. C++ provides for improved type safety, deterministic memory management and compiler optimizations resulting in best in class execution times (Franke et al., 2015).

4.15.1 Synchronous Modelica for Model-Based Control

Continuous-time physical models are treated with discrete sample times in digital control applications. Modelica's synchronous language elements extension was introduced for precisely defining and synchronizing sampled-data systems with different sampling rates (Modelica Association, 2017). OpenModelica was the second Modelica tool which supported this extension, implemented both on top of the OpenModelica C runtime and the C++ run-time. This can be used for model-based control, using Modelica and FMI (Franke et al., 2017).

4.15.2 Control and Optimization of Electric Power System as Exported FMUs

Figure 21 shows OMEdit with an example model of an electric power system, covering an off-shore interconnector combined with wind farms and a back-to-back HVDC coupling. The object-oriented model comprises 4 722 variables. 788 of those variables are non-trivial. OMEdit exports the model as FMU 2.0 using the OpenModelica C++ runtime.

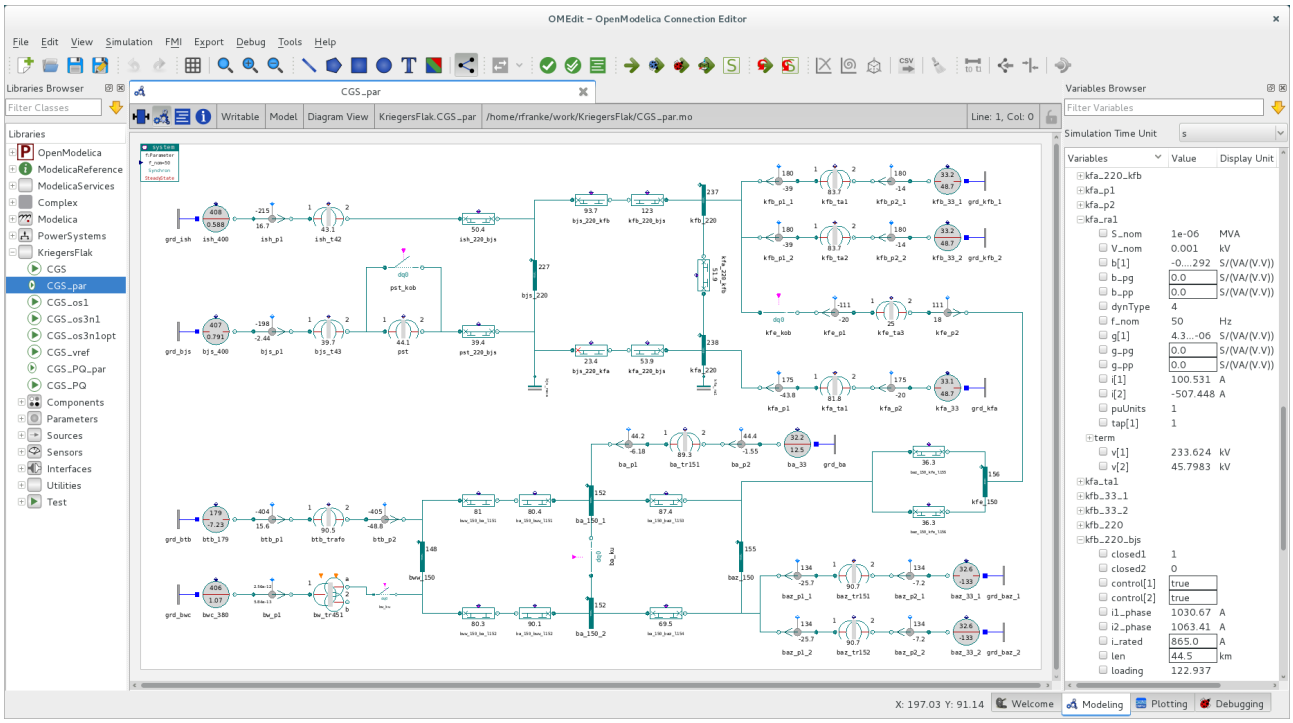


Figure 21: Model of an electric power system.

The control task is to maximize the transfer capacity of the interconnector on top of collected wind power, considering active and reactive power flows subject to grid and voltage limitations. The prediction horizon spans 96 sample intervals. This results in 76 436 ($97 \cdot 788$) non-trivial optimization variables.

Table 7 shows speedups achieved with different FMU features and optimization solver configurations. The reference configuration uses multiple shooting with 1 CPU, exploiting sparsity that results from the time staggering structure over 96 intervals, but neglecting sparsity inside model Jacobians at each time point. The Jacobians are obtained with the method of finite differences. A speedup of 1.9 is achieved with sparse Jacobians by exploiting information in the XML interface description. The speedup increases to 3.9 when additionally enabling OpenModelica's algorithmic differentiation and re-use of numerical factors of equation systems inside the model.

Parallel multiple shooting using a separate FMU instance for each CPU further increases the speedup to 6.3 with 5 CPUs and up to 7.6 with 20 CPUs. The speedup is still 7.0 with finite differences and 20 CPUs. This is at the cost of twice the CPU usage though, whereas algorithmic differentiation leaves more CPU capacity for other tasks running at the same time.

4.16 Model-based Control System Design

In addition to the scripting API commands mentioned in Section 4.9, the APIs OMPython (Lie et al., 2016), OMJulia (Lie et al., 2019), OMMatlab (OpenModelica, 2020) also allow for getting and setting linearization options, and carrying out linearization. The linearize method returns a tuple of linear, time invariant (LTI) matrices (A,B,C,D), which can be further used in various control tools, e.g., the MATLAB Control System Toolbox (MathWorks, 2019b), the Python Control Systems Library (Murray and Livingston, 2019), or the Julia Control Systems Toolbox (JuliaControl, 2019).

We consider a liquid reactor (Figure 22) taken from (Seborg et al., 2011), where we seek to control the effluent temperature by manipulating the influent cooling temperature.

The model is implemented in Modelica, and an object is created in a scripting language (here: Julia, using OMJulia). At the operating point, the linearize method is used to find an LTI approximation. Based on this LTI approximation, Julia's ControlSystems package can be used to do a root locus plot (Figure 23) on how the closed loop eigenvalues vary with proportional gain in a Proportional controller (P-controller),

Based on the root locus plot, a suitable controller gain for the P-controller can be found, and likewise a suitable reset time/integral time in a Proportional+Integral controller (PI controller). The result-

Table 7: Speedup achieved with different FMU features and parallel computing.

Speedup for different solver configurations	Finite Differences	AD with refactoring	AD with factor reuse
Sequential shooting, Dense model blocks, no AD, 1 CPI	1.0	-	-
Sequential shooting, Sparse model blocks, 1 CPU	1.9	1.3	3.9
Parallel multiple shooting, Sparse model blocks, 2 CPUs	2.8	2.2	4.6
Parallel multiple shooting, Sparse model blocks, 5 CPUs	4.6	3.7	6.3
Parallel multiple shooting, Sparse model blocks, 20 CPUs	7.0	6.5	7.6

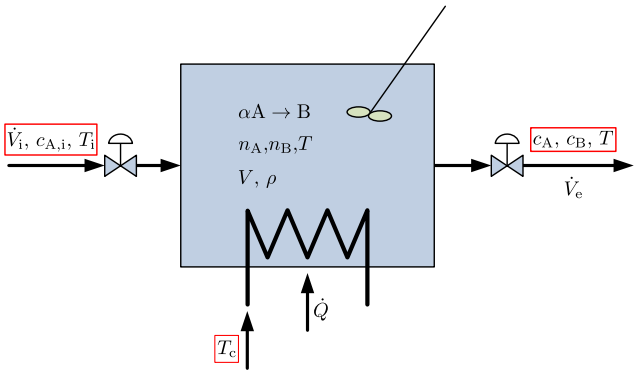


Figure 22: Cooled liquid reactor.

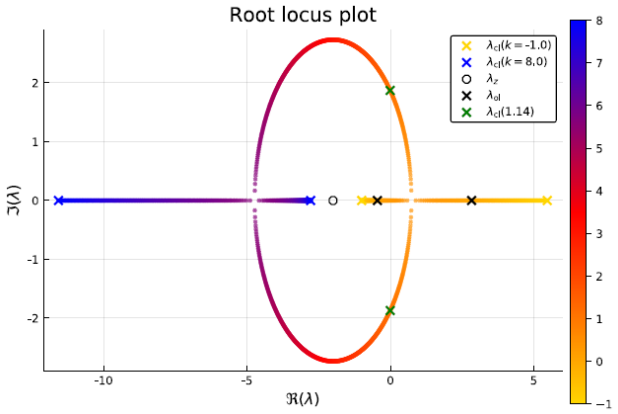
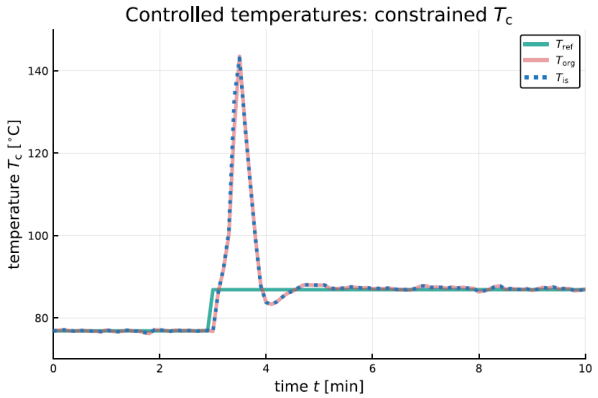
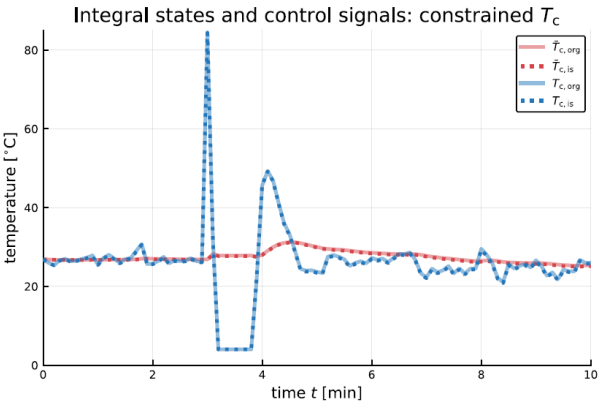


Figure 23: Root locus plot of Seborg reactor. A modified plotting routine has been used, instead of the root locus method of the ControlSystems package.



(a) Reactor temperature controlled with a constrained PI controller with anti-windup. A continuous controller is used, implemented in Modelica.



(b) PI control signals — cooling temperature, with constrained PI controller with anti-windup. A continuous controller is used, implemented in Modelica.

Figure 24: Results for output T and controller T_c

ing PI controller gives quite good control of the reactor temperature.

However, the resulting cooling temperature input dips down to minus twenty Celcius, which is unrealistic. Constraining the cooling temperature to lie in the range of liquid water, together with integral anti-windup leads to the results in Figure 24a and Figure 24b for output T and controller T_c, respectively.

More details of this example can be found in Khalili and Lie (2018), with other examples including state estimation and Linear Quadratic control.

