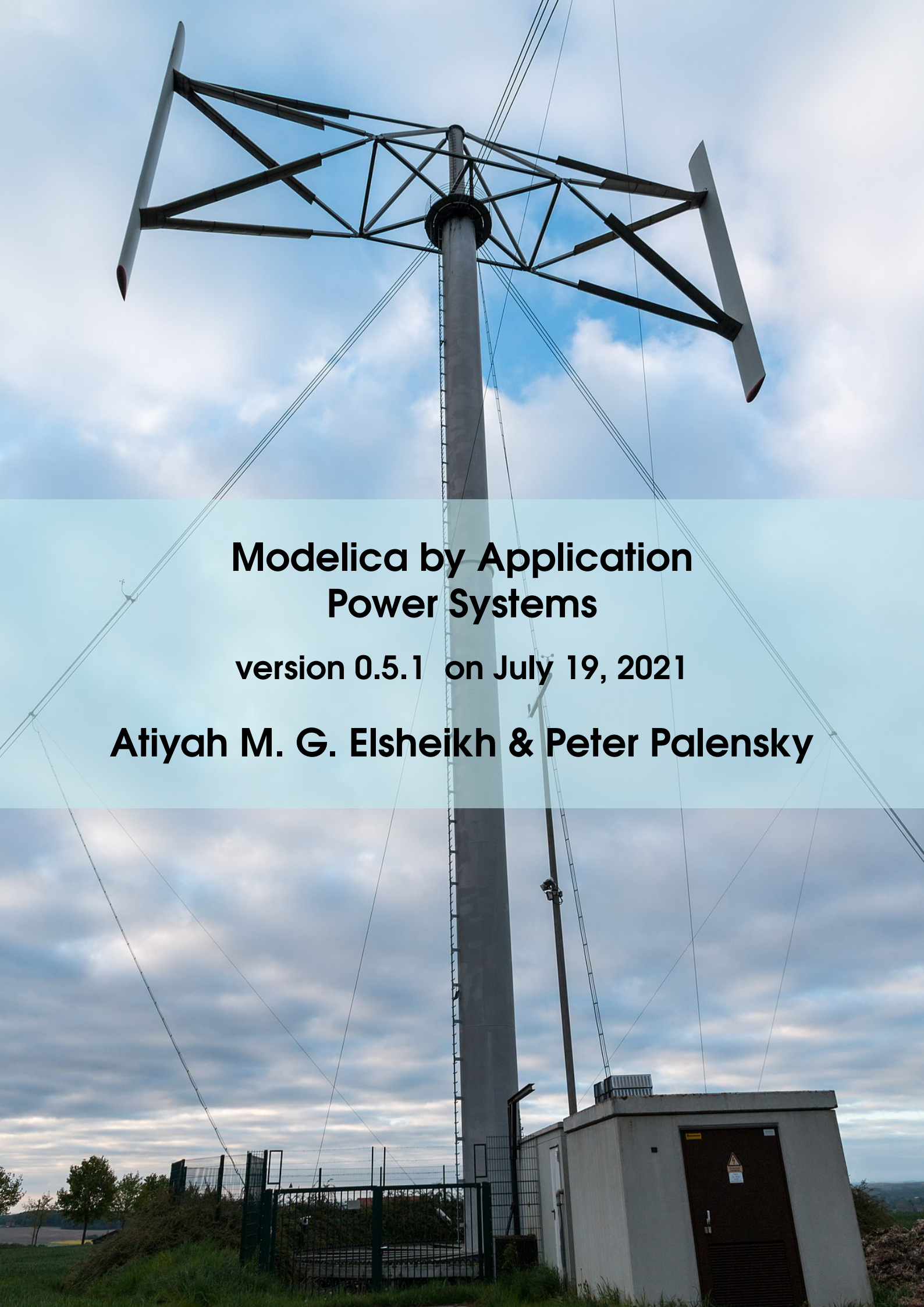


Modelica by Application
Power Systems

version 0.5.1 on July 19, 2021

Atiyah M. G. Elsheikh & Peter Palensky

Book Website:

https://github.com/Mathemodica/ModelicaPowerSystemBook

Dedication from the first author to

First Edition (V1.0) to appear 1st of Sep. 2021

Pre-order a free (electronic) edition 1.0 @
https://gum.co/mathemodica-powsys-free

Sponsor the maintenance and progress
by the first author through

I. Pre-ordering the book for
as-much-as-you-think-this-book-deserves @
https://gum.co/mathemodica-powsys

II. Get a continuous access to the actual version:
through sponsorship

A. Single-time sponsorship @
https://www.paypal.com/paypalme/mathemodica

B. Periodic sponsorship @
https://gum.co/mathemodica-powsys-sponsorship
https://github.com/sponsors/AtiyahElsheikh

Subscripe to newsletters and posts from
Mathemodica.com @
https://gumroad.com/mathemodica

About

This is a comprehensive but a concise and educational (e-)book aiming at advertising Modelica-based technologies particularly useful for power system modeling applications. We hope that this book is useful not only for power system modelers desiring to get a quick idea about the benefits of employing Modelica but also for those Modelica modelers desiring a starting guide into the world of Power System.

Involvement & Conditions

If you are clearly involved in power-system related activities using the Modelica language, you are highly encouraged to actively improve the state of this book whenever and/or wherever possible. For this reason, this book is available on the platform Overleaf which allows collaborative writing. Your useful scientific involvement, in whichever form, shall be acknowledged, unless explicitly communicated that this is not desired.

However, it is important to note that, any suggested enhancement should be valuable, concise, accurate and elegant. The authors have the right to reject or to ask for specific corrections or improvements to any suggested enhancement.

Overleaf.com

In case you don't have an account on overleaf.com, to this book benefit's consider creating an account using the following referral link:

`https://www.overleaf.com?r=e7d83309&rm=d&rs=b`

Contact

Consider contacting atiyah.elsheikh -at- mathemodica.com if you would like to:

- contribute to the text: Consider providing a brief summary in advance of the purpose of your desired involvement
- provide suggestions or pdf-annotated review, suggested corrections, suggested text, etc.
- provide a general feedback
- provide suggested topics or materials that this book should cover
- have access to the latex sources for whatever purpose you need, e.g. project proposals, user guides, etc.

About Mathemodica.com

By the time of relasing version 0.5 of the book (July 2021), Mathemodica.com is currently a virtual organization. It currently reflects the main hobby-based (but occoasionally professional) activites of the first author ususally on his free-time.

One of the idea behind Mathemodica.com, which is still evolving and subject to continuous improvement, is to provide a transparent, collaborative and independent platform for those who would like to sponsor their own ideas and works concerning Modelica-like technologies (libraries, tools, educational e-books, tutorials, etc.).

It is hoped that the resulting products to be open-source and free. If you'd like to become a part of this and contribute to the evolve of Mathemodica.com consider viewing:

http://mathemodica.com/modelicans/

Sponsorship

Sponsorship (cf. to the quick urls given at the beginning of the book) is appreciated as an aid and accelerator for

- financing the continuation of maintaining, actualizing and progressing this book
- executing similar initiatives for establishing educational contents (tutorials, books and libraries)
- among other similar activities by members of Mathemodica.com, cf.
  http://mathemodica.com/projects

These activities are in conformance with the spirit of open science initiative.

Acknowledgment

We are acknowledging our former employer, Austrian Institute of Technology GmbH. This book has started initially as a technical report during our roles there. The early version was still in a primitive state until it was recently decided to re-write it as a comprehensive book.

Moreover, couple of capitals of this book has been written by others. Without their contribution, the book would be definitely less valuable. Thus, we'd like to thank (in alphabetical order of family names):

- Prof. Andrea Benigni, RWTH Aachen and Research Center Jülich, with his great help, this book was further tuned for Electrical Engineers. Particularly, major parts of Chapter 2 and Section 10.1 were originally written by him.
- Assoc. Prof. Omar Faruque, Florida State University, for presenting this initiative at a PES general meeting
- Prof. Antonello Monti, RWTH Aachen, being the initiator of the idea of having a comprehensive report that gathers all useful aspects Modelica can provide for power system modeling applications. The first chapter was originally written by him.

We believe that online Modelica educational materials need to be gathered together and since the idea of having a freely accessible book that is meanwhile sponsored (or to be sponsored) by any one on the basis of pay-as-much-as-you-think-this-book-deserve is inspired by the author of the book "Modelica by Examples", thus, special appreciation goes to Dr. Michael Tiller, for:

- his initial agreement in hosting or linking a possible future html-based version of this book to the platform
  `https://modelica.university`
- his technical tips, recommendations and his willing to help us (despite apparently being a very busy person with his own duties)

We hope to have enough energy in near future to learn the technology needed to bring this book to the platform modelica.university and to establish url-links to adequate materials in his book whenever more in-depth clarification of Modelica syntax is needed. In that way, the focus of this book can remain on the applications side of power systems rather than attempting to illustrate the tiniest details of the Modelica language[1]

We also would like to thank

[1] By the release version 0.5, I still did not invest enough time in this issue. Moreover, I am not so sure about a good way to convert and synchrnoize latex code to html. If anyone with proper technical knowledge would like to get engaged he/she is thankfully encouraged to contact us

- Dr. Mathias Legrand for allowing to employ this wonderful latex template accessible under
  https://www.latextemplates.com/template/the-legrand-orange-book

The pictures in this book come from:

- Dietmar Rabich / Wikimedia Commons / “Dülmen, Rorup, Windenergieanlage – 2015 – 5972” / CC BY-SA 4.0
- Dietmar Rabich / Wikimedia Commons / “Schöppingen, Schöppinger Berg – 2015 – 8513” / CC BY-SA 4.0
- Peter Haas / Wikimedia Commons / Windpark Höflein, Niederösterreich / CC BY-SA 3.0
- J. E. Wolters / Wikimedia Commons / Wasserkraftwerk an der Isar bei Niederaichbach / CC BY-SA 4.0
- Kreuzschnabel / Wikimedia Commons / Das Kraftwerk Heilbronn / CC BY-SA 3.0
- Dmitry Makeev / Wikimedia Commons / Stung Hav Coal Power Plant / CC BY-SA 4.0

Anyone who made his own nice pictures and would like to share it here is encouraged to contact us.

Contents

I Introduction

1. Motivation and Outline

Original text by

Prof. Antonello Monti (+ AE)

Traditionally, modeling and simulation has been conducted within **a single physical domain type**. Electrical engineers used to develop electrical systems primarily focusing on **grid modeling** and mostly neglecting other interacting domains. One of the significant exception to this rule has been a simplified representation of the rotating mass for stability analysis (Kundur, Balu, and Lauby 1994; Ulbig, Borsche, and Andersson 2014).

In the last decade though, we have experienced a growing interest towards **multi-physics design** in many different areas of applications. Two concrete examples of this type are the avionics industry and ship industry in relation to programs such as More Electric Aircraft (Hafez and Forsyth 2009; Rahrovi and Ehsani 2019) and All Electric Ship (Sulligoi, Vicenzutti, and Menis 2016; Thongam et al. 2013). The development of such projects has shown the limitation of the single-domain approach and has paved the way to comprehensive approaches for multi-physical modeling and simulation.

One response to the problem has been the development of **co-simulation** interfaces and standards (Blochwitz and al. 2011; Gomes et al. 2018; Müller et al. 2016). Co-simulation offers the user the possibility to operate in a domain-specific environment and to relegate the integration of other simulation platforms to be mostly a software challenge.

While this approach has some clear advantages from user perspective, first of all the possibility to continue operating with the same tools also in the new operating conditions, co-simulation tends to have a significant computation and implementation overhead given by the co-presence of more than one simulation platform. Nevertheless, many tools exist simplifying the co-simulation implementation overhead, e.g. (Ochel and al. 2019; Widl et al. 2013).

Furthermore, it is quite easy to face versioning issue created by the fact that the project as a whole is stored in multiple files. There are so many technical difficulties. Detailed illustration goes beyond this text.

Proposition 1.1 In summary, one day co-simulation will be most effective if it converges to the state that a modeler is neither aware nor technically caring that his descriptive models are executed via co-simulation.

Consequently there is a growing interest towards **multi-physical modeling languages**. Multi-domain languages provide an interesting solution, majorly because they are languages and not simulation platforms.

This means that the modeling effort and the solution effort are clearly separated. This separation brings an incredible benefit from the user point of view that is able to migrate from one simulation tool to another without repeating the modeling effort. This is because the same model specification is used among the different tools.

Proposition 1.2 Converting from a tool user to a model designer via a well-designed modeling language extends the horizon of realizable applications to an extent that is beyond imagination.

There are many simulation languages of this type available s.a. VHDL-AMS(Ashenden, Peterson, and Teegarden 2003), gPROMS (*Process Systems Enterprise*) and **Modelica** (Elmqvist 2014). The last language has been developed as an initiation for a standard modeling language used by a large modeling community in many modeling domains. This has resulted in an open-source non-property specification language continuously maintained, supported and progressed both from academia and industry.

Outline of this book

The outline of the book is structured as follows.

In Part I we first review **the major challenges** in power system modeling and simulation. These challenges impose **new requirements** for modeling and simulation tools aiming at **modern power system modeling applications**. In order to appreciate the evolution process of such modern modeling languages, we reveal how the accumulation of progressive

milestones along many decades led to the concepts on which some current state of the art of modeling languages are based on.

In Part II, we dive into the Modelica language, giving a non-familiar reader though rather a compact overview of the language but hopefully generous enough to recognize its potentials in the scope of power system modeling applications. This demonstration is based on one of the existing and elegant open-source libraries. This particular library provides the opportunity to demonstrate many syntactically significant features of the Modelica language.

In Part III we overview current state of the Modelica language. Particularly, we summarize

- **the language specification**, the underlying process of its development through the Modelica association
- **the Modelica Standard Library** and some of its major components significant for power system modeling,
- **the functional mockup interface** including related projects
- as well as other significant related topics that may matter the reader

Furthermore, reflecting our humble struggle to eliminate the thirst of a knowledge seeker, we provide a comprehensive list of **open-source Modelica libraries** in the field of (and in many areas related to) power systems. The list includes libraries in the field of energy in buildings and cities as well as other fundamental methodological libraries, e.g. for generating statistical distributions or for imitating a hybrid paradigm.

Additionally, another list of **commercial Modelica technologies** is aimed. However, tool vendors, library developers and interested parties need to report us their relevant technologies and provide us adequate text, references and corresponding websites.

Finally, since power systems models are naturally large-scale, in Part IV we demonstrate current **scalability challenges** facing Modelica-based tools. These are addressed together with recently and actually conducted research efforts for overcoming these scalability challenges. We summarize with **a list of benefits the Modelica language** can offer in the context of modern power system modeling applications. We overview **a list of potential extensions** to the Modelica language in order to enable further capabilities, e.g. partial differential equations or prototyping of simulation-optimization problems.

2. Modeling Challenges

Original text by

Prof. Andrea Benigni (+ PP + AE)

Simulation is one of the most important tools in power system planning and analysis, both for the electrical system as a standalone subject and as a part of an interdependent, multi-physical system. In this chapter a glossary of essential terminologies to be consulted or refreshed is introduced. Afterwords, traditional approaches to power system simulation are reviewed. Then modern aspects in power systems, that need to be address by simulation tools, are demonstrated.

2.1 Terminologies of power systems

We would love this book to be readable for Scientists and Engineers not necessarily familiar with power systems. However, this would mean to overload this book with the responsibility of competing with many excellent existing books and even some existing online materials. This is definitely not the purpose of this book.

Nevertheless, we rather recommend readers not familiar with power system components (e.g. transmission lines, loads, busses, etc.) and terminologies s.a. phase angles, reactive power, etc. to consult relevant introductory texts in the long-term.

In the short-term, the following hyper-linked terminologies can be quickly consulted from Wikipedia, Circuit Globe and possibly other online resources:

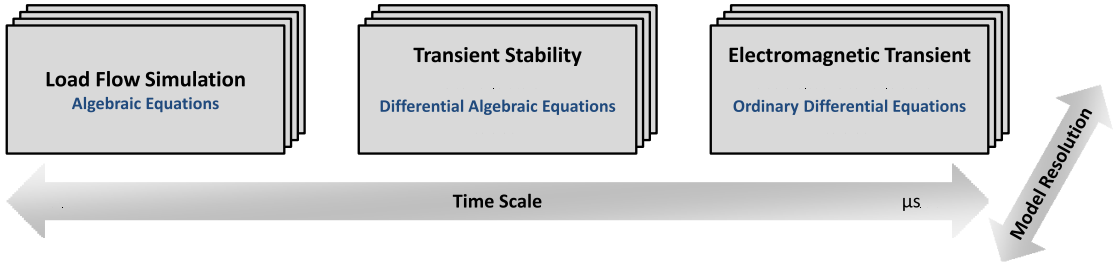


Figure 2.1: Typical classical power system simulation studies w.r.t. time domain and model types

- Electric Power Systems (Wikipedia) or (Circuit Globe)
- Loads (Circuit Globe)
- Direct vs. alternative currents (Circuit Globe)
- Transformer (Circuit Globe)
- Synchronous generator (Circuit Globe)
- Phase angle (complex numbers – Wikipedia) or Phase angle (waves – Wikipedia)
- Active and reactive power (AC power – Wikipedia)
- Voltage stability limits

R Consider recommending us further terminologies and resources that need to be initially consulted or refreshed

2.2 Traditional power system simulation studies

Traditionally, simulation analysis in support of power system design is performed by well-established mathematical techniques. Figure 2.1 demonstrates three commonly utilized types of simulations briefly summarized in the following subsections.

2.2.1 Load-flow studies

Load-flow studies (Prabhu et al. 2016) are performed to determine **the steady-state operation** of an electric power system. It corresponds to the energy flow through each transmission line described via a set of **non-linear algebraic equations**. The solution determines the magnitude and phase angle of the voltage at each bus, and the real and reactive power flowing in each line.

The main **applications** of this type of analysis is to determine if

- the system voltages remain within specified limits under various contingency conditions (e.g. N-1 analysis)
- whether equipment such as transformers and conductors are overloaded

Load-flow studies are often used for planning, economic scheduling, and control of an existing system as well as planning its future expansion identifying the need for additional

generation, lines, VAR support, or the placement of capacitors and/or reactors to maintain system voltages within specified limits.

2.2.2 Transient stability simulation

Transient stability simulation (Halder, Pal, and Mondal 2018) is often referred as **quasi-dynamic simulation** described by a set of **Differential Algebraic Equations** (DAE). It is used to determine whether in consequence of a contingency the power system returns to a stable and acceptable operating point. Typical **applications** are:

- identifying fault clearing time
- checking generator rotor angle stability
- assessing system stability margin
- evaluating motor dynamic acceleration and reacceleration impact and
- preparing and testing load shedding schedule

Some quantities are assumed to be

- constant (e.g. LTC position)
- fast enough to be represented by algebraic relation (e.g. synchronous machine stator dynamics, voltage source converter dynamics)
- represented by differential equation (e.g. turbine and synchronous machine rotor dynamics)

Transient stability covers simulation windows that go from tens of millisecond to hundreds of seconds.

2.2.3 Electromagnetic / Electromechanical transient stability simulation

Electromagnetic / Electromechanical transient stability simulation (Dommel 1969; Huang 2016) is described with a set of nonlinear **Ordinary Differential Equations** (ODE) model of the network and is typically used to analyze dynamical behavior induced by rapid events. Typical applications are to analyze

- the insulation coordination as consequence of fast transient with the purpose of determining surge arrestor ratings and characteristics
- overvoltage due to circuit breaker operation or transients operations due to power electronics systems and ferroresonance phenomena

Typically the analyzed time-axis spans from hundreds of nanoseconds to tens of microseconds.

Changing the analyzed dynamics typically involves a re-modeling of the complete system with different component models and tools, therefore doubling the modeling effort and creating an additional source of errors. The situation is typically even worse because for each time scale analysis various models are created each comprising a different resolution.