4.17 Model-based Fault and Dependability Analysis

The purpose of reliability, and more generally, of dependability studies is to evaluate non-functional performances, that is, to calculate probabilities of undesirable events such as the failure of the mission of a system, or to estimate the probability distribution of quantities like: total production on a given time interval, maintenance cost, number of repairs etc. Usually, dependability studies are performed with dedicated methods and tools, based on discrete (and often even Boolean) models of systems: fault trees, Markov chains, Petri nets, BDMP (Boolean logic Driven Markov Processes) etc. EDF (Electricité du France) designed the Figaro modeling language in 1990 (Bouissou et al., 1991). This language generalizes all the above cited models, and allows casting knowledge in categories of systems in libraries. It is the basis of KB3 which is the reference tool used for building fault trees and dynamic models for probabilistic safety analyses of nuclear power plants and most other reliability analyses at EDF.

In order to benefit from this type of analysis a prototype for coupling Modelica models with their Figaro counterpart has been developed (Bouissou et al., 2016). This coupling presented two main issues:

- The systems are modeled with different degrees of granularity in the two languages.
- Links are explicit objects in the Figaro world that can have properties and behavior, whereas the default port connections in Modelica are not objects. When mapping Figaro to Modelica this is handled by letting connections go through intermediary objects that contain that information.

Therefore, the mapping between Modelica and Figaro components is not one-to-one. Instead component types with a Figaro counterpart are identified in Modelica through special interfaces and this information is then used to export the Figaro model from the corresponding Modelica model.

The reliability analysis performed on the Figaro model can be then used to identify potential issues (for example, critical components) and this information can be fed back into the Modelica simulation (for example, investigate in more details the effect of the failure of a critical component).

Another approach has been to use Monte Carlo simulation on Modelica models. In order to do this, random failures (and possibly repairs) are added to the original simulation model, which is slightly modified in order to propagate the effects of failures. The article by Bouissou et al. (2014) explains how standard Modelica solvers can be used to simulate systems with failure rates that depend on continuous variables (like temperature, etc.). This was tested with a well-known benchmark that was first published in (Aldemir, 1987) and since then has been solved with many different methods and tools.

In the OpenModelica setting, scripting is used to run the simulation a large number of times (10 000 times in this example), which allows to calculate by simple statistical estimators the probability of various undesirable events over time, and quantities such as average production, life-cycle cost etc.

4.18 Data Reconciliation for Enhanced Accuracy of Sensor Data

The operation of power plants requires a good quality of the process data obtained through sensor measurements. Unfortunately, sensor measurements such as flow rates, temperatures, and pressures are subject to errors that lead to uncertainties in the assessment of the system's state. Operational margins are therefore set to take into account uncertainties on the system's state. Their effect is to decrease production because safety regulations put stringent limits on the thermal power of the plants. It is therefore important to compute the best estimates of the measurement uncertainties in order to increase power production by reducing operational margins as much as possible while still preserving system safety.

The best estimates can be obtained by combining data statistics with a knowledge a priori of the system in the form of a physical model. Data reconciliation is a technique that has been conceived in the process industry for that purpose. It is fully described in the VDI 2048 standard (VDI – Verein Deutscher Ingenieure, 2012, 2017, 2018) for the "Control and quality improvement of process data and their uncertainties by means of correction calculation for operation and acceptance tests". Up to now, it was only available in dedicated tools such as VALI from Belsim (2019) that require to develop a specific model of the system under consideration. The main drawbacks are that such models are costly to develop and difficult to validate. A natural answer to this problem is to perform data reconciliation on Modelica models. However, under the current state-of-the-art of Modelica tools, such task is not possible because an appropriate subset of the Modelica model of the system under consideration must be considered for data reconciliation. This subset contains the (presumably) exact physical laws that constrain the variables of interest to be reconciled. All other equations such as boundary conditions or approximated equations that affect the variables of interest

must be removed. This subset is called the "auxiliary conditions" in VDI 2048. The auxiliary conditions are therefore underdetermined (more unknowns than equations) and cannot form a valid Modelica model for simulation.

OpenModelica is currently being extended to perform data reconciliation on regular Modelica models. To that end the following is developed:

- New annotations are introduced to tag the variables of interest and the approximated equations.
- An algorithm is being developed to automatically extract the auxiliary conditions.
- The data reconciliation procedure taking as inputs the variables of interest as tagged by the user, their measured values (mean values and statistical confidence intervals) as provided by the user and the automatically extracted auxiliary conditions from the Modelica model provided by the user, is being implemented. It produces as outputs the reconciled values and their reduced confidence intervals.
- The GUI is being extended to handle inputs and outputs.

The main benefit is to be able to perform data reconciliation on existing validated model without having to modify them for that purpose. This is a considerable improvement with respect to the current state-of-the-art.

4.19 Artificial Neural Networks for Model Calibration and Augmentation

The pursuit of both model accuracy and simplicity in general results in conflict. Ultimately, a model should meet its accuracy requirements whilst remaining as simple as possible. To validate a model, measurements taken from the original system can be used (Zhu et al., 2007). As an example, the trajectories from simulations of the model can be compared to the actual measurement data to validate the model behavior.

A more sophisticated approach may involve the extraction of high-level features from said trajectories. For example, a good model of a pendulum should exhibit frequency and amplitude of oscillation similar to those of the original pendulum.

Inherently, after the modeling process, there are differences between the measurements of the model and the original system that, by magnitude or trend over time, cannot purely originate from measurement noise. Assuming negligible numerical error, parameter errors or model errors must be assumed. The former can be eliminated by choosing accurate physical parameters.

Model errors, however, cannot be completely avoided and the question arises how their impact can be reduced. In practice it is usually not possible to distinguish between parameter and modeling errors and hence during model calibration often physically "wrong" parameters compensate for modeling errors. This can be seen as a motivation for grey-box modeling, where reference measurements are not only used for model calibration, but to augment the first principles model (Modelica model) with a data-based model. That is, specific relations within a model shall be learned on the basis of reference data while keeping physically established relations unchanged. This is different from pure black box modeling (Mohajerin et al., 2018). The idea of localized adaption of single equations is especially applicable to object-oriented modeling (like in Modelica) and aims at keeping learning results understandable to the user by separation from existing "white box" relations.

In order to evaluate an artificial neural network (ANN) grey-box modeling approach in OpenModelica, a framework for the training of Tensorflow machine learning models (Tensorflow.org, 2019) as shown in Figure 25 is set up. The simulation data generated in OpenModelica using the reference model is used to train the ANN. This is achieved with the help of the Python modules OMPython (Lie et al., 2016), numpy, and Tensorflow. The integration of Tensorflow models in a Modelica model is done using the "external C" interface of Modelica.

The approach by Bruder and Mikelsons (2019) has been tested using a dynamic system model of a motorcycle (from the Planar Mechanics library (Zimmer, 2012) extended by a simple driver model. Particular mathematical relations (drag force, dynamical wheel loads and tire forces) therein were learned by feedforward ANNs which are trained using simulation data generated from the original model driving specific maneuvers (eight shaped trajectories with and without acceleration). These machine-learnt relations were then used to replace the original relations in new grey-box motorcycle models.

This emulates a situation in which the original relations (e.g. those of a real system) are unknown but measurements indicate an interdependence between variables. The resulting grey-box model is then simulated in order to validate it. The validation maneuver is a double lane change and the simulation results are shown in Figure 26. It can be seen that learning the drag force and dynamical wheel loads worked out quite well, while the learned tire force model is not applicable at all.

Moreover, the extrapolation quality of the data-based models in this example is investigated, e.g. using

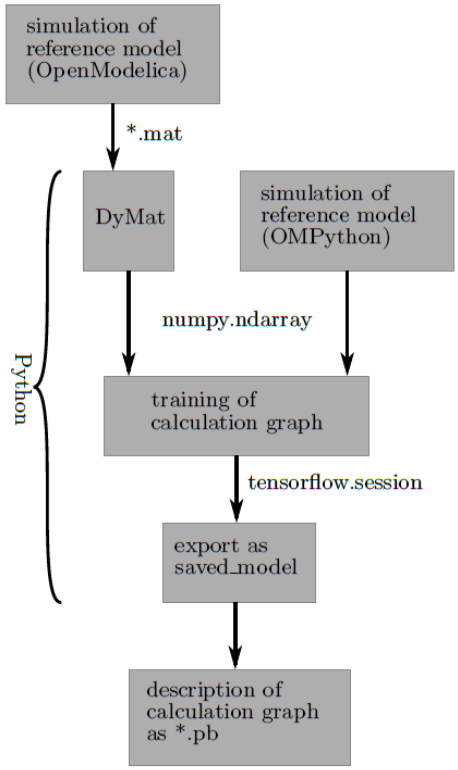


Figure 25: A framework for training Tensorflow models.

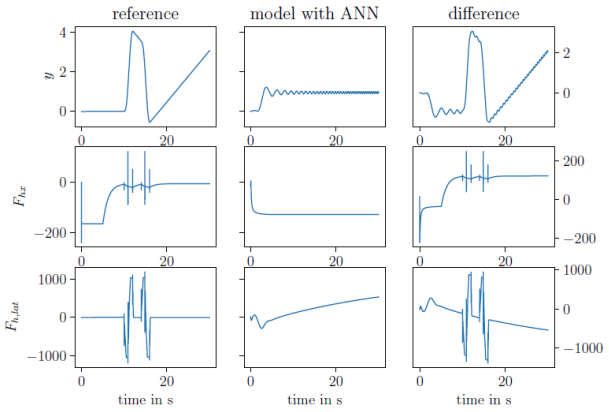
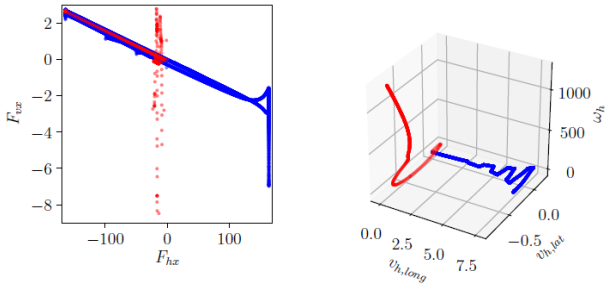


Figure 26: Virtual measurement data of both the reference model and the model with wheel joints including the ANN (artificial neural network). The data was generated using OpenModelica and plotted using matplotlib.

the plots of the training data and requested data points shown in Figure 27.



(a) dynamic loads (b) wheel joints

Figure 27: Inputs to ANN. Blue: learning set. Red: inputs during simulation of the double lane change.

4.20 Embedded System Support

OpenModelica provides code generation of real-time controllers from Modelica models, for small foot-print platforms such as Arduino boards or in tools for RexRoth PLCs (Menager et al., 2014).

One example of code generation to small targets is the Single board heating system (Figure 28) from IIT Bombay (Arora et al., 2010). It is used for teaching basic control theory, and usually controlled by a serial port (set fan value, read temperature, etc.). OpenModelica can generate code targeting the ATmega16 on the board (and other AVR microcontrollers (Thiele et al., 2017) or STM32F4 (Berger et al., 2017)).

The program size is 4090 bytes including LCD driver and PID-controller (out of 16 kB flash memory available). The ATmega16 we target has 1 kB SRAM available for data (stack, heap, and global variables). In this case, only 130 bytes is used for data variables.

To simplify interfacing of low-level devices from Modelica, OpenModelica supports the Modelica_DeviceDrivers library (Thiele et al., 2017), which is a free library for interfacing hardware drivers that is developed primarily for interactive real-time simulations. The library is cross-platform (Windows and Linux). Using this library, modeling, parameterization and configuration can be done at a high level of abstraction using Modelica, avoiding the need for low level C programming. Another example using the embedded system support is the Arduino controlled electromagnetic levitation system depicted in Figure 29. The application is based on a commercially available electromagnetic levitation kit by Zeltom LLC (2019), which is targeted at educational applications. The controller design is described in Thiele et al. (2019) and uses additional OpenModelica technologies like the interactive Julia scripting (Section 4.9) and the synchronous language elements extension (Section 4.15.1).

Figure 28: The SBHS (Single Board Heating System), an example embedded target system for OpenModelica.

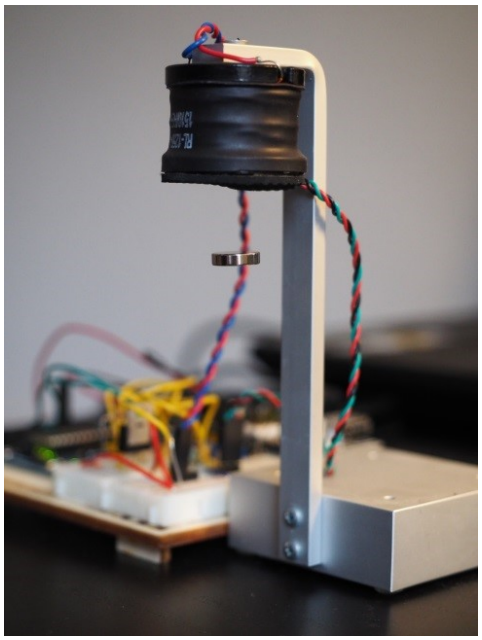


Figure 29: Arduino controlled electromagnetic levitation system.

4.21 MDT Eclipse Plug-in

The MDT (Modelica Development Tooling) Eclipse plug-in (Figure 30) (Pop et al., 2006; Pop, 2008), is an Eclipse-based textual development environment for Modelica and MetaModelica model development.

It provides the usual facilities for software development such as browsing, building, cross referencing, syntax checking, and showing useful information such as types, function signatures, etc.

MDT is primarily used for development of medium to large scale Modelica projects, such as Modelica libraries written in standard Modelica and the OpenModelica compiler (currently containing more than 200 packages) written in MetaModelica.

4.22 ModelicaML UML Profile and Eclipse Plug-in

ModelicaML (Figure 31), (Schamai, 2013; Schamai et al., 2014) is an Eclipse plug-in and Modelica-UML profile for the description of system architecture and system dynamic behavior. It is based on an extended subset of the OMG Unified Modeling Language (UML) as well as Modelica, and is designed for Modelica code generation from graphical models such as state machines and activity diagrams, supporting hardware/-software co-modeling and system requirement verification against selected scenarios. The current prototype has not been updated recently and only works together with an old version of Eclipse.

4.23 Verification of Designs against Requirements using Simulation

Mastering the development of today's complex systems requires a structured approach called Systems Engineering. One of the activities involved is design verifi-

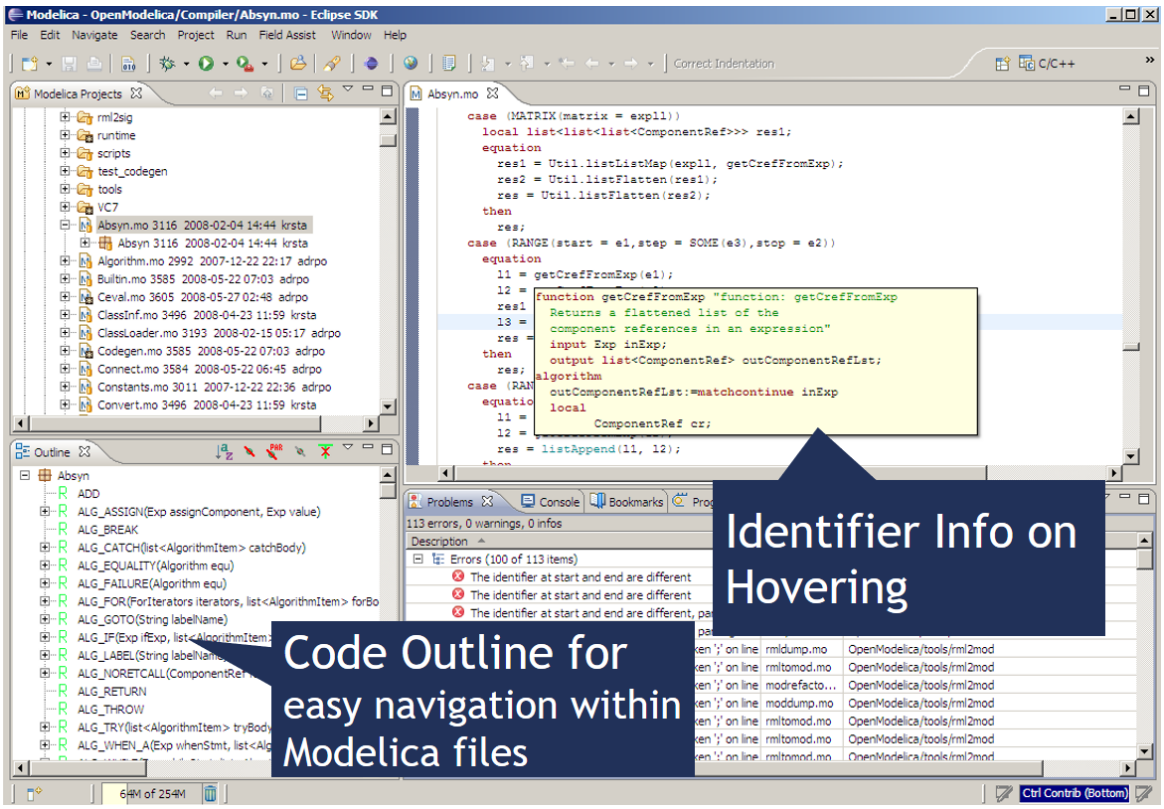


Figure 30: The OpenModelica MDT (Modelica Development Tooling) Eclipse plug-in.

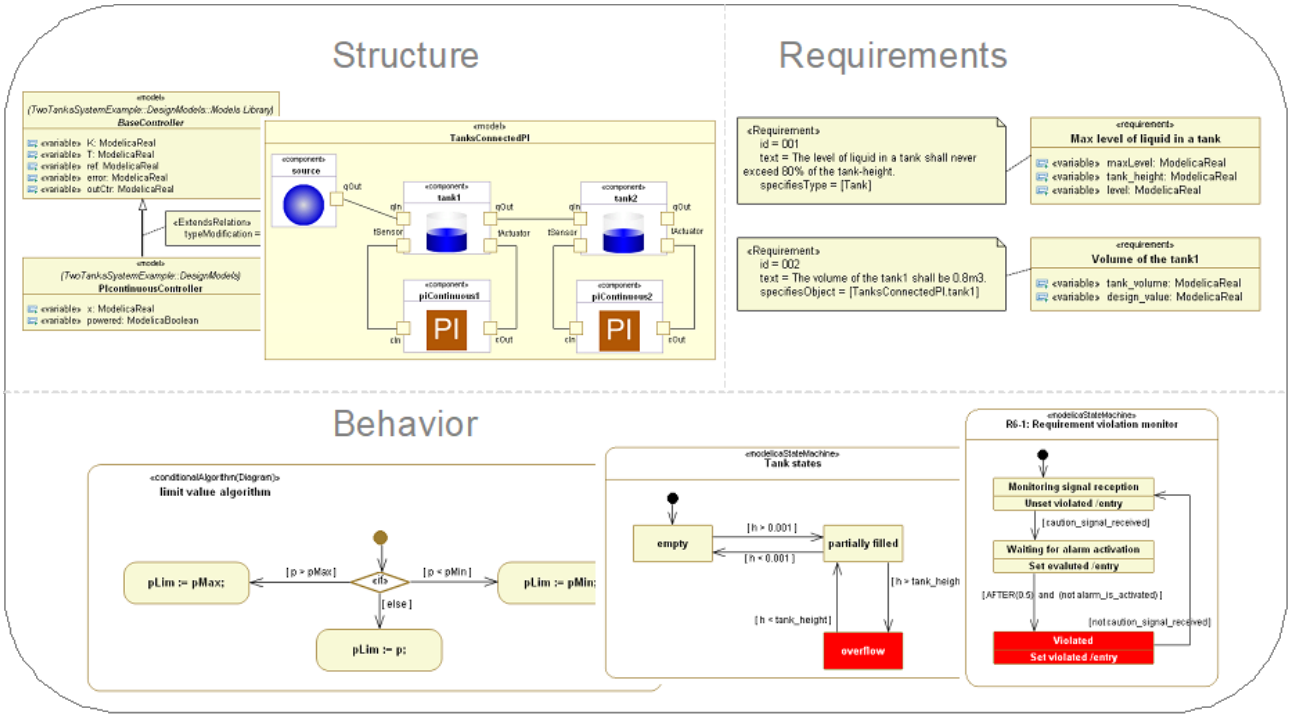


Figure 31: The ModelicaML Eclipse plug-in and UML-Modelica profile for integrated software-hardware modeling and requirements verification.

cation is to determine if a given design meets a set of specified requirements. Designs are often modeled and can then be simulated. In contrast, requirements are typically expressed in natural language to serve for better communication between different stakeholders involved. The drawback of using natural language (e.g., English) is that it may make the requirement specification prone to human errors and ambiguity.

To address these challenges, a modeling approach called vVDR (virtual Verification of Designs vs. Requirements) was developed that allows formalizing requirements and creating executable Modelica models, called requirement monitors, for each requirement statement (Schamai, 2013; Schamai et al., 2015; Otter et al., 2015). Once connected to executable design models, requirement monitors show the status of the requirement violation with at least three literals: not applicable, not violated, violated, at any simulated time instant, as well as the accumulated status (i.e., has been tested, has been violated, etc.).

Furthermore, the vVDR modeling approach creates separate models for scenarios that can be used for testing different requirements. For this to be efficient it includes a way to automatically compose executable models each including the design to be tested, the scenario to be used, and the relevant requirement monitors. This is enabled by the binding concept and an algorithm that iterates over design alternatives and all available scenarios, and uses semantic equivalents of requirement monitor inputs/outputs to identify monitor models to be included (Schamai, 2013; Schamai et al., 2014).

The approach has been tested successfully in several case studies, including at Airbus (Schamai et al., 2015), Scania (Liang et al., 2012), Electricité du France (Schamai et al., 2014), and the Swedish Road and Traffic Institute (Andersson and Buffoni, 2018). OpenModelica was used as the prototyping environment when running the case studies.

The vVDR approach was first made available in the OpenModelica ModelicaML Eclipse plug-in mentioned in Section 4.22, using a combination of UML and Modelica for formal requirement specification. Later, a Modelica-only version of vVDR has been designed and implemented in OpenModelica using OMEdit as user interface, requirement specification in Modelica, and a vVDR Modelica library (Buffoni and Fritzson, 2015; Buffoni et al., 2017; Buffoni, 2019). That library enables defining binding information which is processed by the binding algorithm implemented in OpenModelica.