2.3 Modern aspects in power system modeling applications

2.3.1 Monitoring and control

The monitoring and control of future power systems are expected to be characterized by the distribution of functions and by the dependence on **communications** (Sabeeh and Gan 2016). Distributed monitoring and control is increasingly involved because of the

1. features of new resources, i.e. **renewable energy resources**, generation and storage
2. participation of loads in the energy management, e.g. buildings or houses equipped with solar plants
3. the demand for fast, local reactivity for dynamic control and protection

Given the finer granularity of the controllable energy entities, and given the natural variability of some renewable sources, the sensitivity to individual load variation is greater than in traditional systems. Furthermore, the future energy grid is expected to incorporate electrical, gas and heat networks, to achieve maximum usage of the available energy in every form, and storage capacity particularly in thermal form (Leitner et al. 2019).

2.3.2 Complexity

In this context the design of each part (subsystem, component, algorithm, logic) is a challenge due to the interactions, the interdependencies, and the dimension. This cannot be simplified via de-coupling, without losing essential dynamics. The impact of the individual element on the system, and vice versa is not straightforward to be inferred analytically (Hiskens and Pai 2000).

Consequently, the traditional design spiral cannot be applied to the considered kind of complex power systems. Overall this leads to the creation of very large systems that need to be analyzed via simulation in reasonable runtime not slowing the design process and enabling real-time execution.

2.3.3 Variety of time scales

The selection of the proper time scale and level of details has been based on the experience gained on well-defined operation, design conditions and on quite well-known and separate dynamic behaviors. Meanwhile considering the increasing complexity that terrestrial power systems are experiencing due to implementation of the **smart grid** concept and the need for a more systematic approach to simulation accuracy selection are issues of growing importance.

Specifically, future electrical subsystems will exhibit more interaction with several different physical phenomena of different time constants. Involved **distribution networks** will experience more dynamical behaviour dramatically enlarging the size of the system to be analyzed. Additionally, the growing presence of power electronics converters will significantly decrease the typical power system time constants.

2.3.4 Stochastic effects

Stochastic effects of future power systems are expected to be highly frequent mainly due to the volatility of some of the new, pervasive energy sources (particularly large and small renewable sources, and consumer owned small sources) and the effects of communication networks (e.g. delays). These stochastic phenomena are in addition to those that are typical also of classical power networks, originating from load behavior, prices and components reliability.

2.3.5 Communication aspects

The distribution of control and monitoring functions, and by the remote coordination of its components leads to an increasingly relevant role of the communication infrastructure and to a heavier dependence of the performance of the entire system on communication performance.

Effects such as time delay, packet loss, packet corruption, disconnections, packet re-ordering, jittering, etc. may affect the timely, stable reaction of power components and control. Conversely the operating mode of the power components affects the traffic, delay, etc. of the communication network.

2.3.6 Design process

Mismatches are to be expected between the models used in the design process together with field conditions and the models of the designed device and the real device. These mismatches cannot be compensated by oversizing or guaranteeing large margins from the critical operating points, as this would lead to scaling up quickly, leading to infeasibility.

Furthermore, the testing of the model of the device, even within the model of the entire system, may be insufficient. Hence, physical realization must be tested in most realistic settings at the greatest possible extent prior to deployment.

In this context incremental prototyping tools have to be considered as a fundamental tool to support the development of new products. As consequence real-time simulation and (Power) Hardware In the Loop techniques are to play a fundamental role.

3. The Rise of Modelica

3.1 Pre-era Modelica

3.1.1 Block diagrams

Before the rise of digital computers, modeling and simulation of mathematical Ordinary Differential Equations (ODEs) were realizable by earlier electrical **analog computers**. Exploiting physically continuous phenomena have enabled the simulation of ODEs. This can be achieved by the means of **analog circuits** components, (Ragazzini, Randall, and Russell 1947) as well as with mechanical based analog computers.

The underlying typical **block diagram**, conceptually and visually corresponding to such analog circuits, is composed of many interconnected blocks (Cellier 1991). Each block:

- physically corresponds to a common element of analog circuits, e.g. a resistor, an inductor, a capacitor, among other possibilities
- conceptually maps a mathematical function of a set of input signals to an output signal

Practically, compositions of such blocks are capable of imitating an ODE among other possibilities of mathematical continuous functions (Kolmogorov 1956).

Due to this background, the block diagram approach was initially the dominant modeling approach by earlier general-purpose simulation languages. Given a block diagram representation of a system model (e.g. an electrical circuit), it is straightforward to map this

representation to a set of equations.

Proposition 3.1 The **computational structure** of a given physical system is well-preserved in the block diagram representation.

Nevertheless, this modeling approach, though particularly useful in some application domains like control applications (Otter and Cellier 1996), was not the most suited for the new era.

Why?

The reason is that, it is usually not possible to recognize the real topology of the system (e.g. a large-scale electrical circuit) from a large-scale block diagram. In other words, it is not always possible to easily recognize the real topology of a system:

- its physical subsystems
- their components
- their connections and interactions

cf. (Cellier 1991, Ch. 7) for further detailed illustration.

This has enormous drawbacks on prototyping capabilities. For instance, A slight modification in the physical system (e.g. insertion of a resistor in an electrical circuit) can not be easily reflected in the corresponding block diagram.

Proposition 3.2 The **topological structure** of the physical system is not reflected when using the block diagram representation.

3.1.2 Bond graphs

On April 24th 1959, a new era of modeling paradigms emerged. Namely, the MIT-Professor Henry M. Paytner pioneered the physically-universal **Bond Graph** modeling concept allowing a system representation that both reflects the computational and topological structure (Paynter 1961).

What does the last sentence actually mean?

As we have just learned, a connection between components within a block diagram approach corresponds to a causal relation between an output signal and a set of input signals.

In contrary within a bond graph, a connection (i.e. an edge) between system components (i.e. nodes) rather corresponds to universal physical laws of Physics, e.g. Kirchhof's lows for current (so-called **1-Junction nodes**) or Kirchhof's lows for voltage (so-called **0-Junction nodes**).

The 1-junction nodes exhibit the flow of power while 0-junction nodes maintain efforts through adjacent nodes, interested readers can consult (Åström, Elmqvist, and Mattsson 1998; Cellier 1991) for further detailed illustration.

This concept is not domain-specific. It generalizes well to other physical domains. This is achieved by properly selecting the flow nodes and efforts nodes as interfaces for plugging components with each others by edges. For example, the choices for multibody applications would be torque and angular velocity, respectively.

Proposition 3.3 The physically universal modeling concept allows the modeling of systems exhibiting multi-physical phenomena.

Diving more into details, this modeling concept corresponds to acausal relation among system variables with no notions of inputs-output relationship. Practically, modeling becomes the matter of dragging, dropping and connecting icons together to form up an **object diagram** that looks very similar to the system we would like to imitate. It is straightforward to imitate a modification to a system by correspondingly altering an object diagram with additional or alternative components using a graphical editor.

The bond graph concept converged to a stand-alone discipline in the Systems Engineering domain (Filippo et al. 1991). Many modeling frameworks for modeling dynamical systems have been established very early, cf. (Filippo et al. 1991) for a demonstration of such prototyping frameworks based on the bond graph concept. The majority of these frameworks were composed of a set of FORTRAN routines where the modelers need to specify subsystems and components using a set of line codes.

3.2 The evolve of the Modelica language

3.2.1 In the beginning it was Dymola

One may argue that a cumbersome exploitation and extension of the bond graph concept was realized by the Dymola (DYnamic MOdeling LAguage) invented by Hilding Elmqvist in a PhD thesis (Elmqvist 1978) late seventies. Dymola, viewed as the main predecessor of the Modelica language (Åström, Elmqvist, and Mattsson 1998), was a stand-alone modeling language with object-oriented facilities allowing the description of a system in terms of hierarchical composition of subsystems and classes.

Dymola enabled equations prototyping rather than assignments[1] for describing the physical behavior of a model. Dymola compiler engine made use of graph algorithms and symbolic computation for converting an object diagram to a numerically integrable ODE system.

Nevertheless, a fully-reliable acausal modeling concept desired progress both at hardware

[1] An equation describes an acausal relationship between variables that need to be fulfilled concurrently where as an assignment is an explicit causal relation between an output variable and a set of input variables

technology and methodology levels.

3.2.2 Hardware technologies

The progress in hardware technology allowed acausal modeling languages to become more reliable in terms of model size for practical applications. Even a simple model can be compiled into a high-dimensional equations system that hardware technologies at early time were not mature enough for a reliable evaluation, yet. Interested readers can further consult (Elmqvist 2014) for more details.

3.2.3 Compiler methods

Further methodological progress for establishing algorithms capable of efficient simulation of large-scale equation systems was required, for instance:

- Fundamental concepts for efficient equation-based compiler techniques by which a very-large scale equation system is simplified to a much smaller one by removing trivial equations and conducting simple symbolic manipulation (Cellier and Kofman 2006; Maffezzoni, Girelli, and Lluka 1996; Murota 1987)
- Methods for structural solvability analysis by which a large-block of an equation system is decomposed into smaller causal cascaded blocks of equation systems solved in a sequentially faster way (Cellier and Elmqvist 1993; Leitold and Hangos 2001).
- Resolving algebraic loops in-which inter-dependencies of system variables arise. Here, tearing algorithms are applied for solving sparse large-scale non-linear equation systems (Elmqvist and Otter 1994; Kron 1963)

A detailed coverage of many of the above topics can be consulted in (Cellier and Kofman 2006).

3.2.4 The differential index problem

Typical models correspond to a mixture of Differential and Algebraic Equations (DAE). The numerical treatment of such equation systems desired a special care. The straightforward applications of known ODE solvers to such equation system may cause the numerical solution to draft far away from the true solution.

The reason is that implicit dependencies among state variables may arise. The inherent degree of such implicit algebraic constraints is referred to as **the differential index**.

However, for a DAE system of arbitrary index, methodological progress for its numerical integration demanded further progress:

- **Index reduction** algorithms by which a high-index differential algebraic equations is transformed to a numerically solvable DAE system of index one or an ODE system (Mattsson and Söderlind 1993; Pantelides 1988)

- **Consistent initialization** of DAE by which consistent start values of the DAE needs to be identified through solving a nonlinear equation system using a Newton-iteration scheme.

The later can be a challenging problem due to the influence of initial guesses on identifying a reliable solution. Thus, advanced strategies are needed s.a. (**Bachmann2015**; Sielemann and Schmitz 2011). This is still an active area of research (Casella and Bachmann 2021).

3.2.5 And then Modelica came

By mid nighties many modeling languages existed established from several working groups (Åström, Elmqvist, and Mattsson 1998; Marquardt 1996). This caused not only that each language or modeling tool brought its own strength and weakness, but also significant duplicated efforts were not avoidable. Model exchange and reuse among different tools were not possible. In other words as stated by (Elmqvist 2014):

> ... Modeling requires reuse of stored knowledge, i.e. there must be a standard language. It does not make sense that various tool vendors invent their own language and that a new language is created for every Ph.D. thesis on modeling ...

In order to unify the splintered activities within the modeling community, an initiative was made for collaborative efforts to design a unified standard modeling language. This initiative suggests to include the **commonly-agreeable** best and state-of-the-art of established modeling concepts (Elmqvist and Mattsson 1997) including

- Support for both of the **object diagram** (i.e. acausal) and the **block diagram** (i.e. causal) approaches
- **Object-oriented paradigms** facilitating the ability to hierarchically decompose a system model into many reusable encapsulated components.
- **Equation-based semantics** (Augustin et al. 1967) [2] as the main building blocks for implementing model components via hybrid DAE

These efforts have converged to the Modelica language with the first open-source specification of the Modelica language published 1997, cf. Modelica Association (MA)

`https://modelica.org/`

3.3 Advantages of the Modelica language

Based on the previous demonstration of modeling concepts, we summarize some of the clear advantages. Due to the adopted features, particularly the ultimate acausal modeling concept, cf. Figure 3.1, the Modelica language provides several obvious benefits for

[2] Early versions allowed explicit representation of equations. Sorting and symbolic algorithms later allowed implicit representation, e.g. in DYMOLA

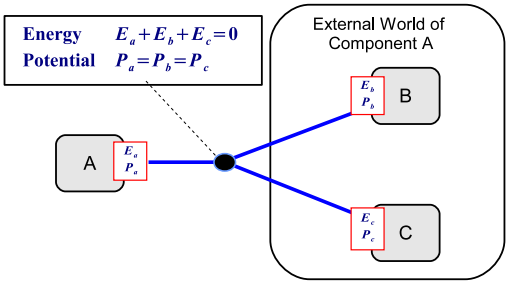


Figure 3.1: Acausal modeling concepts reflecting standard universal lows of energy conservation are used to express the interactions between a system component A and the external world (components) B, C.

modeling applications according to scientifically proposed metrics, cf. (Abel 2010; Pollok, Klöckner, and Zimmer 2019; Schweiger et al. 2020b; Wetter and Haugstetter 2006):

1. An object diagram topologically resembles the physical system
2. It is easier to modify a system and reuse model components
3. rapid prototyping is efficiently more productive
4. Multi-domain physical phenomena are easier to capture and describe

For instance, following the causal modeling approach, a power network is usually modelled using the traditional admittance matrix method. In contrary, with the acausal modeling approach, the power system topology can be visualized and modified in a graphical editor (Larsson 2000).

Analogously, an implementation of synchronous machines in the block diagram based tool LabView results in a graphical model composed of interconnected I/O blocks each describing an (a set of) equation(s) (Karnavas and Lygouras 2018). However, the topology of the underlying system is not directly viewable in the graphical model. On the other hand, for a similar application realizing rotating machines in Modelica with a completely transparent, accessible and modifiable equations (Kral and Haumer 2011a), the topology of rotating machines is reflected in the graphical editor.

II Designing a Modelica library

4. Basic concepts

In (Bartolini, Casella, and Guironnet 2019), the Modelica library `PowerSystems` (Franke and Wiesmann 2014) is briefly evaluated as follows:

> The very high level of abstraction, coupled with an extensive use of inheritance, allows to accommodate a very wide scope in a unified framework spanning DC, one- and three-phase AC in both quasi-static and dynamic regimes. The end result is a code base which is very general, elegant and concise; however, the price to pay for these features is that the code is quite difficult to comprehend for non-Modelica experts.

While we agree with the above statement, it encourages us to carry out two challenging but adventurous missions oriented towards non-Modelica experts:

1. initiating a first impression of Modelica syntax and features
2. making it easier to understand and deal with the `PowerSystems` Modelica library as well as other open-source Modelica libraries

Some capabilities of the `PowerSystems` library is demonstrated in a step-by-step manner simultaneously clarifying the features of Modelica language that make power system modeling a matter of fun. In this journey, we place ourselves in the head of a power system Modelica modeler and attempt to imitate the way he is tackling the design and implementation challenges.

4.1 Equation identifiers

Question 4.1 How are identifiers of an equation system characterized?

Mathematical **identifiers** within an equation system describing a power system are classified under three types of **variability**:

1. Time-dependent **variables** whose values can change during a dynamic simulation course
2. Input **parameters** whose values are constant during a simulation but can change from a simulation to another
3. **Constants** whose values are fixed and don't change at all

Question 4.2 What are Modelica built-in types? and how to declare equation identifiers upon them?

Modelica provides language keywords for differentiating between the possible levels of identifiers variability at declaration level. **Declaration of identifiers** may look as follows:

```
Real v                    "Voltage";
Real i                    "Current";
parameter Real  R = 1.0   "Resistance";
constant  Integer n = 1   "# of independent voltage and
                                  current components";
constant  Integer m = 0   "# of reference angles";
```

Listing 4.1: Declaration

The following remarks illustrate the previous code:

1. The keywords `Real` and `Integer` are **built-in types**
2. The keywords `parameter` and `constant` are used for distinguishing parameters and constants from variables.
3. Any declaration can be briefly **commented** with the string `"comment ..."`.

Common simulation environments generate **HTML-based documentation** that lists all declared identifiers of all components of a library in an organized tabular form.

4.2 Physical units

Question 4.3 How to construct physical unit types based on built-in types?

An essential requirement for physical modeling is the capability of associating **physical units** to identifiers rather than declaring them using common **standard data types**. This can be realized by defining physical units as follow:

```
type Voltage = Real(final unit = "V",
                    final quantity = "Voltage");
```

```
type Current = Real(final unit="A",
                    final quantity="Current");
```

Listing 4.2: Implementation of physical types

Here, a physical-unit type is a real-valued type associated with a physical unit using **the attribute** *unit* of type `Real`. With the above types, variables can be declared using these types:

```
Voltage v "Voltage";
Current i "Current";
```

Listing 4.3: Declaring a voltage variable

Consequently equations can be physically validated through **unit-checking** at compile time by common simulation environments. Other integer-valued physical types are also declarable but with the keyword `Integer`.

Question 4.4 How to construct constrained types?

Moreover, **constrained types** with maximum and/or minimum values can be associated to provide a mechanism of error reporting if a variable violates these constraints during simulation runtime. This is done by making use of the attributes `min` and/or `max`.

Question 4.5 Is there a reusable packages of standard physical units?

Usually, most of common physical units are provided within **the Modelica Standard Library** (MSL), introduced in Subsection 7.2, within the `Modelica.Units` package.

The concept of a **package** is going to be clarified in a moment. The name `Modelica` stands for MSL while `Units` stands for the package of physical units within MSL.

Thus, explicit implementation of physical units is usually not needed. However, the modeler can still implement his own specific types with further own attributes and special physical units.

R More in-depth details related to the last two sections can be consulted under

https://mbe.modelica.university/behavior/equations/variables/

4.3 Packages

Question 4.6 How to organize related model information, types, functions, components etc. together?

A `package` in Modelica resembles the *class* concept in common object-oriented languages though a package is not instantiated by declaring an object. A Modelica package is usually composed of a set of attributes. An attribute can be a constant, a function, a declarable component among other possibilities. Declaring a Modelica library named `PowerSystems` is realizable by declaring a top package as follows:

```
package PowerSystems
 ...
 import Modelica.Units.SI.*;
 ...
 // implementation of the library takes place here
 ...
end PowerSystems;
```

Listing 4.4: Declaring the library PowerSystems

Question 4.7 How to make use of existing types within packages?

The **import statement** implies that all physical types within the MSL package `Modelica.Units.SI` can be directly employed across all components of the library:

```
Voltage v;
// instead of Modelica.Units.SI.Voltage v;
```

Listing 4.5: Declaration of Voltage

Alternatively, since the modeler may like to implement his own non-SI physical units, e.g. different than those within MSL, it is proper to keep it clear whether a physical unit is SI-standard or an own declared physical unit:

```
import Units = Modelica.Units.SI;
...
    Units.Voltage v;
...
```

Listing 4.6: Declaration of Voltage

Here the package `Units` is a dummy replacement of the long package name `Modelica.Units.SI` that otherwise needs to be associated to every declared variable. It is straightforward to employ different own-implementation of physical units package, e.g.

```
// import Modelica.Units.SI.*;
import PowerSystems.Units.*;
...
    Voltage v;
...
```

Listing 4.7: Declaration of Voltage

though in this case the own package `PowerSystems.Units`:

1. should provide an implementation for all physical units employed across the library
2. does not need to follow SI standards

Mutual employment of physical units in both packages is possible with proper usage of dot notations.

4.4 Organization of packages and subpackages

Question 4.8 How to realize multiple levels of packages?