As mentioned, the vVDR simulation-based approach can be used to verify (Figure 32) design alternatives against sets of requirements using different scenarios.

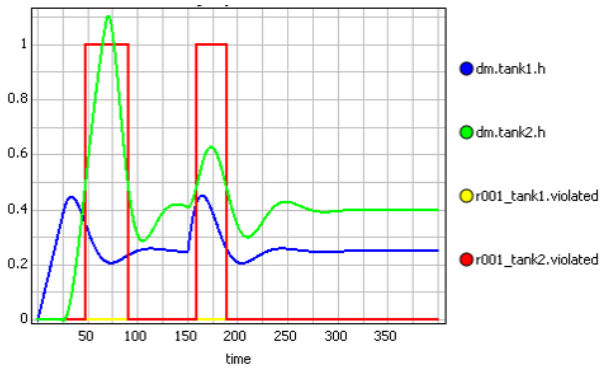


Figure 32: Simulation-based requirement verification for the two-tanks example. Requirement r001 regarding the level of tank2 is violated twice (shown in red).

The tool automatically generates verification models in Modelica, performs the simulations, compares the results, and generates a report about verification results.

4.24 Parallelization and Multi-Core

Work on generating parallel code from Modelica models has been ongoing for OpenModelica during several years. Both automatic and explicit parallelization approaches have been investigated and implemented.

Automatic parallelization of simulation code for Modelica models has been investigated in different contexts and perspectives (Aronsson, 2006; Walther et al., 2014; Gebremedhin and Fritzson, 2017). These parallelization approaches attempt to automatically detect, extract and utilize potential parallelism in the equation systems generated from Modelica models.

Two recent approaches, hpcom (Walther et al., 2014) and parmodauto (Gebremedhin and Fritzson, 2017; Gebremedhin, 2019), are similar in the sense that they both utilize equation level processing of strongly connected components of large equation systems. The hpcom parallelization approach utilize a semi-static cost estimation approach based on previous execution history of a given model to effectively schedule and load balance large simulation executions. This has the advantage that there will be a very small overhead on the execution of a given simulation. However, it also means that simulations cannot effectively respond to changes in computational load and behavior during one simulation run. On the other hand, the parmodauto approach utilizes a runtime profiling and scheduling approach where each simulation run is monitored and load-balanced dynamically at runtime. This approach has two main advantages. The first is that no prior information is needed. In addition, it is also able to respond to systems with simulation of dynamic behav-

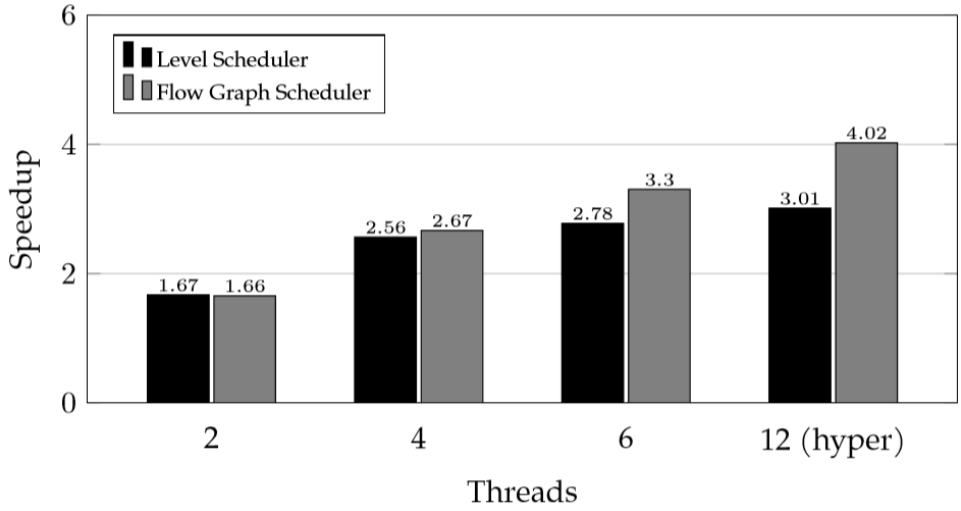


Figure 33: Example of speedup of parallel code from OpenModelica for the BranchingDynamicPipes model.

ior. However, this also means that there is an additional overhead involved in performing the monitoring and profiling of simulation executions.

An explicit parallelization approach and language extension, ParModelica (Gebremedhin, 2011; Gebremedhin et al., 2012; Gebremedhin, 2019), based on the OpenCL (2018) framework targeting shared memory multi-core processors is also currently available in OpenModelica. The explicit parallelization brings support for expressing parallelization directly in a Modelica model using language constructs that are partially based on familiar General-Purpose Graphics Processing Unit (GPGPU) frameworks such as CUDA (Nvidia, 2008) and OpenCL. The Modelica language is extended with constructs such as parallel for-loops and parallel functions among others. These constructs can be utilized to write an explicit parallel program in the algorithmic parts of a Modelica model. The OpenModelica compiler analyzes these constructs and generates OpenCL code that can be executed on general purpose CPUs, GPUs and accelerators without requiring any change to the original Modelica source code.

5 Selected Open Source Modeling and Simulation Applications

In the following a few open source and/or crowd-sourced applications of OpenModelica are briefly presented.

5.1 Process Modeling Using Extended Petri Nets in Modelica

Process modeling is not the most common application for Modelica modeling. Fortunately, the open source PNLib library (Proß and Bachmann, 2012) has been developed in Modelica to support the xHPN (extended Hybrid Petri Net) formalism. This formalism supports modeling of processes that can combine elements which are stochastic, deterministic, discrete, and continuous which gives very powerful modeling capabilities. The library was later extended and generalized to version 2.0 and tool support in OpenModelica was implemented by Ochel (2017), including applications to biological processes. Figure 34 illustrates a restaurant process model with customers arriving stochastically, with ordering, waiting, serving, and eating.

5.2 Examples of Crowd-Sourced Applications with OpenModelica

One of the benefits of OpenModelica being open source software is that it is possible to engage the community to contribute to appropriate content generation. An example of this is already provided in Section 4.8, wherein the Textbook Companion effort is explained. In this section, we describe a few crowd-sourced simulation activities using OpenModelica.

Very few open source chemical process simulators are available to the community. The situation is worse when it comes to general purpose dynamic simulation of chemical processes. A prerequisite for this is the ability to simultaneously solve all equations that make up the flowsheet or the circuit. As there could be tens of thousands of nonlinear equations in such problems,

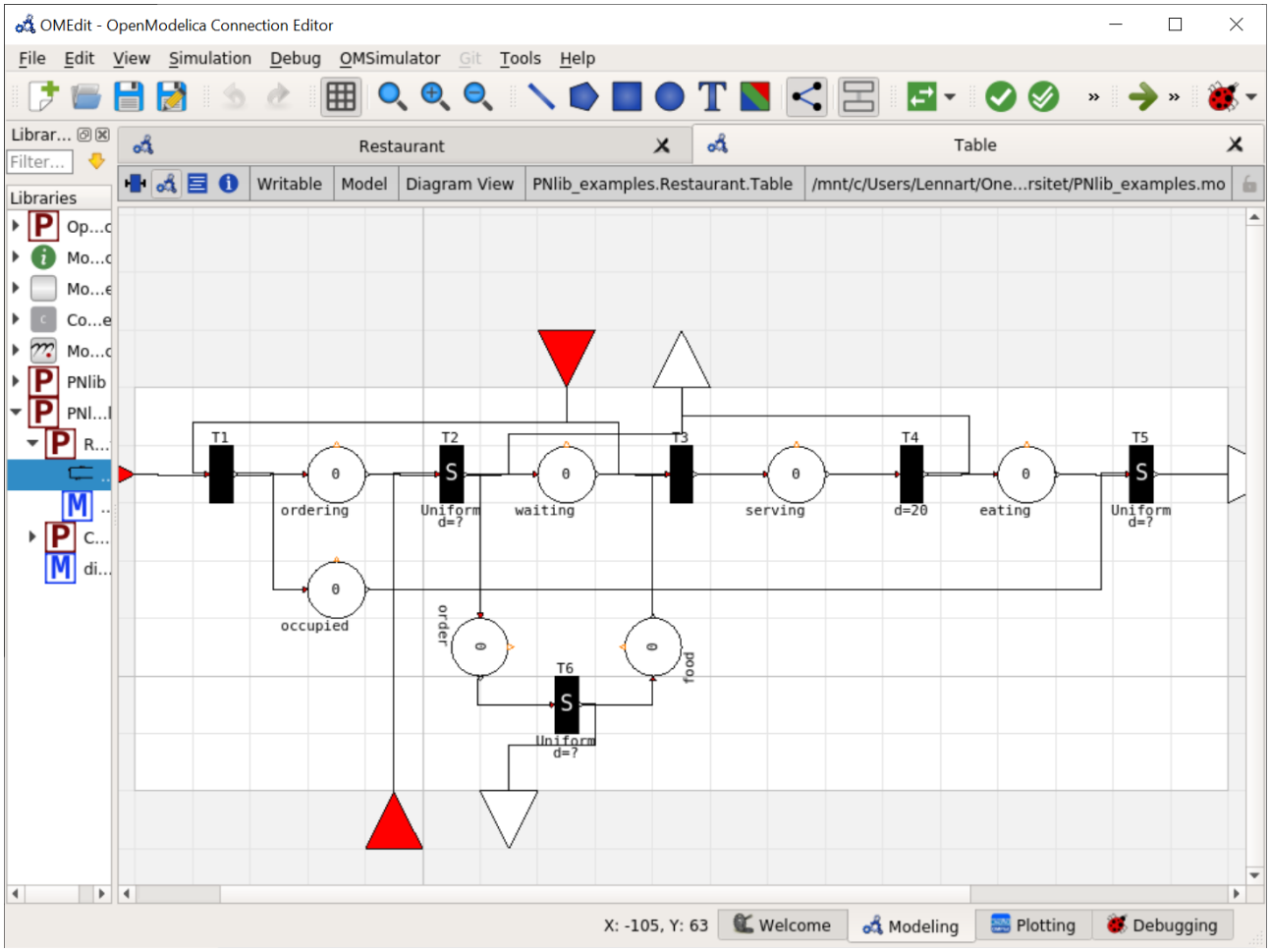


Figure 34: A restaurant process model using PNLib in OpenModelica, with ordering, waiting, serving, eating.

unless proper tools are made available, it is difficult to get contributions from the community.

In order to make it convenient for the community to contribute, a library of thermodynamic models has been made available. This was achieved by a port of thermodynamics to Modelica, including component data and correlations for the calculation of properties (Jain et al., 2019; FOSSEE-OMChemSim, 2020).

Models of chemical process unit operations, building blocks of chemical engineering operations, have been created using OpenModelica. Using these and a thermodynamics library models available in OpenModelica, chemical process flowsheets have been created (Nayak et al., 2019; FOSSEE-Flowsheets, 2020). A schematic of a sample flowsheet is given in Figure 35.