It is common for a library to be composed of multiple levels of packages[1]. Assuming that the `PowerSystems` library includes the packages `Control` and `Units` among others, and that the package `Control` includes the packages `Exciters` and `Governers`, then the corresponding textual representation may look as follows:

```
package PowerSystems
  ...
  package Control
    ...
        partial package Exciters
          ...
          // models for exciters come here
          ...
        end Exciters;

        package Governers
          ...
        end Governers;
        ...
  end Control;
  ...
  package Units
    ...
    // declaration of physical types
    ...
  end Units;
  ...
end PowerSystems;
```

Listing 4.8: Package hierarchies

There is no need to implement hierarchies of several packages in a textual way. Common Modelica simulation environments provide the capabilities for flexible creation of packages and subpackages using a GUI-based editor.

[1]The concepts of library and package are interrelated. We can view a library as a top package with various internal packages. When a package is somehow a stand-alone set of components and further internal packages out of which realistic models can be built, then we can view it as a sublibrary.

Another significant remark is that the textual implementation of a library does not need to take place in one-single file. It is actually favourable from a software engineering and model design perspectives to separate the implementation into multiple files. A typical single ".mo" file may correspond to a package or even a separate model component.

R Further in-depth implementation details of packages and import statements can be consulted under:

https://mbe.modelica.university/components/packages/package_def/

https://mbe.modelica.university/components/packages/importing/

4.5 Connections

First of all, in the sake of simplicity, a power system with **one phase-system**, i.e. direct current, is initially assumed. The generalization to other multi-phase systems is discussed in the next Section.

Before implementing power system components (i.e. lines, loads, generators etc.), it is intuitive to first consider the way these components get plugged into each other in an object diagram. In other words, having two icons of object components in a GUI-based object diagram editor, the question is:

Question 4.9 *How to establish a physically well-defined connection mechanism enabling a physically-meaningful connection line between two icons in an object diagram?*

In Modelica, to make a component A connectable (either textually or graphically) to other components (say B and C), all components should include **(a) connector(s)** acting as attached communication port(s), cf. Figure 3.1. A typical definition of a `connector` specifies the variables of a model component (A) that:

- externally influences
- internally influenced by

the external world, i.e. the connected components B and C. All components, i.e. A, B and C should share at least an identical `connector` type. In the context of a power system, a definition of a DC-connection between typical components (e.g. lines, loads, generators, etc.) may look as follows:

```
connector Terminal "DC terminal"
  Voltage v        "Voltage";
  flow Current i   "Current";
end Terminal;
```

Listing 4.9: Implementation of a DC-connector

Typically, two types of variables are usually[2] specified:

1. **across or potential variable(s)**, i.e. voltage
2. **through or flow variable(s)**, i.e. current

In the object diagram of Figure 3.1 a **connection set between the components** A, B and C is established. Assume that `T` is a connector of type `Terminal` associated with the mentioned components. The connection set can be established in Modelica:

```
...
equation
...
  connect(A.T,B.T);
  connect(A.T,C.T);   // or connect(B.T,C.T)
...
```

Listing 4.10: the connection set of Figure 3.1

The `connect` statement expects two arguments, each is an instance of a `connector`. The order of the arguments is irrelevant. The two types of connectors have to be logically similar to each other, i.e. **physical compatibility**.

Mathematically, the connection between the potential variables of connected components resembles an identity equation:

$$A.T.v = B.T.v = C.T.v \tag{4.1}$$

Meanwhile, the flow variables of connected components, distinguished by the keyword `flow`, result in a sum-to-zero equation:

$$A.T.i + B.T.i + C.T.i = 0 \tag{4.2}$$

These two equations correspond to the known Kirchhoff's laws for voltage and current.

In other words, these mathematical equations express the exchange and sharing of relevant information among model components. In case a connector does not include a flow variable, it is a **causal connector**, otherwise an **acausal connector**. In the later case, the modeler is absolutely not concerned about the direction of information exchange, as it might be but not necessarily the case with causal connectors.

More details on special causal connectors:

https://mbe.modelica.university/components/connectors/block_connectors/

[2] There could be less but also more than two types as there are other connection mechanisms particulary useful for experssing reversed flow (Franke et al. 2009b).

A model component can get connected to another model component only through their connectors. Their connectors have to be of the same (or logically similar) type (say via a connector of type `Terminal` for each component). If there are two physically-distinct definitions of two types of connectors, then they can not be connected to each other.

R More details on connectors:

https://mbe.modelica.university/components/connectors/

https://mbe.modelica.university/components/connectors/connector_def/

Models are typically constructed in a graphical way. Hence, connectors are also graphically distinguished and are associated with user-defined icons. These icons can be graphically attached with component icons to make these components connectable.

R More details on graphical connectors

https://mbe.modelica.university/components/connectors/graphics/

4.6 Model components

Question 4.10 How to realize common model components?

After establishing a well-defined physical connection mechanism, it is possible to implement typical components of an electric network. Given that in an object diagram the flow of current from left to right through an arbitrary common component is conventionally defined to be positive, cf. Figure 4.1. An implementation of `Resistor` may look as

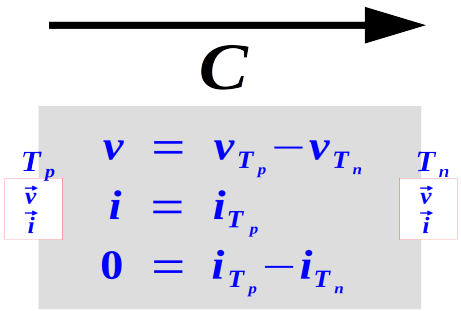


Figure 4.1: The flow of current and the voltage difference within a component C with two terminals T_p and T_n. The internal voltage and current through the component C can be defined in terms of those of the terminals.

follows:

```
1  model Resistor
2    Terminal  T_p "terminal from left";
3    Terminal  T_n "terminal from right";
4    parameter Resistor R = 1.0 "Resistance";
5    Voltage   v   "Voltage";
6    Current   i   "Current";
7  equation
8    v   = T_p.v - T_n.v;
9    i   = T_p.i;
10   0   = T_p.i + T_n.i;
11   R*i = v "Ohm's law";
12 end Resistor;
```

Listing 4.11: Implementation of a Resistor component

The implementation of the `Resistor` component is realized as follows:

1. Line 1: the keyword `model` is employed to provide a definition for the `Resistor` component. This is one of the most common ways of defining a component among other possible approaches. A model is mainly composed of two parts: declaration part (lines 2-6) and equation part (lines 7-11)
2. Lines 2 and 3: two connectors (positive and negative) are attached to the `Resistor` component. These terminals can get connected to terminals of other components from the left and/or the right sides, correspondingly
3. Lines 4-6: basic declarations of resistance, voltage and current
4. Line 7: the equation section is declared using the keyword `equation`
5. Lines 8-10: the relationship between the terminal variables and the internal behavior of `Resistor` component is mathematically defined, cf. Figure 4.1
6. Line 11: the physical behavior (i.e. Ohm's law) is implemented using an implicit equation
7. Line 12: as in most cases, a model definition ends with the keyword `end` followed by the name of the model.

The overall model apparently results in three equations in four unknown. This is a **component model** (or a **subsystem model**) which is meaningless to attempt to simulate it as a stand-alone simulation program without including it in a physically meaningful **system model**, e.g. an electric network.

R More on model definitions

https://mbe.modelica.university/behavior/equations/model_def/

R More on equations can be consulted at

https://mbe.modelica.university/behavior/equations/equations/

4.7 A simple electrical network

Question 4.11 How to setup an electrical circuit using the implemented components library?

Assuming that in a power system network, two lines get connected, then this configuration results in six equations in eight unknowns. The missing two equations are implicitly present within the connection, cf. Equations (4.1) and (4.2). They are automatically generated by the employed Modelica compiler. With these two equations the model is **balanced** and is equivalent to a **consistent equation system**. The connection of two lines can be described as follows:

```
model ElectricNetwork
  ...
  Resistor line1(R=2.0);
  Resistor line2(R=4.0);
  ...
  // other components
  ...
equation
  ...
  connect(line1.T_n , line2.T_p);
  ...
  // other connections
  ...
end ElectricNetwork;
```

Listing 4.12: Connecting two lines

The `connect` statement can be either textually implemented or generated using a GUI-based editor. The latter way is simply accomplished by dragging, dropping two components (say of type `Resistor`) in an object diagram editor and connecting them together through their internal connectors.

The last model is a **system model** composed of **subsystem models** (**component models**). This system model is expected to simulate using a particular Modelica simulation environment.

R General issues on acausal modeling

https://mbe.modelica.university/components/components/component_models/

R Useful details on system models and connections:

https://mbe.modelica.university/components/components/system_models/

More details on subsystems:

https://mbe.modelica.university/components/subsystems/sub_interface/

.

.

5. Object-Oriented features

Modelica adopts many common object-oriented features that simplify and enable a powerful system design. In this chapter we overview some of these features that result in a flexible implementation of a power system components library enabling generic realization of system models with arbitrary phase systems.

5.1 Abstract models and inheritance

Question 5.1 What is an abstract model? What is it useful for?

A common object-oriented feature is the capability of introducing a **generic abstract component** from which several specific declarable components are derived, i.e. inheritance from abstract components[1].

The first three equations in the implementation of `Resistor` in Listing 4.11, is the same in most components of a power network. In contrary, the implementation of a transformer should convert an input current to another current value according to a given ratio.

In order to avoid such repeated set of equations in multiple implementations of various components, a possible object-oriented design approach can make use of **an extendable abstract model** for gathering such equations. An abstract model is an **unbalanced model** with an under-determined set of equations (i.e. more unknowns than equations):

[1] Inheritance in Modelica can be also realized through non-abstract components

```
model PartialTwoTerminals
  Terminal  T_p  "terminal from left";
  Terminal  T_n  "terminal from right";
  Voltage   v    "Voltage";
  Current   i    "Current";
equation
  v   = T_p.v - T_n.v;
  i   = T_p.i;
  0   = T_p.i + T_n.i;
end PartialTwoTerminals;
```

Listing 5.1: implementation of an abstract model with two terminals

The keyword `partial` defines the model `PartialTwoTerminals` to be abstract. This prevents a modeler from declaring a `PartialTwoTerminals` instance component in a system model.

Question 5.2 How to inherit from abstract models?

Models such as `Resistor` and `Capacitor` can extend `PartialTwoTerminals`. In this way, they provide an implementation only of their physical behaviors, e.g.

```
model Capacitor
  extends PowerSystems.Ports.PartialTwoTerminal;
  parameter Capacitance C = 1.0 "reactive component";
equation
  C * der(v) = i;
end Capacitor;
```

Listing 5.2: Implementation of Capacitor

The `extends` statement specifies that the model `Capacitor` inherits the base abstract model `PartialTwoTerminal`. The operator `der` expresses the time derivative, i.e. `der(v)` corresponds to dv/dt. The `Resistor`, `Conductor`, etc. models are (re-)implemented in a similar way.

5.2 Arbitrary phase systems by an abstract packages

Question 5.3 What is an abstract package? What is it useful for?

Another feature of common object-oriented paradigms is the capability of providing generic templates out of which specialized classes can be easily and flexibly realized. In Modelica, **generic packages** provides such powerful capabilities.

For power systems modeling, this provides the opportunity to realize generic implementation of power systems components parametrizable via phase systems. This enables a unified implementation of power system components, e.g. lines, for all possible phase systems. A **generic phase system package** is as follows:

```
partial package PartialPhaseSystem  "Base package of all
                                       phase systems"
  ...
  constant String phaseSystemName = "UnspecifiedPhaseSystem";
  constant Integer n "Number of independent voltage and
                                    current components";
  constant Integer m "Number of reference angles";
  ...
end PartialPhaseSystem;
```

Listing 5.3: Implementation of generic phase systems

The keyword `partial` implies that this abstract package has undefined attributes. These attributes are subject to specification within extended non-abstract packages.

Question 5.4 How to extend from an abstract package?

For instance, a specification is provided by the package `DirectCurrent`:

```
package DirectCurrent  "Direct Current"
  extends PartialPhaseSystem(
            phaseSystemName = "DirectCurrent",
            n=1, m=0);
  ...
end DirectCurrent;
```

Listing 5.4: Implementation of Direct Current phase system

Similarly, a three-phase system in dq0-representation is realized as follows:

```
package ThreePhase_dq0  "AC system symmetrically loaded
                           three phases"
  extends PartialPhaseSystem(
            phaseSystemName = "ThreePhase_dq",
            n=3, m=2);
  ...
end ThreePhase_dq0;
```

Listing 5.5: Implementation of dq0-based AC system

The realization of further phase systems, e.g. dq-representation with $n = 2$ and $m = 2$, can be consulted from the available implementation of the library.

5.3 Function interfaces

Question 5.5 What is an abstract function? what is it useful for?

The realization of power system components depends on the specific phase system on which they operate. The implemented equations can be expressed in terms of common quantities s.a. active power, system voltage, system current, among others. However, such

quantities are differently computed according to the phase system they are operating on.

Thus, it is desired to associate an arbitrary generic phase system with **functions** which compute such mentioned quantities. In this way, the mathematical behavior of many components regardless of the phase system is uniquely expressed in terms of these functions. Hence, a generic implementation of model components becomes realizable.

This is realizable by embedding **interfaces**, i.e. **abstract functions**, in the abstract package `PartialPhaseSystem` (cf. Listing 5.3). In this way, any specific phase system, e.g. `DirectCurrent`, is enforced to provide a specific implementation to these interfaces:

```
1  partial package PartialPhaseSystem
2    ...
3    partial function j "Rotation by 90 degree"
4      input  Real x[:];
5      output Real y[size(x,1)];
6    end j;
7
8    partial function thetaRef "Absolute angle of rotating
                                    reference system"
9      input  Angle theta[m];
10     output Angle thetaRef;
11   end thetaRef;
12
13   // other interfaces for systemVoltage, systemCurrent,
14   // phasePowers, phaseVoltages, etc.
15   ...
16
17   partial function activePower "Total power"
18     input  Voltage v[n]  "phase voltages";
19     input  Current i[n]  "phase currents";
20     output ActivePower P "active system power";
21   end activePower;
22   ...
23 end PhaseSystem;
```

Listing 5.6: Interfaces for arbitrary PhaseSystems

In contrast to the implementation within an `equation` section (cf. Listing 4.11), **functions are causal** and evaluates a set of output quantities from a given set of input quantities. The following notes of Listing 5.6 are highlighted:

1. The `partial function` keywords declare an interface (lines 3,8 and 17)
2. The `input` keyword declares the input(s) of a function (lines 4,9,18 and 19)
3. The `output` keyword declares the output(s) of a function (lines 5,10 and 20)
4. Arrays are declared using brackets, e.g. `theta[m]` line 9 where m is to be specified by a specific implementation of a phase system, e.g. `DirectCurrent` (cf. Listing 5.4)
5. The declaration `x[:]` (line 4) specifies that the function `j` accepts an input array of

arbitrary length

6. The built-in function `size(T,k)` (line 5) evaluates the length of the multidimensional tensor `T` along the *k*-th dimension
7. Thus, the declaration `y(size(x,1))` (line 5) specifies that the output vector `y` should be of the same length as the input vector `x`

R More details on arrays can be consulted under

https://mbe.modelica.university/behavior/arrays/#review

5.4 Implementation of function interfaces

Question 5.6 How to implement abstract functions?

Realization of the above functions for a component operating on a direct current phase system is trivial due to the absence of reference angles:

```
1  package DirectCurrent
2    extends PartialPhaseSystem(...);
3    ...
4    // Direct current has no complex component
5    function j "Rotation by 90 degree"
6      extends PartialPhaseSystem.j;
7    algorithm
8       y := zeros(n);
9    end j;
10
11   // No absolute angle for DC
12   function thetaRef "Absolute angle of rotating reference
        system"
13     extends PartialPhaseSystem.thetaRef;
14   algorithm
15     thetaRef := 0;
16   end thetaRef;
17   ...
18   function activePower "Total power"
19     extends PartialPhaseSystem.activePower;
20   algorithm
21     P := v*i;
22   end activePower;
23   ...
24 end DirectCurrent;
```

Listing 5.7: Implementation of interfaces for DC

1. The implementation of functions `j` (lines 5-9) and `thetaRef` (lines 12-15) are trivial for the phase system `DirectCurrent`

2. A function inherits the declarations from **the base function** by the `extend` statement (lines 6, 13 and 19)
3. The implementation of a function takes place in an `algorithm` section (e.g. lines 20-22) rather than an `equation` section
4. In an `algorithm` section the **assignment notation** `:=` (e.g. line 21) is employed, in contrary to the common equation notation `=` in an `equation` section
5. The built-in function `zeros(n)` (line 8) assigns a one dimensional zero vector to the output vector `y`

Non-trivial implementation of such functions is carried out in e.g. AC phase system in dq0-representation:

```
1   ...
2   function j "Rotation(pi/2) of vector around {0,0,1}"
3     extends PartialPhaseSystem.j;
4   algorithm
5     y := cat(1, {-x[2], x[1]}, zeros(size(x,1)-2));
6   end j;
7
8   function thetaRef "Absolute angle of rotating reference
       system"
9     extends PartialPhaseSystem.thetaRef;
10  algorithm
11    thetaRef := theta[2];
12  end thetaRef;
13  ...
14  function activePower "Total power"
15    extends PartialPhaseSystem.activePower;
16  algorithm
17        P := v[1:2]*i[1:2];
18  end activePower;
19  ...
```

Listing 5.8: Implementation of interfaces for DC

1. The function `cat(k,...)` concatenates multiple arrays along the k-th axis
2. Active power is calculated using a dot product of the first two entries from the vectors v and i (line 17)

R More details on functions:

https://mbe.modelica.university/behavior/functions/#review

R More details on model interfaces

https://mbe.modelica.university/components/architectures/int_vs_imp/

5.5 Generic connectors

Question 5.7 What are generic connectors?

Having established a generic package of arbitrary phase systems as well as specialized realization of specific phase packages, it is desirable to provide generic implementation of connectors and common model components (cf. next Section). This is realizable by parameterizing connectors and components with s.c. **replaceable packages**. The actual implementation of the connector `Terminal` in Listing 4.10 looks as follows:

```
1 connector Terminal "General power terminal"
2   replaceable package PhaseSystem =
3     PhaseSystems.PartialPhaseSystem "Phase system";
4   Voltage v[PhaseSystem.n]       "Voltage vector";
5   flow Current i[PhaseSystem.n] "Current vector";
6   ReferenceAngle theta[PhaseSystem.m]
7     if PhaseSystem.m > 0  "Phase angles vector";
8 end Terminal;
```

Listing 5.9: Implementation of generic connector

1. This is a generic connector parameterized by the replaceable package *PhaseSystem* assigned with the value `PartialPhaseSystem` (lines 2 and 3)
2. The voltage and the current are declared as arrays of size `PhaseSystem.n` components (lines 4 and 5)
3. Such generic connector cannot be instantiated in a realistic model component because
4. The value of n within the abstract package `PhaseSystem` is initially not specified
5. Thus, the keyword `replaceable` indicates that a re-declaration of the connector is desired to provide a well-defined **instantiatable connector**
6. The `if` condition expresses a **conditional declaration** that takes place only if `PhaseSystem.m` is positive (lines 6 and 7)

Question 5.8 How to establish phase-system-specific connector from a generic connector?

An AC-specific connector in dq0-representation is declared as follows:

```
Terminal T(redeclare package PhaseSystem =
                               PhaseSystems.ThreePhase_dq0);
```

Listing 5.10: Declaration of a specific connector

The keyword `redelare` modifies the value of `PhaseSystem` to a specific phase system, i.e. `PhaseSystems.ThreePhase_dq0`. Since such a specific connector would be often needed, it is adequate to provide a definition for it:

```
connector Terminal_AC3dq
  extends Terminal(redeclare package PhaseSystem =
```

```
                                    PhaseSystems.ThreePhase_dq);
```

Listing 5.11: Declaration of a specific connector

This connector can be then directly employed in the declaration part of common components operating on AC phase system in dq0-representation.

5.6 Generic components

Question 5.9 What are phase-system generic components?