With the above mentioned tools, it has become convenient for the community to create chemical process simulations and offer them as open source. A total of more than 50 chemical process flowsheets solved using OpenModelica are now available (FOSSEE-Flowsheets, 2020) and many more are in progress. Given that simultaneous solution of thousands of equations is a difficult task, training a large number of engineers on this important technology would have been impossible without an open source simulator such as OpenModelica.

Can the above approach be extended to another discipline? We explored applying the same principles to power system simulation. Fortunately, a power system simulation library OpenIPSL (Baudette et al., 2018) is already available. Students have contributed 35 power system simulation models (FOSSEE-Power, 2020), a sample of which is shown in Figure 36[1].

6 Related Work

Since OpenModelica is a Modelica environment it has of course been influenced by other Modelica tools. The

[1] Another open source library for power systems with a more modern design called PowerGrids (Casella and Guironnet, 2020), has recently become available.

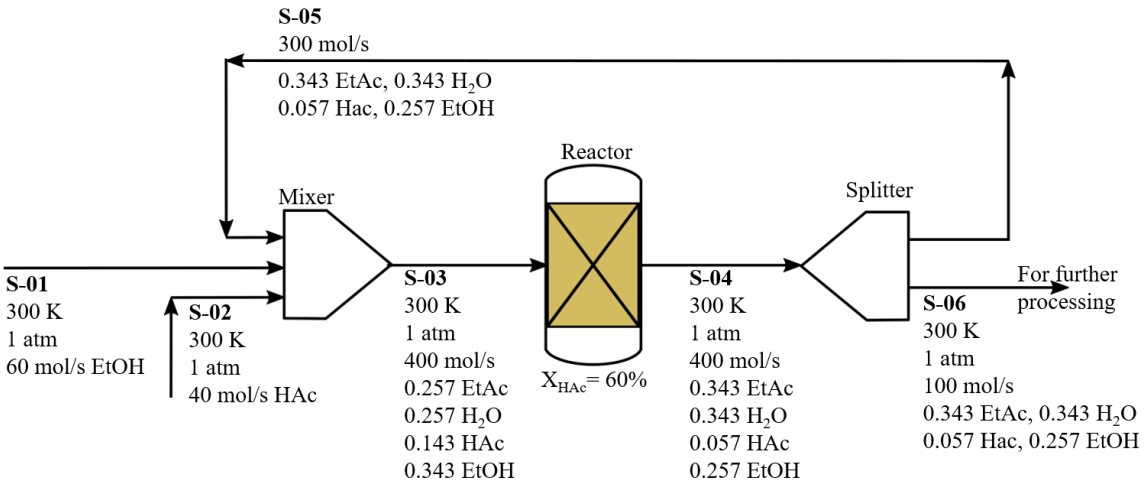


Figure 35: Example flowsheet for Acetic acid esterification by ethanol.

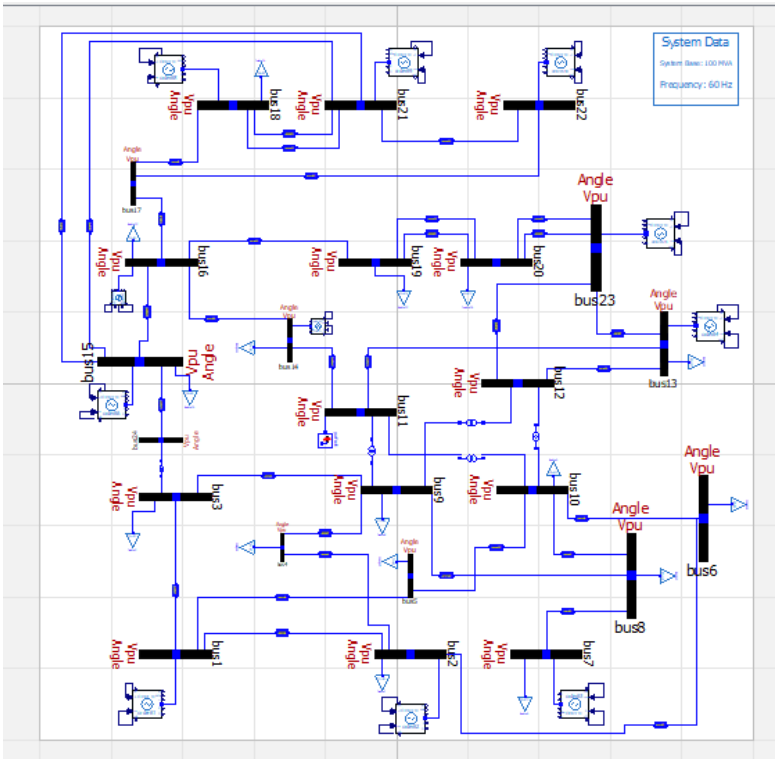


Figure 36: Modeling and simulation of IEEE 24 Bus system using OpenModelica and the OpenIPSL library.

most influential of these tools is Dymola (Elmqvist et al., 1996; Brück et al., 2002; Dassault Systèmes, 2018), which was the first full-scale industrial-strength Modelica environment. Certain aspects have also been influenced by the MathModelica environment (Fritzson, 2006), later renamed and further developed to Wolfram System Modeler (Wolfram Research, 2018). The systems InterLisp (Teitelman, 1974), Mathematica (Wolfram, 2003), and ObjectMath (Fritzson et al., 1995) have influenced the design of OpenModelica as an integrated symbolic-numeric environment. Recently, the rapidly developing symbolic-numeric Julia language (Bezanson et al., 2017; Julialang, 2018) has appeared, with similar goals as MetaModelica regarding integration and efficient execution of both symbolic and numeric operations.

7 Conclusion

OpenModelica has been developed into a powerful open source tool suite for modeling, simulation, and model-based development. It is a unique effort that provides a workbench for research on integration and development of methods, tools and scientific knowledge in an open source setting. Still some challenges are being worked on and remain to be addressed, for example very large models with several million equations. The debugger can be further improved to provide high-level, user-friendly diagnostic messages to help the user resolve run-time numerical errors, a difficult task particularly for novice users. Recently new methods such as data reconciliation and usage of the machine learning TensorFlow framework for model calibration have been integrated. There is room for more such efforts. Integration aspects between tool functionalities can be further enhanced. Just-in-time compilation would improve the system's interactive properties. Two large recent OpenModelica efforts briefly described in this article are the OMC new frontend development for 100% compilation coverage and greatly enhanced compilation speed, and the OMSimulator tool for efficient large-scale FMI-based simulation. A new effort has just been started on designing and implementing an improved compiler backend with enhanced scalable symbolic algorithms to be able to handle very large models. Recently OMJulia has been introduced that provides OpenModelica access from Julia. More powerful integration options between Julia and OpenModelica are also being considered in order to benefit from the Julia libraries and infrastructure.

Acknowledgment

This work has been supported by Vinnova in the ITEA OPENPROD, MODRIO, OPENCPS, EMPHYSIS and EMBRACE projects and in the Vinnova RTISIM and EMISYS projects. Support from the Swedish Government has been received from the ELLIIT project. Support has also been received from the Swedish Strategic Research foundation (SSF) in the LargeDyn project. The OpenModelica development is supported by the Open Source Modelica Consortium. Many students, researchers, engineers have contributed to the OpenModelica system – there is not room here to mention all these people, but we gratefully acknowledge their contributions. The development of Spoken Tutorials and text-book companions was funded by the Ministry of Education, Govt. of India, through a grant given to FOSSEE and Spoken Tutorial projects.

References

Sameer Agarwal, Keir Mierle, and Others. Ceres solver, 2018. URL http://ceres-solver.org. Note: Accessed 2018.

Giovanni Agosta, Emanuele Baldino, Francesco Casella, Stefano Cherubin, Alberto Leva, and Federico Terraneo. Towards a High-Performance Modelica Compiler. In *Proc. of the 13th International Modelica Conference*, Regensburg, Germany, March 2019. doi:10.3384/ecp19157313.

Johan Åkesson. Optimica—An Extension of Modelica Supporting Dynamic Optimization. In *Proc. of the 6th International Modelica Conference*, Bielefeld, Germany, March 2008.

Tunc Aldemir. Computer-Assisted Markov Failure Modeling of Process Control Systems. *IEEE Transactions on Reliability*, 36(4):133–144, 1987. doi:10.1109/tr.1987.5222318.

Edward Anderson, Zhaojun Bai, Christian Heinrich Bischof, Laura Susan Blackford, James Weldon Demmel, J. Dongarra, Jeremy J. Du Croz, Anne Greenbaum, Sven Hammarling, Alan M. McKenney, and Danny C. Sorensen. *Lapack Users' Guide 3rd Edition*. SIAM, August 1999. URL http://www.netlib.org/lapack/lug/.

Anders Andersson and Lena Buffoni. Powertrain Model Assessment for Different Driving Tasks through Requirement Verification. In *Proc. of The 9th EUROSIM Congress on Modelling and Simulation*, pages 721–727, 2018. doi:10.3384/ecp17142721.

Peter Aronsson. *Automatic Parallelization of Equation-Based Simulation Programs*. PhD thesis, Linköping University, Department of Computer and Information Science, June 2006. URL http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Aliu%3Adiva-7446.

Inderpreet Arora, Kannan Moudgalya, and Sachitanand Malewar. A low cost, open source, single board heater system. In *Proc. 4th IEEE International Conference on E-Learning in Industrial Electronics*. IEEE, November 2010. doi:10.1109/icelie.2010.5669868.

Adeel Asghar, Sonia Tariq, Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, Martin Sjölund, Parham Vasaiely, and Wladimir Schamai. An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation. In *Proc. from the 8th International Modelica Conference*, Dresden, Germany, March 2011. doi:10.3384/ecp11063739.

Mikael Axin, Robert Braun, Alessandro Dell'Amico, Björn Eriksson, Peter Nordin, Karl Pettersson, Ingo Staack, and Petter Krus. Next Generation Simulation Software using Transmission Line Elements. In *Proc. of Fluid Power and Motion Control*, pages 265–276. Centre for Power Transmission and Motion Control, 2010. ISBN 978-1-86197-181-4.

Bernhard Bachmann, Lennart Ochel, Vitalij Ruge, Mahder Gebremedhin, Peter Fritzson, Vaheed Nezhadali, Lars Eriksson, and Martin Sivertsson. Parallel Multiple-Shooting and Collocation Optimization with OpenModelica. In *Proc. of the 9th International Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp12076659.

Bernhard Bachmann, Kaja Balzereit, Willi Braun, Jan Hagemann, Lennart Ochel, Vitalij Ruge, and Patrick-Marcel Täuber. Symbolical and Numerical Approaches for Solving Nonlinear Systems, February 2015. URL https://www.openmodelica.org/images/docs/openmodelica2015/OpenModelica2015-talk04-Bernhard-Bachmann-NLSinOpenModelica.pdf. Annual OpenModelica Workshop, Linköping University, Linköping, Sweden.

Maxime Baudette, Marcelo Castro, Tin Rabuzin, Jan Lavenius, Tetiana Bogodorova, and Luigi Vanfretti. OpenIPSL: Open-Instance Power System Library—Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". *SoftwareX*, 7: 34–36, 2018. doi:10.1016/j.softx.2018.01.002. URL https://github.com/ElsevierSoftwareX/SOFTX-D-17-00098.

Belsim. Vali tool, 2019. URL http://www.belsim.com/business/solution/vali-energy/. Accessed: October, 2019.

Lutz Berger, Martin Sjölund, and Bernhard Thiele. Code generation for STM32F4 boards with Modelica device drivers. In *Proc. of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT'17)*, Munich, Germany, December 2017. ACM Digital Library. doi:10.1145/3158191.3158204.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A Fresh Approach to Numerical Computing. *SIAM Review*, 59(1):65–98, 2017. doi:10.1137/141000671.

Marc Bouissou, Henri Bouhadana, Marc Bannelier, and Nathalie Villatte. Knowledge Modelling and Reliability Processing: Presentation of the Figaro Language and Associated Tools. *IFAC Proceedings Volumes*, 24(13):69–75, October 1991. doi:10.1016/s1474-6670(17)51368-3.

Marc Bouissou, Hilding Elmqvist, Martin Otter, and Albert Benveniste. Efficient Monte Carlo Simulation of Stochastic Hybrid Systems. In *Proc. of the 10th International Modelica Conference*, Lund, Sweden, March 2014. doi:10.3384/ecp14096715.

Marc Bouissou, Lena Buffoni, and Bernhard Thiele. From Design to Dependability: A Bridge Between Physical Simulation and Risk Analysis. In *Proc. of Lambda-mu 20*, Saint Malo, October 2016. doi:10.4267/2042/61810.

Willi Braun, Stephanie Gallardo-Yances, Kilian Link, and Bernhard Bachmann. Fast Simulation of Fluid Models with Colored Jacobians. In *Proc. of the 9th International Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp12076247.

Willi Braun, Francesco Casella, and Bernhard Bachmann. Solving Large-scale Modelica Models: New Approaches and Experimental Results using OpenModelica. In *Proc. of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132557.

Dag Brück, Hilding Elmqvist, Sven-Erik Mattsson, and Hans Olsson. Dymola for Multi-Engineering Modeling and Simulation. In *Proc. of the 2nd International Modelica Conference*, Oberpfaffenhofen, Germany, March 2002.

Frederic Bruder and Lars Mikelsons. Towards Grey Box Modeling in Modelica. *Robotics and Mechatronics. ISRM 2019. Mechanisms and Machine Science*, 78, 2019. doi:10.1007/978-3-030-30036-4_17.

Lena Buffoni. VVDRlib, 2019. URL https://github.com/lenaRB/VVDRlib. Accessed: December, 2019.

Lena Buffoni and Peter Fritzson. Expressing Requirements in Modelica. In *Proc. of the 55th Scandinavian Conference on Simulation and Modeling (SIMS'2014)*, Aalborg, Denmark, October 2015. doi:10.11128/sne.25.tn.10314.

Lena Buffoni, Adrian Pop, and Alachew Mengist. Traceability and Impact Analysis in Requirement Verification. In *Proc. of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT'17)*, Munich, Germany, December 2017. ACM Digital Library. doi:10.1145/3158191.3158207.

Francesco Casella. Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspectives. In *Proc. of the 11th International Modelica Conference*, Versailles, France, September 2015. doi:10.3384/ecp15118459.

Francesco Casella and Adrien Guironnet. Tutorial, Modelling and Simulation of Power Systems with OpenModelica and the PowerGrids library, February 2020. URL www.modprod.se. MODPROD Workshop on Model-Based Cyber-physical Product Development, https://github.com/PowerGrids/PowerGrids, Linköping Universitet, Linköping, Sweden.

Francesco Casella, Filippo Donida, and Johan Åkesson. Object-Oriented Modeling and Optimal Control: A Case Study in Power Plant Start-Up. *IFAC Proceedings Volumes*, 44(1):9549–9554, January 2011a. doi:10.3182/20110828-6-it-1002.03229.

Francesco Casella, Michael Sielemann, and Luca Savoldelli. Steady-State Initialization of Object-Oriented Thermo-Fluid Models by Homotopy Methods. In *Proc. of the 8th International Modelica Conference*, Dresden, Germany, June 2011b. doi:10.3384/ecp1106386.

Rodrigo Castro. Arguments on the Imminence of Global Collapse Are Premature when Based on Simulation Models. *GAIA - Ecological Perspectives for Science and Society*, 21(4):271–273, 2012. doi:10.14512/gaia.21.4.9.

François Cellier. World3 in Modelica: Creating System Dynamics Models in the Modelica Framework. In *Proc. of the 6th International Modelica Conference*, Bielefeld, Germany, March 2008.

François Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer Science & Business Media, Berlin Heidelberg, 2006. ISBN 978-0-387-26102-7.

Alejandro Danós, Willi Braun, Peter Fritzson, Adrian Pop, Hugo Scolnik, and Rodrigo Castro. Towards an OpenModelica-Based Sensitivity Analysis Platform Including Optimization-Driven Strategies. In *Proceedings of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT '17, page 87–93, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450363730. doi:10.1145/3158191.3158206.

Dassault Systèmes. Dymola. systems engineering overview, 2018. URL https://www.3ds.com/products-services/catia/products/dymola/. Accessed: September, 2018.

Timothy Davis. Algorithm 832: UMFPACK V4.3—An Unsymmetric-Pattern Multifrontal Method. *ACM Transactions on Mathematical Software (TOMS)*, 30 (2):196–199, June 2004. doi:10.1145/992200.992206.

John E. Dennis Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, 1996. ISBN 978-1-611-97120-0.

John E. Dennis Jr., Nélia Echebest, M. T. Guardarucci, José Mario Martínez, Hugo D. Scolnik, and M. C. Vacchino. A Curvilinear Search Using Tridiagonal Secant Updates for Unconstrained Optimization. *SIAM Journal on Optimization*, 1(3):333–357, 1991. doi:10.1137/0801022.

Iain S. Duff, Albert M. Erisman, and John K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, 2017. doi:10.1093/acprof:oso/9780198508380.001.0001.

Hilding Elmqvist, Dag Brück, and Martin Otter. Dymola—User's Manual, 1996.

Hilding Elmqvist, Toivo Henningsson, and Martin Otter. Innovations for Future Modelica. In *Proc. of the 12th International Modelica Conference*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132693.

Atiyah Mohamed Gamal Elsheikh. *Modelica based computational tools for sensitivity analysis via automatic differentiation*. PhD thesis,

Aachen, 2012. URL https://publications.rwth-aachen.de/record/229154. Prüfungsjahr: 2012. - Publikationsjahr: 2014; Aachen, Techn. Hochsch., Diss., 2012.

Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, and Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In *Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done?*, Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 2006. URL www.openmodelica.org.

FOSSEE-Flowsheets. FOSSEE OpenModelica Team. Flowsheets in OpenModelica, 2020. URL https://om.fossee.in/chemical/flowsheeting-project/completed-flowsheet. Accessed: September, 2020.

FOSSEE-Modelica. Step-by-step Audio-Video tutorials on how to use Modelica and OpenModelica, 2020. URL https://spoken-tutorial.org/tutorial-search/?search_foss=OpenModelica&search_language=English. Accessed: 2020-09-11.

FOSSEE-OM-Textbook. FOSSEE OpenModelica Team. Crowdsourced OpenModelica textbook companions, 2020. URL https://om.fossee.in/textbook-companion/completed-books. Accessed: September, 2020.

FOSSEE-OMChemSim. FOSSEE OpenModelica Team. GitHub link to FOSSEE OpenModelica Chemical Engineering Code, 2020. URL https://github.com/FOSSEE/OMChemSim. Accessed: September, 2020.

FOSSEE-Power. FOSSEE OpenModelica Team. Power System Simulation in OpenModelica, 2020. URL https://om.fossee.in/powersystems/pssp/completed-pssp. Accessed: 2020-09-11.

Rüdiger Franke, Manfred Rode, and Klaus Krüger. On-line Optimization of Drum Boiler Startup. In *Proc. of the 3rd International Modelica Conference*, Linköping, Sweden, November 2003. URL https://www.modelica.org/events/Conference2003/index_html/papers/h29_Franke.pdf.

Rüdiger Franke, Marcus Walther, Niklas Worschech, Willi Braun, and Bernhard Bachmann. Model-based Control with FMI and a C++ Runtime for Modelica. In *Proc. of the 11th International Modelica Conference*, Versailles, France, September 2015. doi:10.3384/ecp15118339.

Rüdiger Franke, Sven Erik Mattsson, Martin Otter, Karl Wernersson, Hans Olsson, Lennart Ochel, and Torsten Blochwitz. Discrete-time Models for Control Applications with FMI. In *Proc. of the 12th International Modelica Conference*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132507.

Dag Fritzson. Transient conformal TEHL algorithms for multibody simulation. *Modeling, Identification and Control*, 39(3):209–232, 2018. doi:10.4173/mic.2018.3.6.

Dag Fritzson, Lars-Erik Stacke, and Jens Anders. Dynamic simulation — Building knowledge in product development. *SKF Evolution*, 1(1):21–26, 2014. URL http://evolution.skf.com/dynamic-simulation-building-knowledge-in-product-development/.

Dag Fritzson, Robert Braun, and Jan Hartford. Composite modelling in 3-D mechanics utilizing Transmission Line Modelling (TLM) and Functional Mock-up Interface (FMI). *Modeling, Identication and Control*, 39(3):179–190, 2018a. doi:10.4173/mic.2018.3.4.

Peter Fritzson. MathModelica — An Object Oriented Mathematical Modeling and Simulation Environment. *Mathematica Journal*, 10(1), February 2006.

Peter Fritzson. *Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. Wiley IEEE Press, 2014. ISBN 9781-118-859124.

Peter Fritzson and Vadim Engelson. Modelica — A Unified Object-Oriented Language for System Modeling and Simulation. In *Proc. of the 12th European Conference on Object-Oriented Programming*, volume 1445 of *LNCS*, pages 67–90, Brussels, Belgium, July 1998. Springer Berlin Heidelberg.

Peter Fritzson and David Kågedal. Generating a Modelica Compiler from Natural Semantics Specifications. In *Proc. of the 1998 Summer Computer Simulation Conference (SCSC'98)*, Reno, Nevada, July 1998.

Peter Fritzson, Lars Viklund, Johan Herber, and Dag Fritzson. High Level Mathematical Modeling and Programming in Scientific Computing. *IEEE Software*, 12(4):77–87, July 1995. doi:10.1109/52.391838.

Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, and David Broman. The OpenModelica Modeling, Simulation,

and Software Development Environment. *Simulation News Europe*, 44/45, December 2005. See also: http://www.openmodelica.org. An earlier version in *Proceedings of the 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS'2005)*, Trondheim, Norway, October 13-14, 2005.

Peter Fritzson, Adrian Pop, David Broman, and Peter Aronsson. Formal Semantics Based Translator Generation and Tool Development in Practice. In *Proc. of the 20th Australian Software Engineering Conference (ASWEC 2009)*, Gold Coast, Queensland, Australia, April 2009a.

Peter Fritzson, Pavol Privitzer, Martin Sjölund, and Adrian Pop. Towards a Text Generation Template Language for Modelica. In *Proc. of the 7th International Modelica Conference*, Como, Italy, September 2009b. doi:10.3384/ecp09430124.

Peter Fritzson, Adrian Pop, and Martin Sjölund. Towards Modelica 4 Meta-Programming and Language Modeling with MetaModelica 2.0. Technical Report 2011:10, Linköping University, PELAB - Programming Environment Laboratory, 2011. URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-68361.