Similarly to generic and specific connectors, the realization of **generic and specific components** takes place. Firstly, A DC-specific partial model `PartialTwoTerminals` in Listing 5.1 is generalized to arbitrary phase systems as follows:

```
1  partial model PartialTwoTerminal
2    replaceable package PhaseSystem =
       PhaseSystems.PartialPhaseSystem;
3    Terminal T_p(redeclare package PhaseSystem = PhaseSystem);
4    Terminal T_n(redeclare package PhaseSystem = PhaseSystem);
5    Voltage v[PhaseSystem.n];
6    Current i[:];
7  equation
8    v = T_p.v - T_n.v;
9    i = T_p.i;
10   zeros(PhaseSystem.n) = T_p.i + T_n.i;
11   if PhaseSystem.m > 0 then
12     T_p.theta = T_n.theta;
13   end if;
14 end PartialTwoTerminal;
```

Listing 5.12: Generic component with two terminals

1. Within the `redeclare` modifiers (lines 3 and 4), the first occurrence of `PhaseSystem` refers to the re-declarable package in the connector `Terminal` (cf. Listing 5.9)
2. The second `PhaseSystem` refers to the re-declarable package of `PartialTwoTerminal` (line 2)
3. Equations are expressed in terms of vectors (lines 8-10)
4. The last **conditional equation** (line 11-13) is considered only if `PhaseSystem.m` is positive

Generic phase-system independent components are designed in a similar fashion. The component `Resistor` in Listing 4.11 is renamed and generalized to `Impedance`:

```
1  model Impedance
2    extends PowerSystems.Ports.PartialTwoTerminal;
3    parameter Resistance R = 1 "active component";
4    parameter SI.Inductance L = 1/314 "reactive component";
5    AngularFrequency omegaRef;
6  equation
```

```
7    if PhaseSystem.m > 0 then
8      omegaRef = der(PhaseSystem.thetaRef(T_p.theta));
9    else
10     omegaRef = 0;
11   end if;
12   v = R*i + omegaRef*L*j(i);
13   if PhaseSystem.m > 0 then
14     T_p.theta = T_n.theta;
15   end if;
16 end Impedance;
```

Listing 5.13: Implementation of impedance

Careful readers should be capable of undrestanding the previous implementation without further illustration. Phase system-specific components can be declared or designed in similar way as in Listings 5.10 and 5.11.

More details on system re-configuration:

https://mbe.modelica.university/components/architectures/replaceable/

6. Examples

6.1 A power flow study

Question 6.1 How to setup a power network model using the implemented components library and perform a load flow study?

The first example demonstrates a typical **load flow** study of a power network model (Oeding and Oswald 2016) using the OpenModelica simulation environment (Fritzson et al. 2020). Having accomplished the implementation of basic components of a power system library, a typical power system dynamical or statical network model can be constructed by two ways:

1. Either by using a graphical modeling approach by dragging, dropping and connecting components together in the graphical model editor called OMEdit (Ashgar and al. 2011), cf. Figure 6.1, resulting in the model diagram of Figure 6.2
2. or by assembling the network model in a textual manner, cf. Listing 6.1

```
1 model NetworkLoop
2
3   FixedVoltageSource fixedVoltageSource1(V=10e3)
4   Impedance line1(R=2, L=0)
5   Impedance line2(R=4, L=0)
6   Impedance line3(R=2, L=0)
7   Impedance line4(L=0, R=1)
8   Impedance line5(L=0, R=3)
9
```

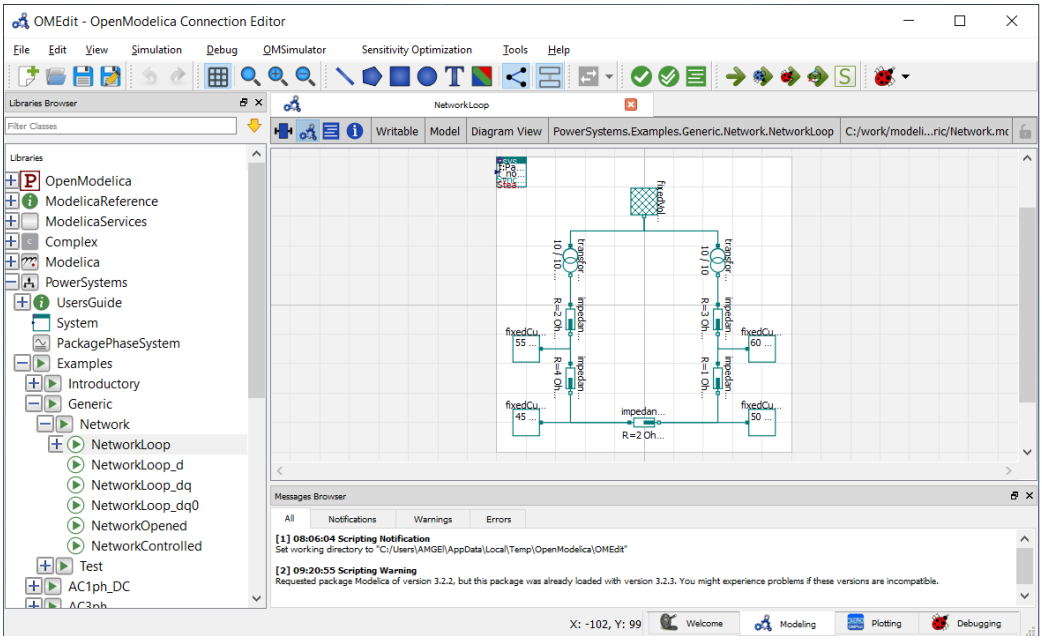


Figure 6.1: The graphical and textual modeling editor OMEdit: a front-end for the OpenModelica simulation environment. Modelers are capable of choosing between several solvers and adjusting their settings. The editor includes a built-in GUI-based debugger and profiler with features for breakpoints, single-stepping, changing variable values, demonstrating the different levels of symbolic transformation of the equation system, runtime profiling, runtime error diagnostic among other features (**Sjolund2015**). Advanced plotting capabilities including animation exist (Waurich and Weber 2017).

```
10    FixedCurrent fixedCurrent1(I=55);
11    FixedCurrent fixedCurrent2(I=45);
12    FixedCurrent fixedCurrent3(I=50);
13    FixedCurrent fixedCurrent4(I=60);
14
15    VoltageConverter transformer1(ratio=10/10.4156);
16    VoltageConverter transformer2(ratio=10/10);
17
18  equation
19
20    // lines
21    connect(line1.T_n , line2.T_p);
22    connect(line2.T_n , line3.T_p);
23    connect(line4.T_p , line5.T_n);
24    connect(line4.T_n , line3.T_n);
25
26    // fixed current & voltage sources
27    connect(fixedCurrent1.T , line1.T_n);
28    connect(fixedCurrent2.T , line3.T_p);
29    connect(fixedCurrent3.T , line3.T_n);
30    connect(fixedCurrent4.T , line5.T_n);
31    connect(fixedVoltageSource1.T , transformer1.T_p);
32
33    // transformers
```

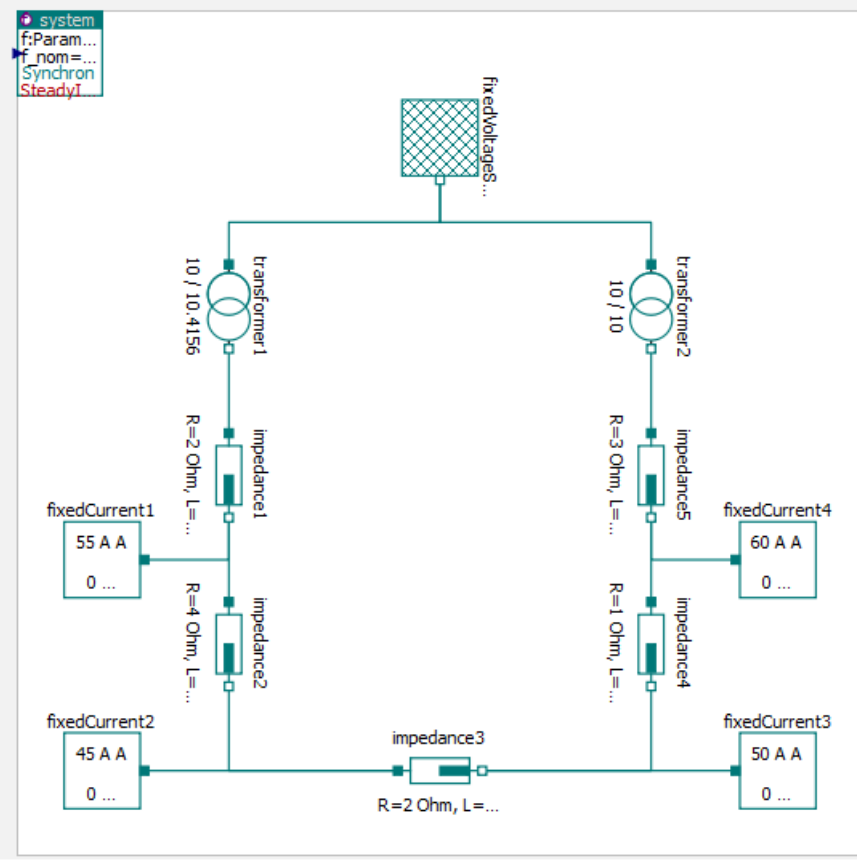


Figure 6.2: A typical power system network *PowerSystems.Examples.Network.NetworkLoop* composed of fixed voltage source (the shadowed square), fixed current sources (the squares), transformers (two intersected circles) and impedances (dashed rectangles)

```
34    connect(transformer1.T_n, line1.T_p);
35    connect(transformer2.T_n, line5.T_p);
36    connect(transformer2.T_p, fixedVoltageSource1.T);
37
38  end NetworkLoop;
```

Listing 6.1: Implementation of network model of Figure 6.2

This textual representation is automatically generated by the editor when constructing the network in graphical manner. The network model assumes that the component are associated with a specific phase system, say AC in dq0-representation.

Nevertheless, the network model can be also generic in terms of phase systems and thus should be declared as `partial`. By assuming that the network model in Listing 6.1 is additionally parameterized with a redeclarable phase system package, cf. the previous Chapter, different models in different representations can be easily established:

```
model NetworkLoop_dq "NetworkLoop example with phase system
   ThreePhase_dq"
```

```
  extends NetworkLoop(
    redeclare replaceable package PhaseSystem =
                PowerSystems.PhaseSystems.ThreePhase_dq);
end NetworkLoop_dq;
```

Listing 6.2: Implementation of network model in dq representation

Having constructing the network model with the set of parameter values in Listing 6.1 with inductance being neglected, the compilation of this apparently simple model results in a nonlinear system of equations of dimension 267 equations out of which 121 equations are trivial, i.e. (equality equations $x = y$ or $x = -y$). Such trivial equations are subject to removal by a Modelica compiler and only an optimized set of equations is evaluated (Maffezzoni, Girelli, and Lluka 1996).

The simulation of this model produces the following samples of results:

Line	Voltage (kV)	Current (A)
Line 1	0.269	314.159
Line 2	0.319	79.633
Line 3	0.069	34.633
Line 4	0.015	15.367
Line 5	0.226	75.367

6.2 Power generation and consumption

Examples for dynamic simulation can be consulted in the `PowerSystems` library, e.g. the s.c. `PowerWorld` example (Franke and Wiesmann 2014) shown in Figure 6.3. Simulation results of the power generation of the power plants and the power consumption of a city is shown in Figures 6.4 and 6.5.

So far the book is concerned with illustrating fundamental concepts of the Modelica language. This can help interested readers to further explore the detailed implementation of this and other examples found in the library source code under https://github.com/modelica-3rdparty/PowerSystems.

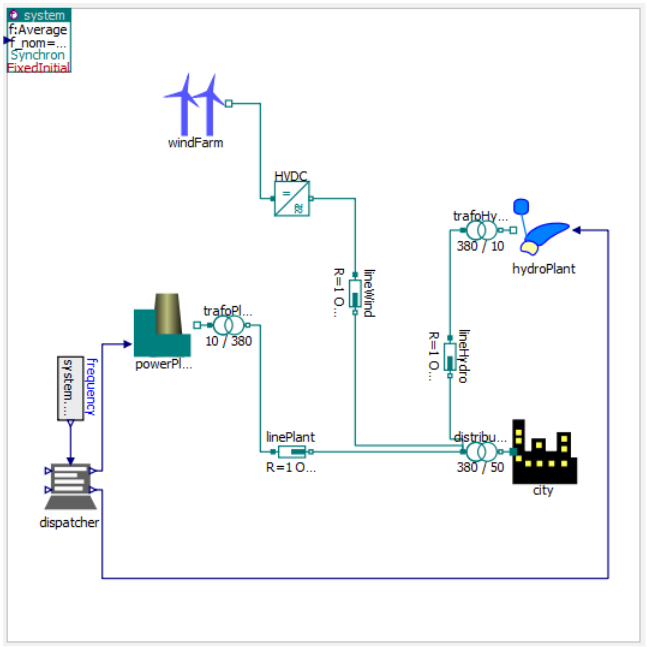


Figure 6.3: A power system network from *PowerSystems.Examples.PowerWorld* composed of a power plant, wind farm, a hydro power plant and consumption data of a city

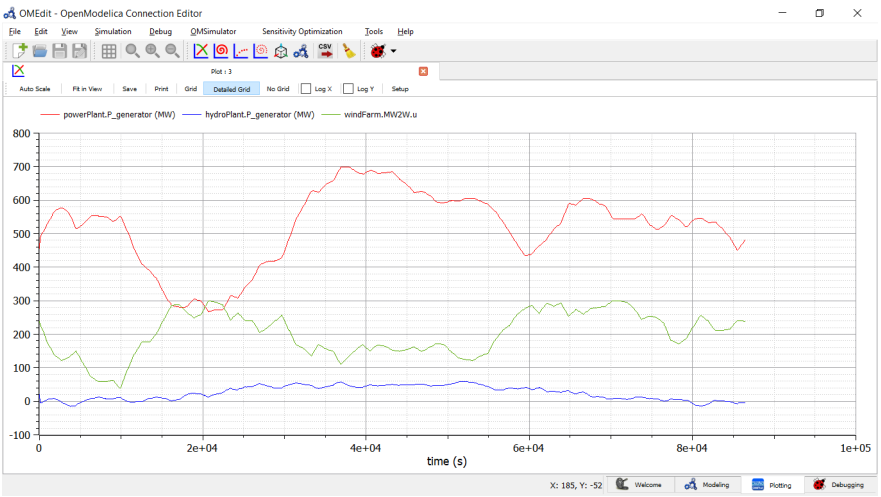


Figure 6.4: Power generation of the plants for meeting the consumption demand of a given city

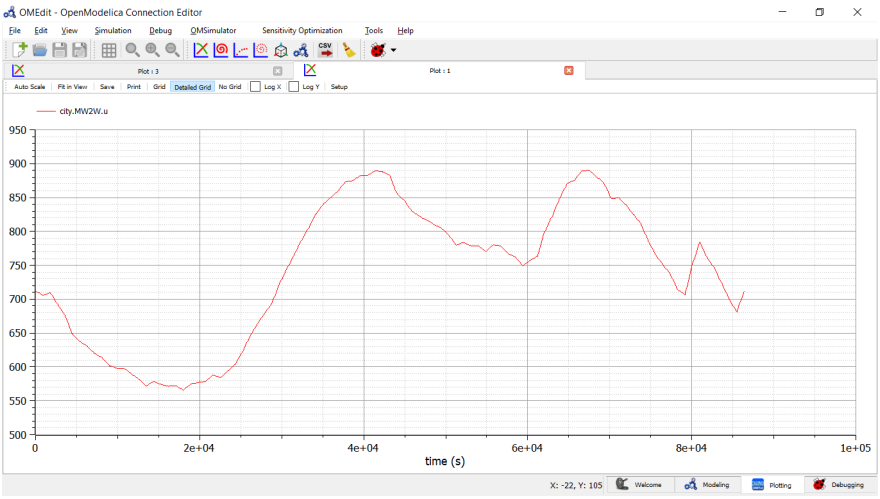


Figure 6.5: Power consumption of a given city

III Actual Aspects

7. Current state of Modelica

7.1 Language specification

While the first version of the **language specification document** released September 1997 focused on continuous-time systems, the specification was and is still continuously subject to improvement and enhancement, cf.

https://modelica.org/documents

The evolve of specification versions is motivated by

1. attempting a unique interpretation of the language among various tool vendors and
2. providing further improved and new features and capabilities

A sample of established **new features** in the past was for describing and supporting:

- discrete systems (versions 1.1, 1.2 and 1.4)
- external interfaces to C/Fortran routines (V1.2)
- array semantics (V1.3, V2.0)
- enumeration types (V2.0)
- hybrid paradigms and state machines (Ferreira and Oliveira 1999; M. Otter and Årzén 2005) (V2.2)
- stream connectors exhibiting bi-directional flow (Franke et al. 2009b) (V3.1)
- operator overloading (Olsson et al. 2009) (V3.1)
- synchronous controllers (Elmqvist, Otter, and Mattsson 2012) (V3.3)

The maintenance and progress of the language design among other issues are initiated

through well-defined procedures and are discussed in periodic **design meetings** organized by **the Modelica Association (MA)** typically 3-4 times a year, cf.

https://modelica.org/events/design_meetings

Decisions are officially electable by MA members in a transparent way. Organizational and individual membership of MA is practically open for any individual or organization under the conditions announced in

https://modelica.org/association

7.2 The Modelica standard library

An open-source **Modelica Standard Library (MSL)** was developed and continuously maintained and enhanced by MA. The library comprises a large set of various model components in many physical domains, cf. Table 7.1.

Table 7.1: Overview of some (sub)packages within MSL. An identifier `L` indicates that `Modelica.L` is a sublibrary of MSL at the top of hierarchy. The symbol `*.X` indicates that `X` is a subpackage of the previously mentioned sublibray. The symbol `*.*.Y` indicates that `Y` is a subpackage of the previously mentioned package.

(Sub)Package	Description
`Blocks`	Basic components for building block diagrams
`*.Interface`	Connectors and abstract models for I/O blocks
`*.Math`	Mathematical functions as I/O blocks
`*.Continuous`	Basic continuous-blocks described by DAEs, e.g. Integrator, Derivative, PI & PID controllers, etc.
`*.Noise` (Klökner, Linden, and Zimmer 2014)	Random noise generators / statistical distribution
`*.Sources`	Signal generation blocks
`*.Tables`	Interpolation of one- & two-dimensional tables
Subpackages	`Logical`, `IntegerMath`, `MathBoolean`, `Nonlinear`, ...
`Electrical`	Diverse electrical components
`*.Analog` (Clauß et al. 2000; Majetta et al. 2009a)	Basic electrical components s.a. resistors, capacitors, inductors, transistors and diodes, heat storage and dissipation, ... (Cellier, Clauß, and Urquia 2007)
`*.Machines` (Kral and Haumer 2005)	Electric machine including loss effects from various sources (Haumer et al. 2009) with consistent thermodynamically established concept
`*.QuasiStationary` (Haumer et al. 2008)	Quasi-stationary analysis with sinusoidal excitation
`*.Spice3` (Majetta et al. 2009b; Majetta et al. 2011)	General devices (resistors, capacitors, inductors, sources) and semiconductor devices

..Converters	Subpackages for converting ACDC, DCDC, ...
Subpackages	Digital, Multiphase, PowerConverters, ...
Fluid (Casella et al. 2006; Franke et al. 2009a)	Zero- and one-dimensional thermo-fluid flow components within subpackages: Vessels, Pipes, Machines, Valves, Dissipation, etc.
Magnetic	Magnetic components in couple of subpackages s.a.
*.FluxTubes	Electromagnetic devices with lumped magnetic networks
*.FundamentalWave (Kral and Haumer 2011b)	Rotating electrical three-phase machines relying on magnetic potential and flux fundamental waves
Math	Mathematical functions operating on matrices
Mechanics	Diverse one- and three-dimensional mechanical components within the subpackages: Multibody, Rotational Translation
Media (Elmqvist, Tummescheit, and Otter 2003)	Definitions for ideal gases, mixtures, water & air models incompressible & compressible media etc.
Thermal	Thermodynamical components within many packages s.a.
*.HeatTransfer	Heat transfer
*.FluidHeatFlow	Thermo-fluid pipe flow
SIunits	Type and unit definitions based on SI units
StateGraph (M. Otter and Årzén 2005)	A framework for modeling discrete event systems in a structured way via finite state machines based on Grafcet principles (David and Alla 1992) with capabilities for representing state charts (Årzén 1996). Earlier attempts were conducted (Elmqvist, Mattsson, and Otter 2001; Ferreira and Oliveira 1999; Mosterman, Otter, and Elmqvist 1998).

MSL can be directly employed or extended by modelers for developing their own libraries

and applications. MSL, being developed by experienced modelers, can act as instructive guidelines for tackling modeling tasks using the state of the art Modelica-based design techniques.

MSL is subject to continuous maintenance and progress according to state-of-the-art software engineering techniques and intensive discussions. Regression testing is altered for every new contribution or a justified modification. The progress of the library state can be monitored through the github website

https://github.com/modelica/ModelicaStandardLibrary

Justified contributions in the form of pull requests are encouraged.

7.3 The functional mockup interface

Due to the increasing demand of model-based technologies in various application domains there is no need to follow an ad-hoc approach for involving Modelica-based components in external model-based applications, e.g. model predictive control, embedded systems and co-simulation among others.

In contrary, utilization of the so-called **the Functional Mockup Interface (FMI)** technology (Blochwitz et al. 2011) is the standard approach. This is achieved by exporting a Modelica models or a model specified in an arbitrary dynamical modeling tools to a simulation program written in the C programming language. This C-source code follows a unified API according to the FMI standard, cf.

https://fmi-standard.org/tools/

for an actual list of tools supporting FMI.

The FMI-compliant exported model together with a descriptive XML file is referred to as **Functional Mockup Unit (FMU)**, cf. Figure 7.1. The FMU can be independently simulated or integrated into other simulation platforms.

Advanced numerical solvers requiring **the Jacboian** of an ODE/DAE model for step-size adaptive numerical integration have been served since the second version of FMI specification (Blochwitz and al. 2012). This version recommends exported FMUs to provide function calls for evaluating **directional derivatives**. This even allows the evaluation of **dynamic parameter sensitivities** for model-based predictive control (Franke et al. 2015) among other possible applications.

The functionalities demonstrated in Figure 7.1 refers to **FMI for model exchange**. Another flavor is to allow the exportation of a bundled numerical solver within the FMU. This is referred to as **FMI for Co-simulation**.

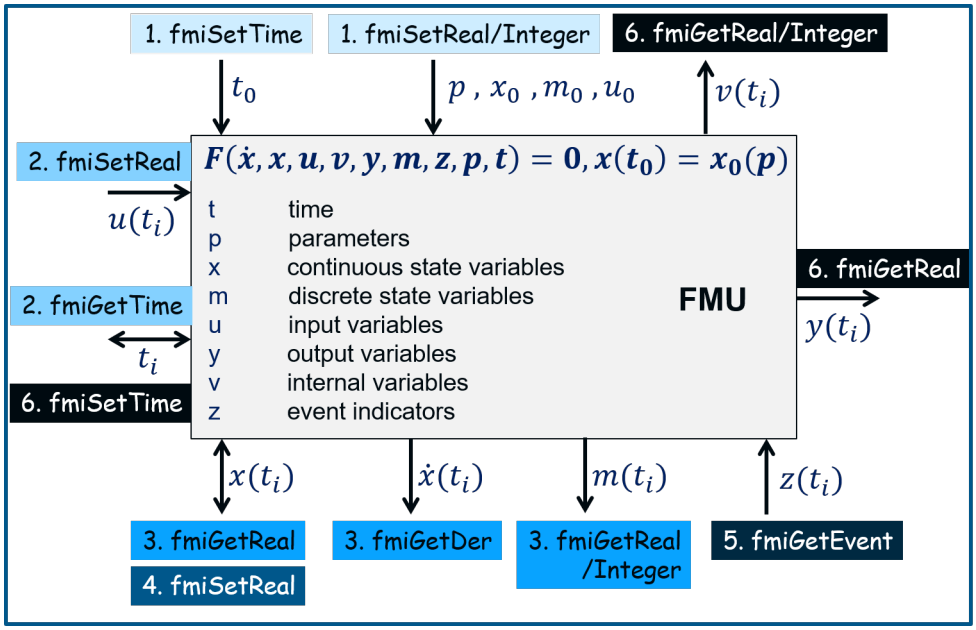


Figure 7.1: Functional mockup unit: Mathematical representation and some C-functions operating on them

Using FMI standards, generic FMU-based application-independent solutions and tools can be implemented, e.g. (Pfeiffer et al. 2012; Vanfretti, Bogodorova, and Baudette 2014). Particularly, the utilization of external design and optimization tool becomes a software engineering task e.g. (Vanfretti et al. 2016b).

Still there is a need for a further level of standardization for interacting FMUs, e.g. in power system modeling applications (Gómez et al. 2020). When describing a complex system comprising multi-physical phenomena, it is ideal to have the whole system described in only one language.

This is however almost impossible due to many reasons. First many other well-established long-rooted simulation tools optimized for specific domains exist. Their involvements in system modeling is highly beneficial. The in-depth accumulation of expertise with such domain-specific tools can not be excluded.

Thus, a practical solution that enables the collaboration of different working groups is to employ **co-simulation** where each tool exports its models as FMUs.

Proposition 7.1 It is beneficial to design such an FMI-based co-simulation of a technically complex system in a standard manner allowing the orchestration among multi FMI-based simulations.

For this purpose, there are couple of FMI-related projects maintained by MA for standardizing the orchestration of FMI-based co-simulations.

System Structure and Parameterization (SSP)

SSP proposes an XML-based specification for describing the system FMU-based components and their interrelations (Köhler et al. 2016). Moreover, the specification covers also the overall parameterization in a consistent rather than isolated manner. This is particularly useful for MiL, SiL and HiL applications.

Distributed Co-simulation Protocol (DCP)

While SSP is more related to the modeling and descriptive part of a complex system, DCP is more related to the technical part of the co-simulation including interoperability aspects in a heterogeneous system (Krammer et al. 2019).

DCP follows a master-slave architecture along a finite discrete state machine characterizing the operation of the system. A standardized I/O data exchange model and description for communication protocol among slaves are provided.

7.4 Projects

The increasing significance of Modelica-based technologies can be best reflected by the amount of involvement in research projects (and participation in organized conferences). Table II lists some large-scale research industry-oriented projects primarily for developing and improving the Modelica language and its related technologies, cf.

https://modelica.org/external-projects

for an actualized list.

Table 7.2: Summary of some large-scale research projects around Modelica-based technologies.

Project	Budget M €	Objective
(EUROSYSLIB 2007-2010)	16	Modelica libraries for embedded systems
(MODELISAR 2008-2011)	27	Developing FMI standards www.fmi-standard.org
(PEGASE 2008-2012)	13.59	Defining the most appropriate state estimation, optimization and simulation for power grid (Chieh, Panciatici, and Picard 2011; Villella et al. 2012)
(OPENPROD 2009-2012)	11	Integration of open model-driven M&S environments with industrial applications
(DYNCAP: 2011-2014)		Dynamic operation of stream power plants with CO2 capture technologies

(MODRIO 2012-2016)	21	Extending M&S tools from system design to system operation
(ITESLA 2012-2016)	19.43	Unified Modelica-based tools for pan European security assessments of large-scale power systems
(IEA EBC Annex 60 2012-2017)		New generation computational tools for building and community energy systems
(TransiEnt 2013-2017)		Efficient integration of renewable energies into urban energy systems
(ACOSAR 2015-2018)		Advanced co-simulation interface for offline and online system integration
(OPENCPS 2015-2018)		Verification/Validation and interoperability of key standard technologies (Modelica, SysML, FMI,...)
(DYNSTART 2015-2019)	0.234	Startup and shutdown processes of steam power plants with fluctuating share of renewable energy
(INDIGO 2016-2020)	2.79	More-efficient next generation of district cooling systems (Costa et al. 2020)
MISSION (2016-2024)		Modeling and simulation tools for more-electrical aircraft design (Valdivia-Guerrero et al. 2016)
(EMPHYSIS 2017-2020)	14	New FMI standards for embedded systems, e.g. electronic control units and micro controllers
(ResiliEntEE 2017-2021)		The resilience of coupled energy systems in the presence high share of renewable energy
(IBPSA Project 1 2018-2022)		Modelica framework for building and community energy system design and operation (Wetter et al. 2019)

7.5 Modelica simulation environments

Many mature Modelica simulation environments, commercial and open-source exist, a list of actual Modelica simulation tools is available under

`https://modelica.org/tools`

These environments are typically enhanced with additional services for model design and optimization, cf. (Fritzson et al. 2020, Section 4) for a compact coverage of tools associated

with OpenModelica.

In addition to FMI solution technology, cf. Section 7.3, simulation environments are usually associated with own and/or external interactive scripting capabilities. Interfaces to existing scientific computing languages s.a. Julia, Mathematica, Matlab or/and Python among other possibilities exist. With such interfaces Modelica-based simulation can be externally adjusted, performed, processed and analyzed, e.g. (Fritzson et al. 2020, Section 4.9).

7.6 Conferences and user groups

Despite the fact of Modelica could be viewed as an European product, the internationalization of Modelica through increasing activities in Asia and North America are remarkably present in the form of active user groups and periodic user meetings, cf.

https://modelica.org/users-groups

Meanwhile, there are periodically organized international conferences in three continents each every two years, cf.

https://modelica.org/events

Few facts include:

1. The 13th European version of the conference attracted couple of hundreds of participants largely from the industry with nearly one hundred peer-reviewed accepted papers
2. The first Japanese international conference was held in Tokyo May 2016
3. The first North American Modelica conference was held in Cambridge October 2018

Despite the actual state of COVID-19 pandemics, latest Modelica conferences were successfully held in a virtual manner. Participation was for free with no fees.

7.7 Modelica association membership

MA memberships are practically available for any interested party whether tool and/or modeling library providers or an end user after accomplishing transparent conditions published in the bylaws, cf.

https://github.com/modelica/MA-Bylaws/releases

The membership enables to take a part in the maintenance and progress of the Modelica language and its projects in a coordinated and organized manners according to the published bylaws. There are two forms of memberships:

1. **organizational memberships** for universities, research institutes and companies, cf.
   https://modelica.org/association
   for an actual list of organizations

2. **individual membership** for persons

The MA organization board taking care of the association common activities is periodically elected. Their members and duties are listed in the previous website.

7.8 Modelica newsletters

Newsletters on the Modelica language, tools, libraries, etc. are periodically published usually three times a year. The contents are collaboratively written by many parties through pull requests on github, cf.

https://github.com/modelica/newsletter

Older newsletters and email **subscription** can be obtained at

https://newsletter.modelica.org/

7.9 Books

Couple of **books on Modelica** in several languages exist, cf.

https://modelica.org/publications

Some of the materials in this e-book whether intentionally or not were influenced by other books. We are listing them here for interested readers who would like to explore some of the addressed topics in great details.

Definitely, the most comprehensive book providing a generous coverage of Modelica in more than a thousand of pages is provided by (Fritzson 2015). Many topics surrounding modeling and simulation is covered in this book s.a.

1. physical modeling principles and their underlying fundamental laws of Physics
2. equation-based compilation techniques
3. numerical integration methods for ODEs and DAEs
4. and many other active research topics

Some books that are not listed under MA website are still extremely useful. Particularly, (Cellier 1991; Cellier and Kofman 2006) were very useful for parts of this e-book regarding bond graphs, compilation and simulation techniques[1].

A remarkable book is **Modelica by Examples** written by Michael Tiller:

https://mbe.modelica.university/

[1] We are not doing any kind of advertisement. These are simply the books the first author have read during his PhD. There are definitely other useful books. We may provide more details in future as we manage to read more books.

The writing of the book was crowdfunded by the Modelica community. While the book is freely accessible online, printable/electronic versions can be purchased on the basis of "pay what you can or pay what you think this book deserves".

7.10 Education efforts

There are dozens of **shared lecture slides** prepared for many recognized universities, cf.

https://modelica.org/education

The OpenModelica simulation environment is equipped with interesting capabilities for preparing **interactive Modelica-based course materials**. These materials can be designed and embedded in interactive notebooks that include self-editable modeling exercises, executable simulations and plotting capabilities. Some of these simulations can be first executed after editing text-based Modelica models, cf. **OMNotebook** (Fernström et al. 2006).

Remarkably, all examples in (Fritzson 2015) are provided in an an OMNotebook-based called **DrModelica** (Sandelin et al. 2003) associated with the OpenModelica environment.

These technologies were adopted and generalized for large-scale interactive training and self-learning of Modelica in India via a so called **spoken tutorial** approach (Moudgalya et al. 2017). A spoken tutorial is a small modular piece of video that lasts for 7 to 15 minutes. It can be replayed and paused. The student needs to:

- reproduce what he just learned via OpenModelica and/or
- interactively edit text-based models and execute them

A series of well-prepared tutorials provides a decent step-by-step coverage of a particular topic. According to (Fritzson et al. 2020), more than 100000 students have been trained by the spoken tutorials on OpenModelica (and libraries). Additionally, there is also another spoken tutorial about the `OpenIPSL` power system library (Vanfretti et al. 2016b). These tutorials are accessible under

https://om.fossee.in/spoken-tutorial

Another helpful self-learning instrument is a cloud web-based generalization of OMNotebook called **OMWebbook**:

http://omwebbook.openmodelica.org

Proposition 7.2 OMNotebook, DrModelica, OMWebbook and the spoken tutorial technologies provide excellent potentials and opportunities to design, prepare and spread further Modelica-based self-learning tutorials on diverse topics

The above initiative is associated with two further interesting projects for self-learners seeking more practice: Power system Simulation Project and Textbook Companion Project

(Moudgalya 2020).

In **Power System Simulation Project**, accessible under:

https://om.fossee.in/powersystems/pssp

, students and volunteers are asked to submit a power network model example implemented in OpenIPSL. The network example may come from any reliable source s.a. textbook, academic publications or a project report. A list of so far completed model examples is accessible under

https://om.fossee.in/powersystems/pssp/completed-pssp

In **textbook companion projects**, (a community of) students and volunteers are asked to crowdsource all relevant examples and exercises within a suggested / proposed existing textbook with Modelica models based on the OpenIPSL library. By the time of writing this Section (July 2021), modeling examples of (Stevenson 2014) and (Kothari and Nagrath 2014) have been accomplished, while those of (Gupta 2005) and (Nasar and Trutt 1998) are still under progress. Further complete set of examples for relevant books in related domains s.a. control, power electronics, differential equations, among 68 books exist. The set of completed books is accessible under

https://om.fossee.in/textbook-companion/completed-books

8. Relevant Modelica Libraries

A large ever-growing set of third-party open-source libraries exist and they get monitored whenever an update takes place (Tiller and Winkler 2014), cf.

www.modelica.org/libraries

or

https://github.com/modelica-3rdparty

Most of these libraries are distributed under **Modelica BSD-like licenses**, cf.

https://www.modelica.org/licenses

for more details. To the best of our knowledge, we provide a list of existing open-source libraries that are or can be useful for power system modeling applications. In this section, a quick overview of existing open-source libraries are summarized.

R Consider reporting us a library that you think it should be listed here

8.1 Open-source power systems libraries

Power system modeling is an active application area within the Modelica community since an early stage of the Modelica language (Bachmann and Wiesmann 2000). The MSL provides some basic capabilities based on first-principle models (e.g. resistors, capacitors, inductors, etc.) for constructing of power system networks, cf. Table 7.1.

Table 8.1 provides an overview of existing open-source Modelica libraries for various power system modeling applications including classical power network modeling.

Table 8.1: Open-source Modelica libraries in power system modeling applications

`Clara` (Brunnemann et al. 2012; Gottelt, Hoppe, and Nielsen 2017) `www.claralib.com/`	**Thermo-hydraulic behavior of stream power plants** with **coal dust firing** with focus on technologies for **carbon capture and its storage**. Processes governing fuel handling, cooling of combustion chamber and electrification of the stream in turbo generators among others are treated. Components include heat exchangers, pumps, valves, reservoirs, turbines, compressors, absorbers, steam accumulator (Richter, Oeljeklaus, and Görner 2019), gases and solvents. Transient analysis of power outputs can be performed. Capabilities for controller testings and investigation of start-up and shut-down processes are provided (Marx-Schubach and Schmitz 2019).
`ComplexLib` (Enge et al. 2006)	**Steady-state analysis of electro-mechanical drives** with electrical AC subsystems employing the phasor method
`ModPowerSystems` (Minz, Netze, and Monti 2016)	A modeling framework facilitating the increasing employment of **power electronics** in power systems. Capabilities for static and dynamic phasors simulation as well as **electromagnetic transient** are present.
`ObjectStab` (Larsson 2000)	Focus is devoted for modeling **transient and voltage stability analysis**. Components include generators as slack or PV nodes, 3rd or 6th order dq-models of excitation and governor control systems, transmission lines, reactive power compensation devices; shunt reactors, shunt capacitance and series capacitance, transformers, static and dynamic loads, buses and faulted lines among others. Although maintenance and progress is inactive, the library is suitable for educational purposes as it does not exploit later established advanced language features.

`OpenIPSL` (Baudette et al. 2018; Vanfretti et al. 2016a)	Validated set of power system components based on **phasor-time approach** validated against reference models from common power system tools. Power system models, including common known bus-systems models, **electro-mechanical transients** comprising power generation, transmission and consumption together with equipment components for control and stabilization can be constructed. Initial guesses corresponding to power-flow simulations are provided through other simulation tools. The library is not only suited for realistic applications but also for educational purposes (Murad et al. 2017).
`PhotoVoltaics` (Brkic et al. 2019)	**Photovoltaic components** including cells, modules and plants, enhanced with **power converters based on MSL** components among other features. Validation was conducted using selected industrial module data sheets. (Brkic et al. 2019) includes a good summary on the features of other PV libraries and their limitations.
`PowerGrids` (Bartolini, Casella, and Guironnet 2019)	Modeling of **electro-mechanical phenomena** in power systems including power generation and transmission targeting **European transmission system and distribution system operators**. Power flow analysis is expressed as initial equations which solution is the initial values of the power system network. By providing a good-initial guess, transient stability analysis can be conducted. The library was used for feasibility studies of realistic large-scale networks with thousands of nodes, generators and lines represented by over half-million equations (Casella, Leva, and Bartolini 2017).
`PowerSystems` (Franke and Wiesmann 2014)	**Arbitrary phase systems** in one single framework covering a large set of components for DC and three-phase AC. Steady-state, transient and asymmetric analysis can be conducted. An extensive overview is presented in Chapters 4, 5 and 6.