Peter Fritzson, Bernhard Bachmann, Kannan Moudgalya, Francesco Casella, Bernt Lie, Jiri Kofranek, and Massimo Ceraolo. *Introduction to Modelica with Examples in Modeling, Technology, and Applications*. Linköping University Interdisciplinary Studies, ISSN: 1650-9625. Linköping University Electronic Press, 2018b. URL http://omwebbook.openmodelica.org/. Accessed: September, 2018.

Peter Fritzson, Adrian Pop, Adeel Asghar, Bernhard Bachmann, Willi Braun, Robert Braun, Lena Buffoni, Francesco Casella, Rodrigo Castro, Alejandro Danós, Rüdiger Franke, Mahder Gebremedhin, Bernt Lie, Alachew Mengist, Kannan Moudgalya, Lennart Ochel, Arunkumar Palanisamy, Wladimir Schamai, Martin Sjölund, Bernhard Thiele, Volker Waurich, and Per Östlund. The OpenModelica Integrated Modeling, Simulation and Optimization Environment. In *Proc. of the 1st American Modelica Conference*, Cambridge, MA, USA, October 2018c. doi:10.3384/ecp18154206.

Peter Fritzson, Adrian Pop, Martin Sjölund, and Adeel Asghar. MetaModelica — A Symbolic-Numeric Modelica Language and Comparison to Julia. In *Proc. of the 13th International Modelica Conference*, Regensburg, Germany, March 2019. doi:10.3384/ecp19157289.

Mahder Gebremedhin. ParModelica: Extending the Algorithmic Subset ofModelica with Explicit Parallel LanguageConstructs for Multi-core Simulation. Master's thesis, Linköping University, Department of Computer and Information Science, 2011. URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-71612.

Mahder Gebremedhin. *Automatic and Explicit Parallelization Approaches for Equation Based Mathematical Modeling and Simulation*. PhD thesis, Linköping University, Department of Computer and Information Science, 2019. doi:10.3384/diss.diva-152789.

Mahder Gebremedhin and Peter Fritzson. Parallelizing Simulations with Runtime Profiling and Scheduling. In *Proc. of the 8th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT'17)*, Munich, Germany, 2017. ACM Digital Library. doi:10.1145/3158191.3158194.

Mahder Gebremedhin, Afshin Hemmati Moghadam, Peter Fritzson, and Kristian Stavåker. A Data-Parallel Algorithmic Modelica Extension for Efficient Execution on Multi-Core Platforms. In *Proc. of the 9th International Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp12076393.

Alan C. Hindmarsh, Peter N. Brown, Keith E. Grant, Steven L. Lee, Radu Serban, Dan E. Shumaker, and Carol S. Woodward. SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31 (3):363–396, 2005.

Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit — An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32 (3):298–312, 2011.

Rahul Jain, Priyam Nayak, A. S Rahul, Pravin Dalve, Kannan Moudgalya, P. R. Naren, Daniel Wagner, and Peter Fritzson. Implementation of a Property Database and Thermodynamic Calculations in OpenModelica for Chemical Process Simulation. *Industrial & Engineering Chemistry Research*, 58(18):7551–7560, February 2019. doi:10.1021/acs.iecr.8b05147.

JuliaControl. ControlSystems — A Control Systems Toolbox for Julia, 2019. URL https:

//github.com/JuliaControl/ControlSystems.jl. Accessed: 2020-09-11.

Julialang. Julia Language Documentation, Release 1.0, 2018. URL https://julialang.org. Accessed: 2020-09-11.

Herbert B. Keller. Global Homotopies and Newton Methods. In *Recent Advances in Numerical Analysis*, pages 73–94. Academic Press, 1978. doi:10.1016/b978-0-12-208360-0.50009-7.

Hassan Khalil. *Nonlinear Systems*. Prentice Hall, third edition, 2002. ISBN 978-0130673893.

Mohammad Khalili and Bernt Lie. Comparison of Linear Controllers for Nonlinear Openloop Unstable Reactor. In *Proc. of the 59th Conference on Simulation and Modelling (SIMS 59)*, Oslo Metropolitan University, Norway, September 2018. doi:10.3384/ecp18153185.

Feng Liang, Wladimir Schamai, Olena Rogovchenko, Sara Sadeghi, Mattias Nyberg, and Peter Fritzson. Model-Based Requirement Verification: A Case Study. In *Proce. of the 9th International Modelica Conference*, Linköping, Sweden, September 2012. doi:10.3384/ecp12076385.

Bernt Lie, Sudeep Bajrachary, Alachew Mengist, Lena Buffoni, ArunKumar Palanisamy, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. API for Accessing OpenModelica Models from Python. In *Proc. of The 9th EUROSIM Congress on Modelling and Simulation*, pages 707–713, Oulu, Finland, September 2016. IEEE. ISBN 978-91-7685-399-3. and in *Proc. of the 57th SIMS Conference on Simulation and Modelling (SIMS 2016)*, https://www.ep.liu.se/ecp/142/103/ecp17142103.pdf.

Bernt Lie, Arunkumar Palanisamy, Alachew Mengist, Lena Buffoni, Martin Sjölund, Adeel Asghar, Adrian Pop, and Peter Fritzson. OMJulia: An OpenModelica API for julia-modelica interaction. In *Proc. of the 13th International Modelica Conference*, Regensburg, Germany, March 2019. doi:10.3384/ecp19157699.

MathWorks. Matlab Product Overview, 2018. URL https://www.mathworks.com/products/matlab.html. Accessed: September, 2018.

MathWorks. MathWorks. Simulink - Simulation and Model-Based Design, 2019a. URL https://www.mathworks.com/products/simulink.html. Accessed: October, 2019.

MathWorks. MathWorks. Control System Toolbox Documentation, 2019b. URL https://se.mathworks.com/help/control/getting-started-with-control-system-toolbox.html. Accessed: October, 2019.

Dennis Meadows, William Behrens, Donella Meadows, Roger Naill, Jørgen Randers, and Erich Zahn. *Dynamics of growth in a finite world*. Wright-Allen Press Cambridge, MA, USA, 1974.

Donella Meadows, Jørgen Randers, and Dennis Meadows. *Limits to Growth: The 30-year Update*. Chelsea Green Publishing, 3rd edition, 2004.

Nils Menager, Niklas Worschech, and Lars Mikelsons. Toolchain for Rapid Control Prototyping using Rexroth Controllers and Open Source Software. In *Proc. of the 10th International Modelica Conference*, Lund, Sweden, March 2014. doi:10.3384/ecp14096371.

Gustavo Migoni, Ernesto Kofman, and François Cellier. Quantization-based new integration methods for stiff ordinary differential equations. *SIMULATION*, 88(4):387–407, June 2011. doi:10.1177/0037549711403645.

Robert Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997. ISBN 0-262-63181-4.

Modelica Association. Modelica: A Unified Object-oriented Language for Physical Systems Modeling, Language Specification Version 3.4, 2017. URL http://www.modelica.org/.

Modelica Association. SSP – MA Project for System Structure and Parameterization of Components for Virtual System Design, 2018. URL https://www.modelica.org/projects. Accessed: September, 2018.

Nima Mohajerin, Melissa Mozifian, and Steven Waslander. Deep learning a quadrotor dynamic model for multi-step prediction. In *Proc. of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2454–2459, May 2018. doi:10.1109/ICRA.2018.8460840.

Kannan Moudgalya. Crowdsourced Information Technology Content for Education and Employment. In *Proc. of the 18th IEEE International Conference on Advanced Learning Technologies (ICALT)*, pages 39–41, IIT Bombay, 2018, July 2018. doi:10.1109/ICALT.2018.00016.

Kannan Moudgalya, Bhargava Nemmaru, Kaushik Datta, Priyam Nayak, Rahul Jain, Peter Fritzson, and Adrian Pop. Large Scale Training through Spoken Tutorials to Promote and use OpenModelica. In *Proc. of the 12th International Modelica Conference*, May 2017. doi:10.3384/ecp17132275.

MSCSoftware. Adams Multibody Dynamics Simulation Software, 2020. URL https://www.mscsoftware.com/product/adams. Accessed: March, 2020.

Richard Murray and Scott Livingston. Python Control Systems Library, 2019. URL https://sourceforge.net/p/python-control/wiki/Home. Accessed: September, 2019.

Ekanathan Natarajan. KLU—A High Performance Sparse Linear Solver for Circuit Simulation Problems. Master's thesis, University of Florida, 2005. URL https://ufdcimages.uflib.ufl.edu/UF/E0/01/17/21/00001/palamadai_e.pdf.

Priyam Nayak, Pravin Dalve, Rahul Anandi Sai, Rahul Jain, Kannan Moudgalya, P. R. Naren, Peter Fritzson, and Daniel Wagner. Chemical Process Simulation Using OpenModelica. *Industrial & Engineering Chemistry Research*, 58(26):11164–11174, May 2019. doi:10.1021/acs.iecr.9b00104.

Akira Nishida. Experience in Developing an Open Source Scalable Software Infrastructure in Japan. In *Computational Science and Its Applications – ICCSA 2010*, pages 448–462, Berlin, Heidelberg, March 2010. Lecture Notes in Computer Science, vol 6017. Springer. doi:10.1007/978-3-642-12165-4_36.

Nvidia. Nvidia. CUDA Compute Unified Device Architecture programming guide Version: 2.0, 2008.

OCaml. Ocaml web site, 2018. URL https://ocaml.org/. Accessed: September, 2018.

Lennart Ochel. *Petri-Netz-basierte Simulation biologischer Prozesse mit OpenModelica*. Doctoral thesis, Universität Bielefeld, AG Bioinformatik Technische Fakultät, 2017. URL https://pub.uni-bielefeld.de/record/2913956.

OMSimulator. *OMSimulator*. OSMC – Open Source Modelica Consortium, 2020. URL https://www.openmodelica.org/doc/OpenModelicaUsersGuide/1.16/omsimulator.html. Accessed: 2020-10-02.

OMSimulator 1.0. *OpenModelica User's Guide, Chapter 6. OMSimulator 1.0*. OSMC – Open Source Modelica Consortium, 2017. URL https://www.openmodelica.org/doc/OpenModelicaUsersGuide/v1.12.0/omsimulator.html. Accessed: September, 2018.

OMSysIdent. *OMSysIdent*. OSMC – Open Source Modelica Consortium, 2020. URL https://www.openmodelica.org/doc/OpenModelicaUsersGuide/1.16/systemidentification.html. Accessed: 2020-10-01.

OpenCL. *The OpenCL Specification Version: 1.0*. Khronos OpenCL Working Group, 2018. URL https://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf. Accessed: November, 2018.

OpenModelica. *OpenModelica User's Guide Version 1.16*. OSMC – Open Source Modelica Consortium, 2020. URL https://www.openmodelica.org/doc/OpenModelicaUsersGuide/1.16/. Accessed: 2020-09-11.

Martin Otter, Nguyen Thuy, Daniel Bouskela, Lena Buffoni, Hilding Elmqvist, Peter Fritzson, Alfredo Garro, Audrey Jardin, Hans Olsson, Maxime Payelleville, Wladimir Schamai, Eric Thomas, and Andrea Tundis. Formal Requirements Modeling for Simulation-Based Verification. In *Proc. of the 11th International Modelica Conference*, September 2015. doi:10.3384/ecp15118625.