SPOT	One of the earliest libraries for power systems with extensive set of model components suited for detailed modeling of **transient effects** (Bachmann and Wiesmann 2000). Its implementation has influenced many subsequent libraries including the PowerSystems library. SPOT provided **a uniform treatment of natural (abc) and modal (dq0) coordinates** in one framework. Per-unit system for parameterization is supported (Franke and Wiesmann 2014). Although current development is inactive, its contents is of educational value as it is using less advanced language features that did not exist in earlier versions of the Modelica language. Actually, before implementing a new library, it is encouraged to explore and learn from the implementation of SPOT.
SolarTherm (Calle et al. 2018)	Typical components, with varying degree of complexity, for building a **solar thermal power plant**. Components include receivers, plates, pumps, heat-transfer, fluids, energy storage, power generation, whether data, control. The library is associated with a range of Python-based tools and scripts that automate the process of testing, simulating, optimizing and visualizing the results.
ThermoPower (Casella and Leva 2006; Schiavo and Casella 2007)	Dynamic modelling of **thermal power plants** with components for water and gas properties. Heat exchangers and electric generators are included. The library has been particularly useful for the study of control systems in traditional and innovative power plants and energy conversion systems including **steam generators, coal-fired power plants, combined-cycle power plants, nuclear power plants, solar plants, organic Rankine cycle plants, and cryogenic circuits for nuclear fusion applications**.
TransiEnt (Andresen et al. 2015; Andresen et al. 2019; Heckel and Becker 2019)	Coupled energy supply grids: **electricity, district heating and gas grids** (Andresen, Bode, and Schmitz 2018) with attention in **energy storage** in the presence of **fluctuating renewable energies** (Bode and Schmitz 2018; Dubucq and Ackermann 2017). Components include conventional and renewable power plants, electrical and thermal loads, buildings, districts, cities, electric- heat- and gas distribution elements and storage. **Conversion technologies s.a. power-to-gas (involving electrolyzer (Webster and Bode 2019)), power-to-heat, gas-to-heat and gas-to-power** are tools for enabling the transition to renewable energy source powered society (Bode and Schmitz 2018).

`WindPowerPlant` (Eberhart et al. 2015)	Simulation of **wind power plants** with components for wind turbines, generators and control

The underlying assumptions would usually presume that (Casella, Leva, and Bartolini 2017):

1. three-phase voltages and currents are always balanced, and can thus be described through the phasor approach (Araujo and Tonidandel 2013; Steinmetz 1893)
2. network frequency remains within an acceptable range around its reference value and thus a constant impedance is considered
3. a fixed network topology is assumed throughout network simulation

Many of these libraries commonly realize most of the following aspects:

- Capabilities of browsing, reusing, modifying or extending existing model components
- Flexible way of modeling, e.g. acausal modeling principles make networks construction very cumbersome, by simply connecting loads, busses, generators and lines together rather than relying on traditional admittance matrix methods
- Uniform framework gathering detailed mechanical power generation (e.g. wind turbines), thermal aspects (e.g. PVs and thermal flow), weather forecast data, classical grid components, control components and strategies, faults, switches and events all together
- Flexible mixing of different components from different libraries

Special attention has been neutrally devoted to the `PowerSystem` library. This library realizes the following advantages:

- Several analysis aspects only in one package including detailed dynamic transient analysis, quasi-steady state and steady state (i.e. transient and voltage stability) analysis and power flow calculations‘
- Uniform treatment of phase-systems (DC, AC one-phase, AC three phases) with a flexible switch between different phase systems
- Uniform treatment of different coordinate representation (e.g. natural abc representation, model dq- and dq0 representations), i.e. the same models can be easily simulated with different representation of phase systems

More detailed comparisons among some of the listed Modelica libraries is provided by (Winkler 2017).

8.2 Energy in buildings and/or districts open-source libraries

Modern energy systems modeling applications may view buildings as a part of the electrical grid. They are not only energy consumers, but can be also energy producers within the

electrical grid. While building modeling applications are usually focused on the internal thermal behavior, still small-order building models can be employed in the context of large-scale energy simulation at district-level (Frayssinet et al. 2017) including communication and transportation (Bollinger et al. 2018; Lu et al. 2019).

The interactions between thermal, gas and electrical networks and the opportunity of storing excess of electrical power in the form of thermal energy among other possibilities make libraries from the buildings modeling community of a vital interest for modern power systems modeling applications.

Table 8.2 provides an overview of existing open-source libraries in the buildings/districts domain.

Table 8.2: Open-source Modelica libraries in (thermal-)energy modeling of buildings and districts

`AixLib` (Müller et al. 2016)	**High and reduced order building models** with focus on **HVAC systems**. It has been employed in the field of district heating and cooling networks (Mans et al. 2019) as well as in hardware-in-the-loop (Schneider et al. 2015) among others.
`Buildings` (Wetter et al. 2014) simulationresearch.lbl.gov	A comprehensive library with extensive set of components for modeling aspects of **thermal energy behavior** in buildings including HVAC, multizone heat transfer and airflow among others. The library is enhanced with a package **interfacing and interacting with the electrical grid** in two directions (Bonvini, Wetter, and Nouidui 2014). Package includes weather modeling for renewable production, PVs, wind turbines among others.
`BuildingSystems` (Nytsch-Geusen and al. 2016)	Main focus on **thermal energy** (heating and cooling) in buildings and districts. Building models range from **simplified 0-dimensional low-order model to 3-dimensional multi-zones models**. Components for HVAC, solar thermal systems and photovoltaic systems are provided.
`DisHeatLib` (Basciotti and Pol 2011)	Modeling **thermal energy** aspects (Heating, cooling, pipes, storage, control, etc.) in **districts** with optional interfaces to the electrical power grid (Leitner et al. 2019).

DCOL (del Hoyo Arce et al. 2018) https://zenodo.org/record/818289	**District heating and cooling systems**, including distribution networks and thermal energy storage. The library was used in establishing a virtual test-bed of a district cooling production plant with compressor and absorption chillers (Zabala et al. 2020). Components include water storage, pipes, multi-zone air-flow, actuators, dampers, valves, radiators, heat pumps, etc.
FastBuildings (Baetens et al. 2015)	**Low-order building models** employing common components of electrical circuits ideal for smart grid modeling applications
GreenHouse (Altes-Buch, Quoilin, and Lemort 2019)	Modeling **greenhouse climate** including indoor climate, ventilation, climate control, generation and storage units (Combined Heat and Power (Altes-Buch, Quoilin, and Lemort 2018) and heat pumps), thermal energy storage and corp yield.
IDEAS (Jorissen et al. 2018)	Main focus is on **district thermal energy simulations** with interfaces to electrical systems, for instance the impact of PV and heat pumps on low-voltage electrical grids can be investigated (Protopapadaki and Saelens 2017; Protopapadaki and Saelens 2019).
modelica-ibpsa (Wetter and al. 2015)	**Basic interfaces and components common for modeling applications of energy of buildings**. It provides guidelines and **standardization** for advanced Modelica libraries in the Energy in Buildings domain. By relying on this standard, it easier to exchange components among various Modelica libraries targeting different scope of applications. There are many libraries listed in this Table that are based on modelica-ibpsa.

8.3 Useful open-source libraries

Table 8.3 provides a list of some 3rd-parties open-source Modelica libraries providing fundamental methodologies and/or tools for useful purposes. These libraries are not directly related to modeling of power systems but still they can be useful in some aspects. Many of these libraries are actively maintained, and some of them are discontinued. Thus, for those who are involved in similar activities, they can

- build on these efforts rather than starting from scratch
- directly contribute to these libraries and/or make these efforts active again

Table 8.3: Open-source Modelica libraries potentially useful for some Power Systems Modeling applications

ABMLib (Sanz, Bergero, and Urquia 2018)	**Agent-based modeling framework**. Source can be obtained under www.euclides.dia.uned.es.
ADMSL (Elsheikh 2014)	A demonstrative example for computing **dynamic parameter sensitivities** of Modelica models using **equation-based algorithmic differentiation techniques** (Elsheikh 2015)
AdvancedNoise (Klöckner, Knoblach, and Heckmann 2017)	Additional features to the existing Noise library within MSL incorporating more **stochastic distributions** and tools
AlgebraTestSuite (Sielemann, Casella, and Otter 2013)	Models for **testing initialization problems** of typical large-scale differential algebraic equations using a proposed initialization heuristic (Sielemann and Schmitz 2011; Sielemann 2012)
CellularAutomataLib (Sanz, Urquia, and Leva 2014; Sanz, Urquia, and Leva 2016)	**Describing dynamic phenomena dependent on spatial coordinates**. One- or two-dimensional cellular automates are present.
DESLIB (Sanz et al. 2012) www.euclides.dia.uned.es	Simulation of **discrete-event systems** following DEVS formalism. This has been used to reproduce some parts of the SIMAN language (Sanz et al. 2013). A library corresponding to the arena simulation language (Prat, Urquia, and Dormido 2006) suited for industrial dynamics (Forrester 1961), e.g. describing processes of factory automation, is included.
ExternalMedia (Casella and Richter 2008)	Enabling inclusion of **external fluid property code** in Modelica compatible to interfaces provided by MSL. Focus is devoted on **two-phase single-substance fluids**.

`FailureMode` (Pop et al. 2012)	A collection of failure modes assisting modelers to systematically recognize the **root-cause of common runtime** errors through useful debugging information by a Modelica simulation tool. Typical errors include incorrect parameterization, invalid initialization, boundaries violation among others. This is particularly useful in the development stage.
`FaultTriggering` (Linden 2014)	**Proposed standardization for triggering common faults during simulation runtime**. This is realized by fault-output blocks attached to model components to trigger user-chosen faults.
`FMITest`	**Testing connections of FMUs** particularly during initialization and iteration.
`LinearMPC` (Hölemann and Abel 2009)	**Model-based predictive control** for linear processes
`Modelica_DeviceDriver` (Bellmann 2009; Thiele et al. 2017)	Allows **access to hardware devices** such as input devices (keyboard, 3D-connecion, SpaceMouse, etc.) within Modelica models. This is particularly useful for interactive simulations with animations, Human-in-the-Loop and/or Hardware-in-the-Loop simulators as well as control applications. Part of the library is included in MSL 4.0.0.
`Modelica_LinearSystems2` (Baur, Otter, and Thiele 2009; Otter 2006)	Data structures for **linear time-invariant differential and difference equation systems** as well as polynomial-based operations on complex numbers. Typical operations on these structures are provided enabling description of transfer functions for common linear control analysis.
`Modelica_Requirements` (Otter and al. 2015)	Formal modeling of **requirements and their verification** during simulation
`Optimisers`	**Specification of dynamical optimization** using Modelica components. The optimization problem is solved using external solvers.

`ModelicaDEVS` (Beltrame and Cellier 2006)	A realization of **discrete event system simulation formalism** (Ziegler 1976) in Modelica, cf. Subsection 9.2.3
`ScalableTestSuite` Casella 2015b & `ScalableTestGrids`	A collection of parameterizable large-scale Modelica models appropriate for **benchmarking**. The later library is recently developed and is more focused on power system models. The library is to be reported in the Modelica conference 2021 in Linköping.
`SystemDynamics` (Cellier 2008)	Modeling various dynamical phenomena in many areas including economics, industry, environments, according to **the principles of system dynamics** of J. Forrester (Forrester 1969; Forrester 1971a; Forrester 1971b)
`ThermalSeparation` (Joos, Dietl, and Schmitz 2009)	Gas-separation, absorption, adsorption and rectification processes including heat and mass transfer within mixed-phases mediums (Marx-Schubach and Schmitz 2019).
`XogenyTest`	**Unit testing** of Modelica models and libraries using Modelica components

8.4 Commercial libraries

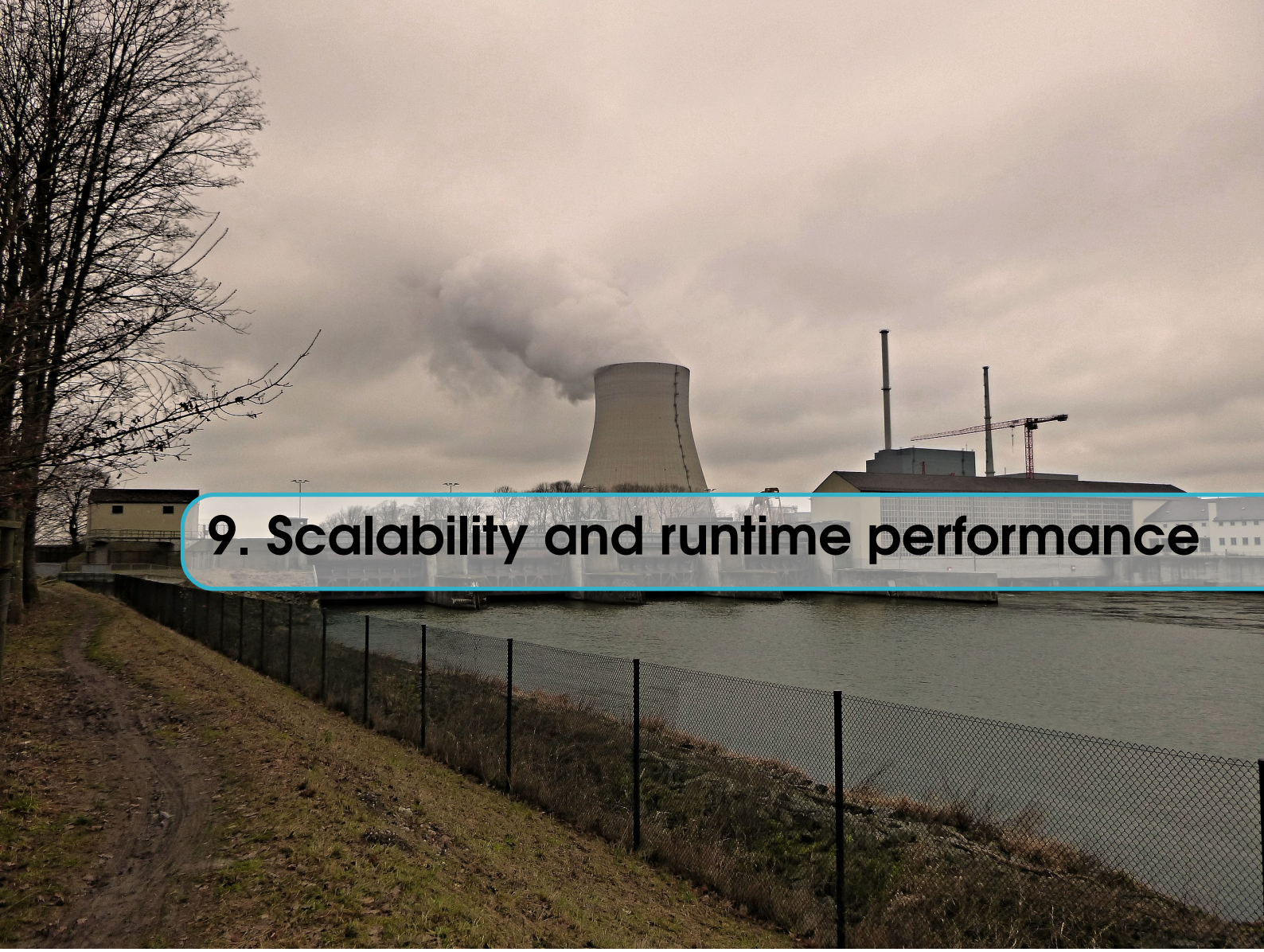
R To the owners of relevant commercial Modelica libraries, your library can be reported here. The table is ordered in alphabetical order. As the case with Open source Modelica libraries, the description of any library shall not exceed 100 words. However, as many academic references as possible can be reported. A URL to a comprehensive and actual description of the library can be also provided, where then an interested reader can explore more details.

Table 8.4 provides an incomplete list of commercial Modelica libraries that are directly or indirectly useful for power system modeling applications.

Table 8.4: Open-source Modelica libraries potentially useful for some Power Systems Modeling applications

`ElectricPower-System` (EPSL) Dassault Systèmes (CAUJOLLE et al. 2019)	EPSL enables **phasor based modeling of AC and DC, as well as combined AC/DC (hybrid) networks**. Key components are provided at different levels of detail. The so-called architectural level uses "standard" phasor theory resulting in a quasi-static description of the system. The functional level uses dynamic phasors (Yang, Bozhko, and Asher 2014) to cover dynamic effects in the models. Additionally the EPSL enables the description of harmonics of the one or multiple independent fundamental AC frequencies, which can be either constant or variable. Currently, AC components are either single or three-phased.

IV Advanced Aspects



9. Scalability and runtime performance

9.1 Limitations

Energy power systems together with their boundary conditions are naturally of **decentralized large-scale complexity**. This aspect is further emphasized by the current transformation of the energy systems, where smaller units (e.g. active buildings, wind energy, farms, solar farms etc.) enhance the complexity of the system.

Currently, multi-domain universal simulation environments may suffer from **scalability issues** when enlarging the modeling size and/or considering low-level detailed micro aspects, e.g. the charging of an electrical car in a smart home within a smart grid.

In (Elsheikh, Widl, and Palensky 2012b; Palensky, Widl, and Elsheikh 2012), a demonstrative smart grid model has been implemented comprising various common concepts:

- **Physical world / continuous models**: power generation, consumption, heating/-cooling, etc.
- **Stochastic aspects**: weather prediction, stochastic distribution for collective description of individuals, etc.
- **Information technology / discrete models**: controllers, communications, etc.

- **Smart agents**: human behavior, management systems, energy markets, etc.

It was found[1] that such large-scale models badly scale up with Modelica-based environments (nearly $O(N^3)$) (Palensky et al. 2013). Moreover, due to **excessive memory requirements**, maximum number of state variables in a Modelica simulation was in the order of $O(10^4)$.

In order to identify the reasons behind the low scalability of such models, it is notable that object-oriented models are of **sparse nature**. Every model typically consists of hierarchical composition of subsystems each with a limited set of components. Each of these components is connected to only few components.

When such a model translates to a system of equations, each equation consists of few variables. Thus, the Jacobian, corresponding to the partial derivatives of equations w.r.t. state variables, is largely sparse with high-amount of zeros, typically $> 99\%$ of the entries are zero (Casella 2015b).

By comparing the runtime performance of both of acausal modeling and **multi-agent based modeling paradigms** (Palensky et al. 2013), the reasons behind such **low runtime performance** of Modelica environments can be deduced as follows (Casella 2015b):

9.1.1 Translation to one single big block of equations

Modelica models get usually translated to **one single ODE/DAE block** and not as **a cascaded or separate subsystems** of much smaller ODEs/DAEs like the case with agent-based modeling paradigms. While the translated model could still result in some isolated blocks of linear and/or nonlinear equation subsystems to be separately evaluated (Duff, Erisman, and Reid 2017, Chapter 6), the main ODE/DAE block would be usually large and computationally dominant.

On the other hand, an **agent-based modeling tool** translates such a model into a cascaded or interrelated set of equation subsystems. The evaluation of such subsystems is conducted separately or in a sequential / parallel manner. Communications of intermediate solutions between subsystem occur in a user pre-defined manner, e.g. specific conditions.

Overall, an agent-based modeling approach results in a much lower runtime complexity. The resulting solution could exhibit **numerical accuracy issues**, though it is usually neglectable.

9.1.2 Single-rate numerical integration

For large-scale networked models, the resulting ODE/DAE comprises phenomena that runs at **different runtime scales or time constants**. For such models, **variable-step**

[1] at that time and not by the time of writing or reading this book

size numerical integrators are the commonly recommended choice of solvers.

Nevertheless, the subsystem with the quickest dynamics (i.e. with the smallest time constant) would cause such an integration scheme to pick a tiny integration step-size that is irrelevant for a large portion of the system with slower dynamics.

While such a common numerical integration paradigm guarantees a precise **numerical accuracy**, still accuracy of agent-based modeling paradigms is usually reliable enough due to the highly loose-coupled nature of large-scale networked models.

9.1.3 No exploitation of sparsity patterns

The resulting ODE/DAE would usually require an **implicit numerical integration scheme** that is numerically provides better stability to the potential presence of stiffness. There is no standard definition what stiffness precisely mean, but the presence of stiffness implies that the ratio of the system's largest time constant to the smallest time constant is so large leading to a high degree of variation in the solution trajectories.

The corresponding **sparse Jacobian** of a system is a part of the underlying nonlinear equation to be solved by a Gauss-Newton iteration scheme for solving an implicit equations. Apparently until few years ago, Modelica simulation environments didn't consider exploiting sparsity patterns of the Jacobian both at computation and memory level, at least in their provided default set of solvers.