Constantinos Pantelides. The Consistent Initialization of Differential-Algebraic Systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988. doi:10.1137/0909014.

Linda Petzold. A Description of DASSL: A Differential/Algebraic System Solver. In *Proc. of the 10th IMACS World Congress*, Montreal, August 1982.

Linda Petzold, Shengtai Li, Yang Cao, and Radu Serban. Sensitivity Analysis of Differential-Algebraic Equations and Partial Differential Equations. *Computers & Chemical Engineering*, 30(10):1553 – 1559, 2006. doi:10.1016/j.compchemeng.2006.05.015.

Adrian Pop. *Integrated Model-Driven Development Environments for Equation-Based Object-Oriented Languages*. PhD thesis, Linköping University, Department of Computer and Information Science, 2008. URN: urn:nbn:se:liu:diva-11416.

Adrian Pop and Peter Fritzson. MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. In David Lightfoot and Clemens Szyperski, editors, *Modular Programming Languages*, pages 211–229, Berlin / Heidlberg, 2006.

Modular Programming Languages. JMLC 2006. Lecture Notes in Computer Science, Vol. 4228. Springer. doi:10.1007/11860990_14.

Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, and David Akhvlediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In *Proc. of the 5th International Modelica Conference*, Vienna, Austria, September 2006.

Adrian Pop, Martin Sjölund, Adeel Ashgar, Peter Fritzson, and Francesco Casella. Integrated Debugging of Modelica Models. *Modeling, Identification and Control*, 35(2):93–107, 2014. doi:10.4173/mic.2014.2.3.

Adrian Pop, Per Östlund, Francesco Casella, Martin Sjölund, and Rüdiger Franke. A New OpenModelica Compiler High Performance Frontend. In *Proc. of the 13th International Modelica Conference*, Regensburg, Germany, March 2019. doi:10.3384/ecp19157689.

Sabrina Proß and Bernhard Bachmann. PNlib - An Advanced Petri Net Library for Hybrid Process Modeling. In *Proc. of the 9th International Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp1207647. URL https://github.com/AMIT-FHBielefeld/PNlib.

Project Jupyter. Jupyter notebooks, 2016. URL https://jupyter.org/. Accessed: 2020-09-11.

Python Software Foundation. Python Programming Language, 2018. URL https://www.python.org/. Accessed: Sept, 2018.

Xiaolin Qin, Juan Tang, Yong Feng, Bernhard Bachmann, and Peter Fritzson. Efficient index reduction algorithm for large scale systems of differential algebraic equations. *Applied Mathematics and Computation*, 277:10–22, 2016. doi:10.1016/j.amc.2015.11.091.

Xiaolin Qin, Lu Yang, Yong Feng, Bernhard Bachmann, and Peter Fritzson. Index reduction of differential algebraic equations by differential dixon resultant. *Applied Mathematics and Computation*, 328:189–202, 2018. ISSN 0096-3003. doi:10.1016/j.amc.2017.12.029.

Vitalij Ruge, Willi Braun, Bernhard Bachmann, Andrea Walther, and Kshitij Kulshreshtha. Efficient Implementation of Collocation Methods for Optimization using OpenModelica and ADOL-C. In *Proc. of the International Modelica Conference*, Lund, Sweden, March 2014. doi:10.3384/ecp140961017.

Eva-Lena Sandelin, Susanna Monemar, Peter Fritzson, and Peter Bunus. DrModelica An Interactive Tutoring Environment for Modelica. In *Proc. of the 3rd International Modelica Conference*, Linköping, Sweden, November 2003. URL www.openmodelica.org.

Wladimir Schamai. *Model-Based Verification of Dynamic System Behavior against Requirements: Method, Language, and Tool*. PhD thesis, Linköping University, Department of Computer and Information Science, November 2013. doi:10.3384/diss.diva-98107.

Wladimir Schamai, Lena Buffoni, and Peter Fritzson. An Approach to Automated Model Composition Illustrated in the Context of Design Verification. *Modeling, Identification and Control*, 35(2):79–91, 2014. doi:10.4173/mic.2014.2.2.

Wladimir Schamai, Lena Buffoni, Nicolas Albarello, Pablo Fontes De Miranda, and Peter Fritzson. An Aeronautic Case Study for Requirement Formalization and Automated Model Composition in Modelica. In *Proc. of the 11th International Modelica Conference*, Versailles, France, September 2015. doi:10.3384/ecp15118911.

Hugo D. Scolnik. A critical review of some global models. In *Global and Large Scale System Models*, Berlin, Heidlberg, 1979. Lecture Notes in Control and Information Sciences, vol 19. Springer. doi:10.1007/bfb0049022.

Dale Seborg, Thomas Edgar, Duncan Mellichamp, and Francis Doyle. *Process Dynamics and Control*. Wiley, 2011. ISBN 978-1-119-28591-5.

Alachew Shitahun, Vitalij Ruge, Mahder Gebremedhin, Bernhard Bachmann, Lars Eriksson, Joel Andersson, Moritz Diehl, and Peter Fritzson. Model-Based Dynamic Optimization with OpenModelica and CasADi. *IFAC Proceedings Volumes*, 46(21):446–451, 2013. doi:10.3182/20130904-4-jp-2042.00166.

Michael Sielemann, Francesco Casella, Martin Otter, Christop Clauß, Jonas Eborn, Sven Erik Matsson, and Hans Olsson. Robust Initialization of Differential-Algebraic Equations Using Homotopy. In *Proc. of the 8th International Modelica Conference*, Dresden, Germany, June 2011. doi:10.3384/ecp1106375.

Martin Sjölund. *Tools and Methods for Analysis, Debugging, and Performance Improvement of Equation-Based Models*. PhD thesis, Linköping University, Department of Computer and Information Science, 2015. doi:978-91-7519-071-6.

Martin Sjölund, Peter Fritzson, and Adrian Pop. Bootstrapping a compiler for an equation-based object-oriented language. *Modeling, Identification and Control*, 35(1):1–19, 2014. doi:10.4173/mic.2014.1.1.

Gustaf Söderlind and Sven Erik Mattsson. Index reduction in differential-algebraic equations using dummy derivatives. *SIAM Journal of Scientific and Statistical Computing*, 14(3):677–692, 1993.

Allan G. Taylor and Alan C. Hindmarsh. User Documentation for KINSOL, a Nonlinear Solver for Sequential and Parallel Computers. Technical report, Lawrence Livermore Technical Laboratory, July 1998. https://computing.llnl.gov/projects/sundials/kinsol.

Watten Teitelman. *INTERLISP Reference Manual*. Xerox Palo Alto Research Center, 1974.

Tensorflow.org. An end-to-end open source machine learning platform, 2019. URL https://www.tensorflow.org/. Accessed: Sept, 2019.

Bernhard Thiele, Thomas Beutlich, Volker Waurich, Martin Sjölund, and Tobias Bellmann. Towards a Standard-Conform, Platform-Generic and Feature-Rich Modelica Device Drivers Library. In *Proc. of the 12th International Modelica Conference*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132713.

Bernhard Thiele, Bernt Lie, Martin Sjölund, Adrian Pop, and Peter Fritzson. Controller Design for a Magnetic Levitation Kit using OpenModelica's Integration with the Julia Language. In *Proceedings of the 13th International Modelica Conference*, Regensburg, Germany, March 2019. doi:10.3384/ecp19157303.

Hubert Thieriot, Maroun Nemera, Mohsen Torabzadeh-Tari, Peter Fritzson, Rajiv Singh, and John John Kocherry. Towards Design Optimization with OpenModelica Emphasizing Parameter Optimization with Genetic Algorithms. In *Proc. of the 8th International Modelica Conference*, Dresden, Germany, March 2011. doi:10.3384/ecp11063756.

John Tinnerholm. An LLVM backend for the Open Modelica Compiler. Master's thesis, Linköping University, Department of Computer and Information Science, 2019. URL http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-154291.

John Tinnerholm, Martin Sjölund, and Adrian Pop. 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*, EOOLT '19, page 11–19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450377133. doi:10.1145/3365984.3365990.

VDI – Verein Deutscher Ingenieure. VDI 2048 BLATT 3 "Uncertainties of measurement during acceptance tests on energy-conversion and power plants – examples, especially preparation of acceptance of a gas and steam power plant", May 2012. URL https://www.beuth.de/en/technical-rule/vdi-2048-blatt-3/152248933.

VDI – Verein Deutscher Ingenieure. VDI 2048 BLATT 1 "Control and quality improvement of process data and their uncertainties by means of correction calculation for operation and acceptance tests, VDI-Handbuch Energietechnik", September 2017. URL https://standards.globalspec.com/std/13137270/VDI%202048%20BLATT%201.

VDI – Verein Deutscher Ingenieure. VDI 2048 BLATT 2 "Control and quality improvement of process data and their uncertainties by means of correction calculation for operation and acceptance tests, VDI-Handbuch Energietechnik – Examples, especially retrofit measures", June 2018. URL https://www.beuth.de/en/technical-rule/vdi-2048-blatt-2/280290570.

Pieter Jacobus Vermeulen and Dawid Cornelius Johannes de Jongh. Parameter sensitivity of the 'limits to growth' world model. *Applied Mathematical Modelling*, 1(1):29 – 32, 1976. ISSN 0307-904X. doi:10.1016/0307-904X(76)90021-4.

Andreas Wächter and Lorenz T. Biegler. On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming. *Mathematical Programming*, 106(1): 25–57, April 2006. doi:10.1007/s10107-004-0559-y.

Marcus Walther, Volker Waurich, Christian Schubert, and Ines Gubsch. Equation based parallelization of Modelica models. In *Proc. of the 10th International Modelica Conference*, Lund, Sweden, March 2014. doi:10.3384/ecp140961213.

Volker Waurich and Jürgen Weber. Interactive FMU-Based Visualization for an Early Design Experience. In *Proc. of the 12th International Modelica Conference*, Prague, Czech Republic, May 2017. doi:10.3384/ecp17132879.

Susann Wolf, Joachim Haase, Christoph Clauß, Michael Jöckel, and Jürgen Lösch. Methods of Sensitivity Calculation Applied to a Multi-Axial Test Rig for Elastomer Bushings. In

Proc. of the 6th International Modelica Conference, Bielefeld, Germany, March 2008. URL http://www.modelica.org/events/modelica2008/Proceedings/html/proceedings.html.

Stephen Wolfram. *The Mathematica Book*. Wolfram Media, Inc, 5th edition, 2003.

Wolfram Research. Wolfram System Modeler Documentation and Overview, 2018. Accessed: September, 2018. URL http://www.wolfram.com/system-modeler/.

Zeltom LLC. Zeltom Electromagnetic Levitation System, 2019. URL http://zeltom.com/emls.html. Accessed: September, 2019.

Quan Min Zhu, Li Feng Zhang, and Ashley Longden. Development of omni-directional correlation functions for nonlinear model validation. *Automatica*, 43:1519–1531, 2007. ISSN 0005-1098. doi:10.1016/j.automatica.2007.02.010.

Dirk Zimmer. A Planar Mechanical Library for Teaching Modelica. In *Proc. of the 9th International Modelica Conference*, Munich, Germany, September 2012. doi:10.3384/ecp12076681.