For example, describing the evaporation of water to steam in a tube for a plant model comprising 800 state variables takes nearly 2 hours with a Modelica simulation environment in comparison with 80 seconds by an in-house simulation tool making use of sparse solvers (Link, Steuer, and Butterlin 2009).

Algorithmic complexity of matrix inversion for solving a nonlinear equation system at every integration step is of order $O(N^3)$. The memory complexity for storing a dense Jacobain is of order $O(N^2)$, not even considering approaches relying on the Hessian matrix.

A system with 10000 state variables would require 1.6 GB of storage (Casella 2015b). This has also significant penalty on runtime performance due to the excessive transfer of data among the various levels of data storage and caches.

9.1.4 Insignificant local events cause tremendous computation

Low-level local events (e.g. opening a window in a flat) which are insignificant for large-scale portion of a smart grid model still cause large-scale intensive computation for identifying a single state-event (another example: automatic switching of a heater in a smart grid model).

The underlying **root-finding algorithm** causes the evaluation of the entire equation system for couple of iterations for detecting each state event (Eich-Soellner and Führer 1998). A state-event can even trigger further state-events complicating the evaluation process, i.e. chattering (Mehrmann and Wunderlich 2009). Having a high number of events extremely slowdowns the runtime simulation performance.

9.2 Active research agenda for improving runtime performance

Achieving significant **simulation runtime improvement** is not impossible. In contrary, classical modern **sparse techniques** for huge-scale linear and nonlinear equation systems have been established in the scientific computing community many decades ago e.g. (Saad 2003).

Apparently, Modelica tool vendors were largely focused on the tedious task of fulfilling a full support of the always growing set of language features. Consequently, not only attention at runtime performance but also additional efforts for improving compiler capabilities and compilation performance are demanded (Pop et al. 2019).

(Casella 2015b) addressed **some research directions** for the Modelica community in order to overcome some addressed runtime performance obstacles as follows:

9.2.1 Exploiting sparsity patterns and sparse solvers

Integration of **sparse numerical solvers** can reduce the overall computational complexity of large-scale decentralized networked models from $O(N^{2.6})$ to $O(N^{1.3}) - O(N^{1.8})$ according to (Casella 2015b).

The OpenModelica simulation environment reported experimental integration of sparse solvers for linear and nonlinear systems (Braun, Casella, and Bachmann 2017). Accordingly, transient simulations of realistic power system network models with thousands of nodes, generators and lines were conducted with runtime performance comparable to domain-specific tools (Casella, Leva, and Bartolini 2017).

The Dymola simulation environment recently reported the utilization of sparse solvers with emphasize on root findings for identifying state events (Henningsson, Olsson, and Vanfretti 2019). Other Modelica simulation tools are expected sooner or later to follow similar directions, if not already done.

A comparative power system modeling **benchmarking** between Dymola and OpenModelica was recently reported (Dorado-Rojas et al. 2020). n this compariso, Dymola has shown superior performance. However, it is thought that sparse solvers were not really employed with OpenModelica due to recent modifications on its compiler front-end.

It is beneficial to keep such benchmarking attempts active among Modelica simulation tools to encourage tool vendors. An excellent Modelica library is the `ScalableTestSuite` and `ScalableTestGrids` libraries reported in Section 8.3.

9.2.2 Multi-rate numerical solvers

The main stream of numerical integrators follows a **single-rate numerical integration** scheme, where at every integration step the entire equation system is evaluated. On the other hand, employing **multi-rate numerical solvers** (Rice 1960) for systems with mixed slow and fast dynamics including stiff systems can largely improve the performance of large-scale decentralized network models.

In multi-rate solvers, numerical integration of the entire equation system takes place globally only at few time steps. However, for many sub-iterations, only those state variables that violate a desired accuracy and/or a stability threshold are numerically integrated at smaller step sizes. Practically every time step aims at the evaluation of different subset of state variables, while employing interpolation or extrapolation for the rest of variables.

A case study in (Rande and Casella 2014) shows that **the runtime computational complexity** of numerical integration via a multi-rate implicit solver is of a quadratic rather than a cubic order. Multi-rate numerical integration is an active research area, e.g. (Roberts, Sarshar, and Sandu 2018; Savcenco 2009), and has attracted the Modelica community, e.g. (Bonaventura et al. 2020).

How to determine the subsystem, statically or dynamically at runtime, is an issue to investigate. In (Casella 2015a), employing common graph-based algorithms models are suggested for identifying strongly interrelated subsets of equations. Each of such strongly connected equation subsystems is numerically integrated using smaller local step-sizes while the whole equation system is globally evaluated at larger step-sizes.

Another approach for realizing multi-rate algorithms is by exploiting **synchronous language constructs** that allow the modeler to define different time clocks with different numerical solvers to each component of a system (Thiele, Otter, and Mattsson 2014). This approach however requires explicit specification of the subsystems.

9.2.3 Solvers for massive number of state-events

Quantized States System (QSS) methods (Zeigler and Lee 1998), though have been first established in the context of distributed simulation, is one of the successful methods suited for networked models with a massive number of state events. The main idea behind QSS methods is that discritisization of state variables rather than the time axis is considered.

Quantized state variables follow piece-wise constant trajectories rather than continuous trajectories. In contrary to classical methods, not all states get updated in a synchronous

manner, but only those violating "quantum level crossings". In other words, each state variable gets updated upon its own pace in an asynchronous manner.

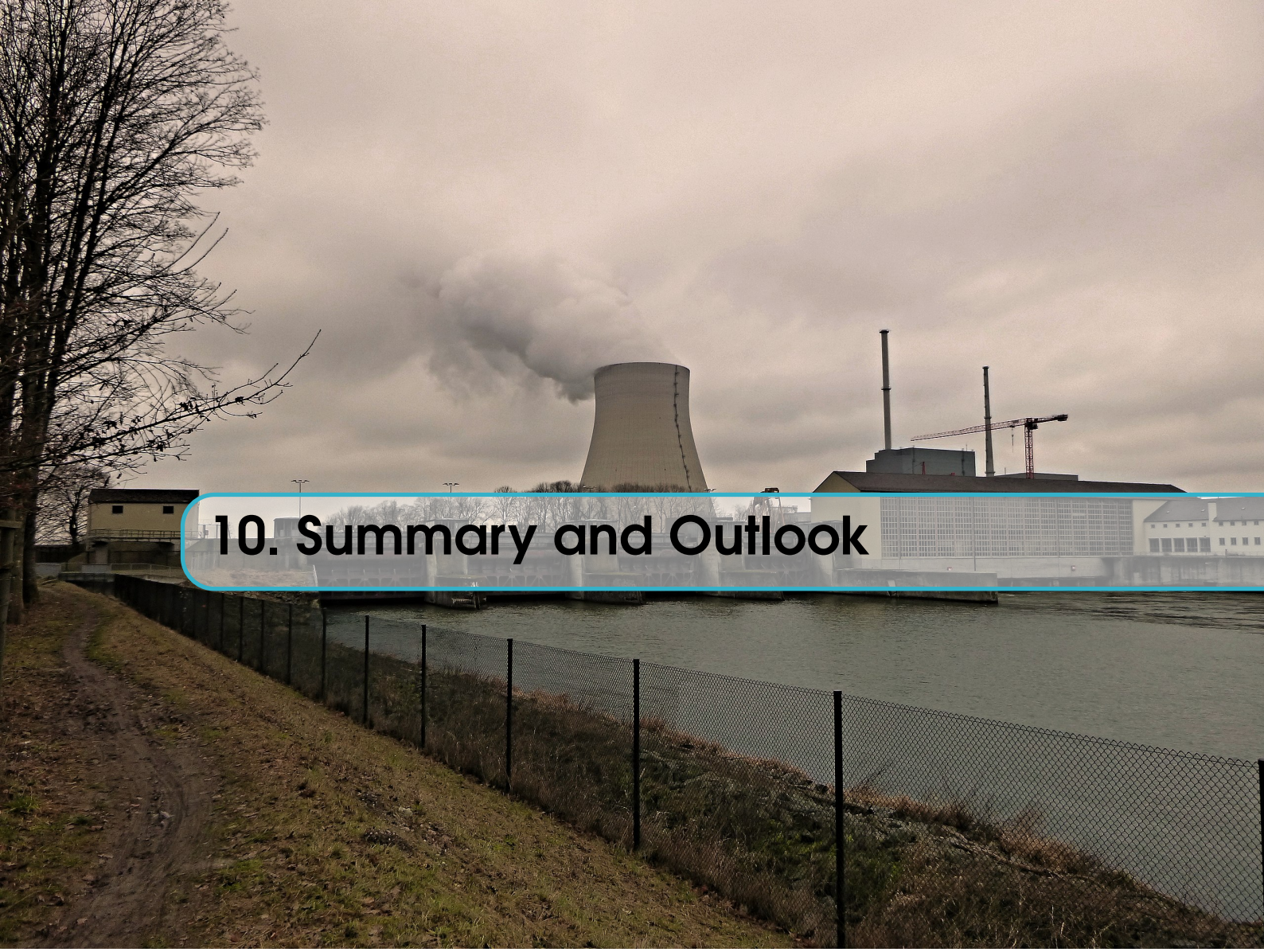
In such a dense evaluation scheme, zero crossings corresponding to state-events are not evaluated using a Gauss-Newton iteration as the case with classical approaches, but they are simply predicted leading to a significant improvement of the overall computational performance, cf. (Floros, Cellier, and Kofman 2010; Kofman 2004) for more details.

Earlier QSS methods used **explicit iteration schemes** for numerical integration (Kofman and Junco 2001). **Second and third-order QSS** methods were further established by letting states variables follow parabolic and cupic piece-wise constant trajectories, respectively (Kofman 2006; Kofman 2002). **Implicit QSS** methods suited for stiff systems have been later established (Migoni and Kofman 2009).

QSS methods have been integrated to the OpenModelica simulation environment (Bergero et al. 2012). **Two benchmarks** (Floros et al. 2014) corresponding to district cooling system (Ceriani et al. 2013) and a smart grid model (Elsheikh, Widl, and Palensky 2012a) demonstrated the power of integrating QSS to OpenModelica. In both benchmarks, the simulation runtime scale quadratically (from $\sim 5N^2$ to $\sim 6N^2$) with the well-known DASSL solver while it scales linearly (from $\sim N$ to $\sim 3N$) with QSS methods.

In another benchmark (Bergero et al. 2018) concerned with **stiff hybrid systems**, several cases studies were evaluated involving demand-side centralized power management of a population of thermostatically-controlled air conditioners in buildings (Perfumo et al. 2012) as well as a centralized cooling power distribution system in a building adopted from (Ceriani et al. 2013). It was shown that **Implicit QSS methods exhibit a scalabiltiy of a linear order**, while classical solvers demonstrate a nearly quadratic runtime performance complexity. Similar conclusions have been drawn for a renewable hybrid energy system models where the presence of DC-DC converters, diodes and transformers cause stiffness (Migoni et al. 2016).

Alternatively, techniques that attempt to solve a much smaller but significant set of equation systems for identifying state-events of large-scale systems have been established, e.g. (Bergero et al. 2018). This can be achieved by exploiting structural information of the model including the way equations are causally sorted (Höger 2013; Sanz, Urquia, and Casella 2014). Further significant improvements in the context of **stiff systems** can be achieved by considering **a mixed-mode solver** that automatically switches between an implicit QSS method and a classical implicit solver based on numerical criteria (Di Pietro et al. 2020).



10. Summary and Outlook

In this Chapter, **the advantages of Modelica features** in the context of future power systems modeling applications are summarized. Then some **future research directions** related to language features are highlighted.

10.1 Advantages of the Modelica language

10.1.1 Object-oriented paradigm

Modelica is a powerful modeling language for rapid prototyping of multi-physical systems by supporting:

- high-level descriptive language elements following common **object-oriented modeling paradigm** which allows flexible modification of complex models and their components
- flexible organization of model components within libraries of browsable packages and subpackages
- **graphical modeling facilities** allowing flexible experimentation of submodel variants
- hierarchical multi-level modeling corresponding to different resolution levels possi-

bly within hybrid paradigms enabling both bottom-up and top-down of model design

among others, cf. Chapters 4 and 5 for more details.

10.1.2 Domain-independent multi-physical modeling concepts

Future power systems incorporate various physical domains, including mechanics, thermodynamics, hydraulics, etc. The adopted physically-universal modeling concept, i.e. the s.c. **acausal modeling concept**, supports various physical domains and meanwhile provides the capabilities for describing the interrelationships among these physical domains, cf. Chapter 3.

10.1.3 Advanced methods for efficient runtime simulation

Definitely, the large-scale dimension of future decentralized power systems networked models enforce simulation tools not only to employ state-of-the-art methodologies for improving runtime performance, but also to become a major driver and contributor to the progress of these methods.

Common Modelica simulation environments (strive to) support:

- **advanced graph and numerical algorithms** for simplifying, optimizing large-scale equation systems and transforming them to solvable equation (sub)system(s)
- common standard as well as advanced solvers for **numerical integration** of hybrid ODEs/DAEs with adaptive error control and treating potential presence of stiffness and events
- capabilities for **hybrid modeling paradigms**

Nevertheless, common Modelica simulation environments are still subject to active progress in order to efficiently handle modern large-scale power system applications, cf. Section 9.2.

10.1.4 Standardized (co-)simulation interfaces

Co-simulation (Gomes et al. 2018; Schweiger et al. 2020a) has been widely used in the last decade to interface power system simulators with communication network simulators (Müller et al. 2016; Palensky et al. 2017). Co-simulation is expected to play a significant role also at power level simulation when the need of highly specialized tools for specific subsection of the system does not allow the use of a single simulation environment.

Employing Modelica models enable co-simulation via the **standardized FMI specification**, established under the Modelica Association Umbrella. Modelica models can be easily exported as importable stand-alone FMU within other tools or modeling paradigms (Elsheikh et al. 2013), cf. Section 7.3. FMUs exported from non-Modelica tools can be imported within a Modelica editor as a connectable model component, cf.

https://fmi-standard.org/tools/

Last but not least, Standardization allows for **general tool-independent methodologies** to take place, leading to significant effort reduction when realizing co-simulation.

10.1.5 Code generation capabilities

Code generation capabilities are fundamental to allow simulation execution on dedicated or **embedded hardware platforms**. This provids a flexible way of incorporating high-level model components for **real-time and hardware in the loop** as well as for **real-time control application** or **model-based predictive control**.

Again, the FMI standards allow Modelica models to be exported as FMI-compliant units (FMUs) as stand-alone simulation programs portable to any environment (Brembeck 2019; Lange et al. 2021) including embedded systems (Bertsch et al. 2015) and control applications (Franke et al. 2015).

10.1.6 Various open-source and commercial libraries in power-system (related-) domain(s)

At the early phase of the evolve of Modelica, already first works for modeling specific aspects in power systems (e.g. transient simulations, steady state analysis, etc.) have been implemented (Bachmann and Wiesmann 2000; Kalaschnikow 2002).

Through the maturity Modelica has achieved with respect to object-oriented facilities and other significant language features, further modern power systems(-related) libraries have provided novel solutions that cover many power systems modeling applications in one single framework, cf. Chapter 8.

10.1.7 Further useful libraries

In addition to modeling libraries concerned with classical power systems modeling applications, Section 8 summarized additional libraries that cover various aspects, e.g. renewable energy, buildings and districts, required for realizing modern power systems modeling applications. Section 8.3 provides a comprehensive list of libraries that are not directly related to power systems, still could be useful in some aspects, e.g. stochastic simulations.

The **Modelica Standard Library**, actively maintained and progressed by the Modelica Association, contains a considerable amount of useful components, cf. Section 7.2. Additionally, a considerable amount of third-party covering fundamental modeling paradigms up to optimization and unit testing exist, cf. Section 8.3.

10.1.8 Modelica for power system modeling applications

For an extended discussion regarding the suitability of Modelica for power system modeling applications, the reader is conferred to (Gómez et al. 2020). Francisco J. Gómez and et al. have extensively discussed and highlighted the feasibility of the Modelica language for fulfilling and formalizing functional and technical requirements addressed by the European Network of Transmission System Operators of Electricity (ENTSO-E), cf. `www.entose.eu/Pages/default.aspx`).

10.2 Challenges and future directions

In addition to the increasing demand on improving simulation runtime performance of large-scale systems, cf. Section 9.2, progress of the ongoing state-of-the-art methodologies and activities can be categorized as follows:

1. Enabling and developing **further modeling libraries**, in particular for cyber physical systems (Zimmer 2016), e.g. stochastic, communications, control etc. with which Modelica wins further ground in the potential scope of applications
2. Conceptualization and implementation of simulation-based mathematical algorithms and tools with which model-based applications can be realized
3. Realization of model-based applications in particular with FMI e.g. (Gräber et al. 2012; Nouidui and Wetter 2014)

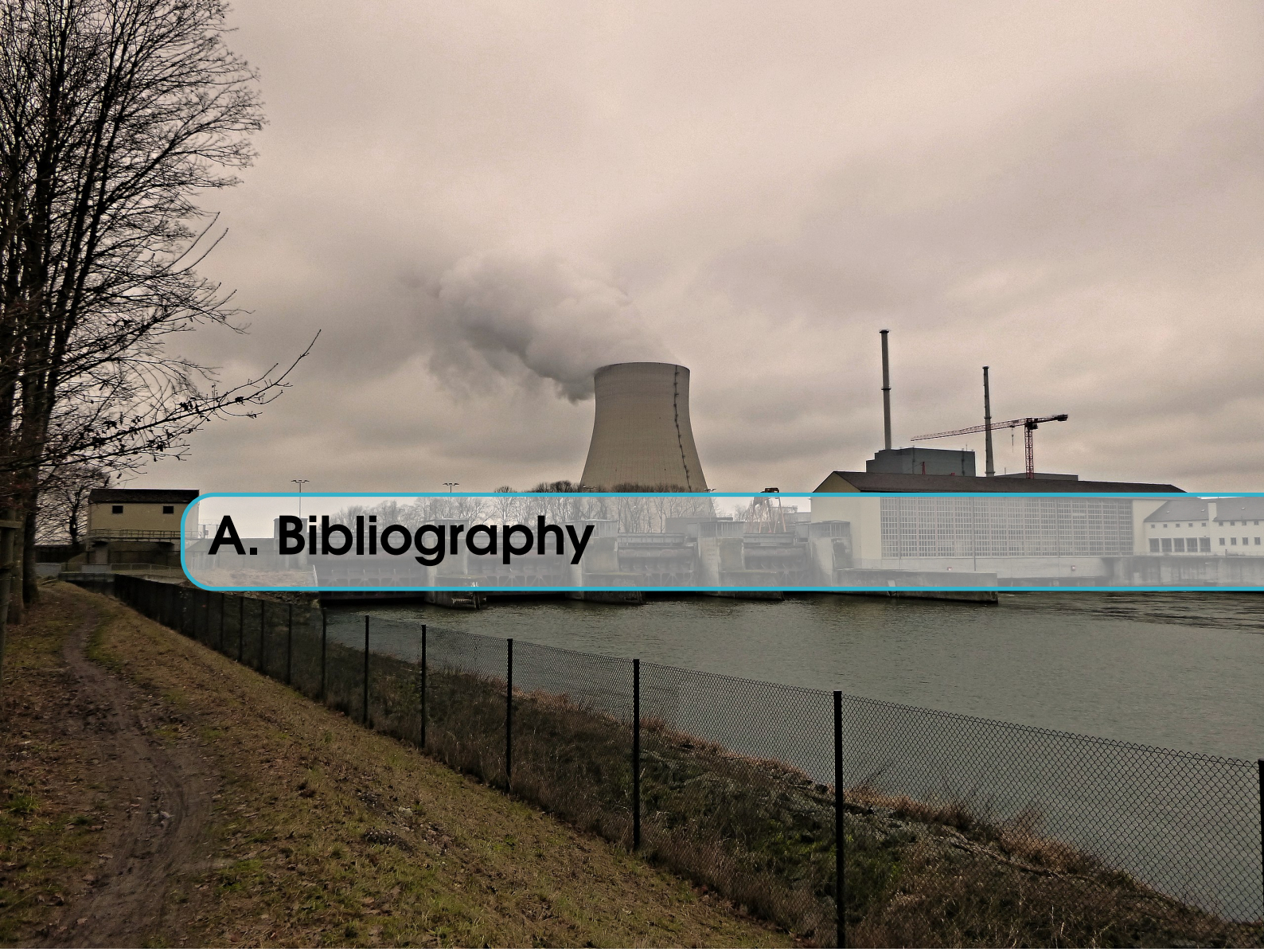
Further research topics attempt to **extend the Modelica language** with additional capabilities not supported yet. These are non-standardized attempts for inserting additional language features for describing[1]:

- **Partial Differential Equations** (Saldamli, Fritzson, and Bachmann 2002; Šilar, Ježek, and Kofránek 2018)
- **Numerical optimization** problems (Åkesson 2008)
- **Uncertainty quantification** (Bouskela et al. 2011)
- **Parallelization** (Gebremedhin et al. 2012)
- **Variable structure dynamics** (Benveniste et al. 2019; Zimmer 2013)
- Message passing communication for **DEVS formalism** (Sanz and Urquia 2016)
- **Stochastic differential equations** incorporating Wiener and Poisson generators externally called from Modelica code (Gevorkyan et al. 2016) [2]
- **Julia**-based Modelica package (Elmqvist, Henningsson, and Otter 2017)
- Modelica-based **meta programming** for symbolic computations, model transformation, compiler specification among others (Fritzson et al. 2019)
- **Model selection** (Brüger 2019)
- **Dynamic parameter sensitivities** (Elsheikh 2019)

[1] Items are sorted according to publication date of the chosen reference (not necessarily the first-ever possible reference)

[2] The Modelica libraries Noise and `AdvancedNoise` are also acknowledged for modeling of stochastic processes

Do you think that there are other categories or items that should be listed above?



A. Bibliography

Abel, Dirk (2010). "Object-Oriented Modelling for Simulation and Control of Energy Transformation Processes". In: *Production Factor Mathematics*. Ed. by Martin Grötschel, Klaus Lucas, and Volker Mehrmann. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 327–344 (cit. on p. 30).

ACOSAR (2015-2018). *Advanced Co-simulation Open System ARchitecture*. https://itea3.org/project/acosar.html (cit. on p. 69).

Åkesson, J. (2008). "Optimica – An Extension of Modelica Supporting Dynamic Optimization". In: *Modelica'2008: the 6th International Modelica Conference*. Bielefeld, Germany (cit. on p. 98).

Altes-Buch, Queralt, Sylvain Quoilin, and Vinvent Lemort (June 2018). "Modeling and control of CHP generation for greenhouse cultivation including thermal energy storage". In: *In Proceedings of the 31st International Conference on Efficiency, Cost, Optimization, Simulation and Environmental Impact of Energy Systems*. 13. Guimaraes, Portugal (cit. on p. 81).

Altes-Buch, Queralt, Sylvain Quoilin, and Vinvent Lemort (Mar. 2019). "Greenhouses: A Modelica library for the simulation of Greenhouse climate and energy systems". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 81).

Andresen, A. et al. (2015). "Status of the TransiEnt Library: Transient simulationof coupled energy networks with high share of renewable energy". In: *Modelica'2015: The 11th International Modelica Conference*. Paris, France (cit. on p. 78).

Andresen, A. et al. (Mar. 2019). "Status of the TransiEnt Library: Transient Simulation of Coupled Energy Networks with High Share of Renewable Energy". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 78).

Andresen, Lisa, Carsten Bode, and Gerhard Schmitz (2018). “Dynamic simulation of different transport options of renewable hydrogen to a refinery in a coupled energy system approach”. In: *International Journal of Hydrogen Energy* 43.42, pp. 19600 –19614 (cit. on p. 78).

Araujo, Antonio Emilio Angueth de and Danny Augusto Vieira Tonidandel (Apr. 2013). “Steinmetz and the concept of Phasor: A forgotten story”. In: *International Journal of Control, Automation and Electrical Systems* 24, pp. 388–395 (cit. on p. 79).

Årzén, K.-E. (1996). “Grafchart: A Graphical Language for Sequential Supervisory Control Applications”. In: *IFAC Proceedings Volumes* 29.1. 13th World Congress of IFAC, 1996, San Francisco USA, 30 June - 5 July, pp. 4831 –4836 (cit. on p. 65).

Ashenden, Peter J., Gregory D. Peterson, and Darrell A. Teegarden (2003). *The System Designer’s Guide to VHDL-AMS*. Morgan Kaufmann (cit. on p. 16).

Ashgar, Adeel and et al. (Mar. 2011). “An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation”. In: *Modelica’2011, the 8th International Modelica Conference*. Dresden, Germany (cit. on p. 55).

Åström, Karl Johan, Hilding Elmqvist, and Sven Erik Mattsson (1998). “Evolution of Continuous-Time Modeling and Simulation”. In: *ESM’1998: The 12th European Simulation Multiconference - Simulation - Past, Present and Future*. Manchester, United Kingdom (cit. on pp. 27, 29).

Augustin, Donald C. et al. (1967). “The SCi Continuous System Simulation Language (CSSL)”. In: *SIMULATION* 9.6, pp. 281–303 (cit. on p. 29).

Bachmann, Bernhard and Hansjürg Wiesmann (2000). “Advanced Modeling of Electromagnetic Transients in Power Systems”. In: *Modelica’2000: The Modelica Workshop*. Lund, Sweden (cit. on pp. 75, 78, 97).

Baetens, R. et al. (Dec. 2015). “OPENIDEAS – an open framework for integrated district energy”. In: *Proceedings of BS2015*. Hyderabad, India (cit. on p. 81).

Bartolini, A., F. Casella, and A. Guironnet (2019). “Towards Pan-European Power Grid Modelling in Modelica: Design Principles and a Prototype for a Reference Power System Library”. In: *Modelica’2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on pp. 33, 77).

Basciotti, Daniele and Oliver Pol (2011). “A theotetical study of the impact of using small-scale thermo chemical storage units in district heating Networks”. In: *The 2011 International Conference for Sustainable Energy Storage* (cit. on p. 80).

Baudette, M. et al. (2018). “OpenIPSL: Open-Instance Power System Library – Update 1.5 to "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations”. In: *SoftwareX* 7, pp. 34–36 (cit. on p. 77).

Baur, Marcus, Martin Otter, and Bernhard Thiele (2009). “Modelica libraries for linear control systems”. In: *Modelica’2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 83).

Bellmann, T. (2009). “Interactive simulations and advanced visualization with Modelica”. In: *Modelica’2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 83).

Beltrame, Tamara and F. E. Cellier (Sept. 2006). “Quantised State System Simulation in Dymola/Modelica using the DEVS Formalism”. In: *Modelica’2006: The 5th International Modelica Conference*. Vienna, Austria (cit. on p. 84).

Benveniste, Albert et al. (2019). “Multi-Mode DAE Models - Challenges, Theory and Implementation”. In: *Computing and Software Science: State of the Art and Perspectives*. Ed. by Bernhard Steffen and Gerhard Woeginger. Cham: Springer International Publishing, pp. 283–310 (cit. on p. 98).

Bergero, F. et al. (2012). “Simulating Modelica models with a Stand-Alone Quantized State Systems Solver”. In: *Modelica’2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 94).

Bergero, F. M. et al. (2018). “On the efficiency of quantization-based integration methods for building simulation”. In: *Building Simulation* 11, pp. 405–418 (cit. on p. 94).

Bertsch, Christian et al. (2015). “FMI for physical models on automotive embedded targets”. In: *Modelica’2015: The 11th International Modelica Conference*. Paris, France (cit. on p. 97).

Blochwitz, T. and et al. (2011). “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *Modelica’2011: The 8th International Modelica Conference*. Dresden, Germany (cit. on p. 15).

Blochwitz, T. and et al. (Sept. 2012). “Functional mockup interfae 2.0: The standard for tool independent exchange of simulation models”. In: *Modelica’2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 66).

Blochwitz, T. et al. (2011). “The Functional Mockup Interface for Tool independent Exchange of Simulation Models”. In: *Modelica’2011: The 8th International Modelica Conference*. Dresden, Germany (cit. on p. 66).

Bode, Carsten and Gerhard Schmitz (2018). “Dynamic Simulation and Comparison of Different Configurations for a Coupled Energy System with 100% Renewables”. In: *Energy Procedia* 155. 12th International Renewable Energy Storage Conference, IRES 2018, 13-15 March 2018, Düsseldorf, Germany, pp. 412 –430 (cit. on p. 78).

Bollinger, L.A. et al. (2018). “Multi-model ecologies for shaping future energy systems: Design patterns and development paths”. In: *Renewable and Sustainable Energy Reviews* 82, pp. 3441 –3451 (cit. on p. 80).

Bonaventura, L. et al. (2020). “A self adjusting multirate algorithm for robust time discretization of partial differential equations”. In: *Computers & Mathematics with Applications* 79.7. Advanced Computational methods for PDEs, pp. 2086 –2098 (cit. on p. 93).

Bonvini, Marco, Michael Wetter, and Thierry Stephane Nouidui (Sept. 2014). “A Modelica package for building-to-electrical grid integration”. In: *BauSim’2014: The 5th BauSim Conference*. Aachen, Germany (cit. on p. 80).

Bouskela, Daniel et al. (2011). “Modelling of uncertainities with Modelica”. In: *Modelica’2011: The 8th International Modelica Conference*. Dresden, Germany (cit. on p. 98).

Braun, W., F. Casella, and B. Bachmann (2017). “Solving large-scale Modelica models: new approaches and experimental results using OpenModelica”. In: *Modelica’2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on p. 92).

Brembeck, Jonathan (Oct. 2019). “A Physical Model-Based Observer Framework for Nonlinear Constrained State Estimation Applied to Battery State Estimation”. In: *Sensors* 19(20):4402 (cit. on p. 97).

Brkic, Jovan et al. (2019). “Open Source PhotoVoltaics Library for Systemic Investigations”. In: *Modelica’2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 77).

Brüger, Christoff (2019). “Modelica language extensions for practical non-monotonic modelling: on the need for selective model extension”. In: *Modelica’2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 98).

Brunnemann, J. et al. (2012). “Status of ClaRaCCS: Modelling and Simulationof Coal-Fired Power Plants with CO2 Capture”. In: *Modelica’2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 76).

Calle, Alberto De la et al. (Sept. 2018). “SolarTherm: A New Modelica Library and Simulation Platform for Concentrating Solar Thermal Power Systems”. In: *The 9th Eurosim Congress*. Oulu, Finland (cit. on p. 78).

Casella, F., A. Leva, and A. Bartolini (2017). “Simulation of large grids in OpenModelica: reflections and perspectives”. In: *Modelica’2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on pp. 77, 79, 92).

Casella, F. and C. Richter (2008). “ExternalMedia: A Library for Easy Re-Use of External Fluid Property Code in Modelica”. In: *Modelica’2008: the 6th International Modelica Conference*. Bielefeld, Germany (cit. on p. 82).

Casella, F. et al. (Sept. 2006). “The Modelica Fluid and Media library fpr modeling of incompressible and compressible thermo-fluid pipe networks”. In: *Modelica’2006: The 5th International Modelica Conference*. Vienna, Austria (cit. on p. 65).

Casella, Francesco (2015a). “Efficient Computation of State Derivatives for Multi-Rate Integration of Object-Oriented Models”. In: *IFAC-PapersOnLine* 48.1. 8th Vienna International Conferenceon Mathematical Modelling, pp. 262 –267 (cit. on p. 93).

Casella, Francesco (2015b). “Simulation of Large-Scale Models in Modelica: State of the Art and Future Perspective”. In: *Modelica’2015: The 11th International Modelica Conference*. Paris, France (cit. on pp. 84, 90–92).

Casella, Francesco and Bernhard Bachmann (2021). “On the choice of initial guesses for the Newton-Raphson algorithm”. In: *Applied Mathematics and Computation* 398, p. 125991 (cit. on p. 29).

Casella, Francesco and Alberto Leva (Aug. 2006). “Modelling of thermo-hydraulic power generation processes using Modelica”. In: *Mathematical and Computer Modelling of Dynamical Systems* 12.1, pp. 19–33 (cit. on p. 78).

CAUJOLLE, Mathieu et al. (June 2019). “HYBRID AC AND DC DISTRIBUTION NETWORKS MODELLING AND PLANNING USING EPSL MODELICA LIBRARY: PRELIMINARY RESULTS”. In:

Proceedings of the 25th International Conference on Electricity Distribution. Madrid, Spain (cit. on p. 85).

Cellier, F. E. (1991). *Continuous System Modeling*. Springer Verlag (cit. on pp. 25–27, 71).

Cellier, F. E. (2008). "World3 in Modelica: Creating System Dynamics Models in the Modelica framework". In: *Modelica'2008: the 6th International Modelica Conference*. Bielefeld, Germany (cit. on p. 84).

Cellier, F. E., C. Clauß, and A. Urquia (Sept. 2007). "Electronic circuit modeling and simulation in Modelica". In: *Congress on Modelling and Simulation*. Ljublijana (cit. on p. 64).

Cellier, F. E. and H. Elmqvist (1993). "Automated formula manipulation supports object-oriented continuous-system modeling". In: *IEEE Control Systems Magazine* 13.2, pp. 28–38 (cit. on p. 28).

Cellier, Francois E. and Ernesto Kofman (2006). *Continuous System Simulation*. Berlin, Heidelberg: Springer-Verlag (cit. on pp. 28, 71).

Ceriani, N. M. et al. (July 2013). "An approximate dynamic pro-gramming approach to the energy man-agementof a building cooling system". In: *European Control Conference*. Zurich, Switzerland, pp. 2026 –2031 (cit. on p. 94).

Chieh, A., P. Panciatici, and D. Picard (June 2011). "Power system modeling in Modelica for time-domain simulation". In: *2011 IEEE Trondheim PowerTech*. Trondheim, Norway, pp. 1–8 (cit. on p. 68).

Clauß, C. et al. (2000). "Modelling of electronic circuits with Modelica". In: *Modelica'2000: The Modelica Workshop*. Lund, Sweden (cit. on p. 64).

Costa, Andrea et al. (2020). "INDIGO Project: A Simulation Based Approach To Support District Cooling Design And Operation". English. In: *Proceedings of the 16th IBPSA Conference*. Ed. by V. Corrado et al. 16th IBPSA International Conference and Exhibition, Building Simulation 2019, BS 2019 ; Conference date: 02-09-2019 Through 04-09-2019. International Building Performance Simulation Association - IBPSA, pp. 1913–1920 (cit. on p. 69).

David, Rene and Hassane Alla (1992). *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Prentice Hall (cit. on p. 65).

del Hoyo Arce, Itzal et al. (2018). "Models for fast modelling of district heating and cooling networks". In: *Renewable and Sustainable Energy Reviews* 82, pp. 1863 –1873 (cit. on p. 81).

Di Pietro, Franco et al. (2020). "Mixed-mode state-time discretization in ODE numerical integration". In: *Journal of Computational and Applied Mathematics* 377 (cit. on p. 94).

Dommel, Hermann V. (Apr. 1969). "Digital Computer Solution of Electromagnetic Transients in Single-and Multiphase Networks". In: *IEEE Transactions on Power Apparatus and Systems* PAS-88.4, pp. 388–399 (cit. on p. 21).

Dorado-Rojas, Sergio A. et al. (Mar. 2020). "Performance Benchmark of Modelica Time-Domain Power System Automated Simulations using Python". In: *Proceedings of the American Modelica Conference 2020*. Boulder, Colorado, USA (cit. on p. 92).

Dubucq, Pascal and Günter Ackermann (2017). "Optimal Use of Energy Storage Potentials in a Renewable Energy System with District Heating". In: *Energy Procedia* 135. 11th International Renewable Energy Storage Conference, IRES 2017, 14-16 March 2017, Düsseldorf, Germany, pp. 158 –171 (cit. on p. 78).

Duff, Iain S., Albert M. Erisman, and John K. Reid (2017). *Direct Methods for Sparse Matrices*. Oxford University Applied Mathematics Research eXpress (cit. on p. 90).

DYNCAP: (2011-2014). *Dynamic Capture of CO2 – Dynamic investigation of steam power processes with CO2 capturing for providing balancing energy*. http://www.kraftwerkforschung.info/en/mehr-flexibilitaet-fuer-emissionsarme-kohlekraftwerke (cit. on p. 68).

DYNSTART (2015-2019). *Start-up behavir of Power Plants - Subtopic CO2*. https://www.tuhh.de/alt/technische-thermodynamik/research/recent-projects/dynstart.html (cit. on p. 69).

Eberhart, Philip et al. (2015). "Open Source Library for the Simulation of Wind Power Plants". In: *Modelica'2015: The 11th International Modelica Conference*. Paris, France (cit. on p. 79).

Eich-Soellner, Edda and Claus Führer (1998). *Numerical Methods in Multibody Dynamics*. Springer Fachmedien Wiesbaden (cit. on p. 92).

Elmqvist, H. (1978). "A structured model language for large continuous systems". PhD thesis. Lund Institute of Technology, Lund, Sweden (cit. on p. 27).

Elmqvist, H., S. E. Mattsson, and M. Otter (2001). "Object-oriented and hybrid modeling in Modelica". In: *Journal EuropÃ©en des systÃ¨mes automatisÃ©s* 35.1, pp. 1–X (cit. on p. 65).

Elmqvist, H. and M. Otter (June 1994). "Methods for tearing systems of equations in object oriented modeling". In: *ESM'94: European Simulation Multiconference*. Barcelona, Spain (cit. on p. 28).

Elmqvist, H., H. Tummescheit, and M. Otter (2003). “Object-oriented modeling of thermo-fluid systems”. In: *Modelica’2003: The 3rd International Modelica Conference*. Linköping, Sweden (cit. on p. 65).
Elmqvist, Hilding (2014). “Modelica Evolution - From My Perspective”. In: *Modelica’2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on pp. 16, 28, 29).
Elmqvist, Hilding, Toivo Henningsson, and Martin Otter (2017). “Innovations for Future Modelica”. In: *Modelica’2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on p. 98).
Elmqvist, Hilding and Sven Erik Mattsson (1997). “Modelica - The next generation modeling language: an International design effort”. en. In: *ESS97: The 9th European Simulation Symposium*. Passau, Germany (cit. on p. 29).
Elmqvist, Holding, Martin Otter, and Sven Erik Mattsson (2012). “Fundamentals of synchronous control in Modelica”. In: *Modelica’2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 63).
Elsheikh, A., E. Widl, and P. Palensky (2012a). “Simulating complex energy systems with Modelica: A primary evaluation”. In: *2012 6th IEEE International Conference on Digital Ecosystems Technologies (DEST)*. IEEE, pp. 1–6 (cit. on p. 94).
Elsheikh, Atiyah (2014). “Modeling parameter sensitivities using equation-based algorithmic differentiation techniques: The ADMSL.Electrical.Analog library”. In: *Modelica’2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 82).
Elsheikh, Atiyah (2015). “An equation-based algorithmic differentiation technique for differential algebraic equations”. In: *Journal of Computational and applied Mathematics* 281, pp. 135 –151 (cit. on p. 82).
Elsheikh, Atiyah (2019). “Dynamic Parameter Sensitivities: Summary of computation methods for continuous-time Modelica models”. In: *Modelica’2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 98).
Elsheikh, Atiyah, Edmund Widl, and Peter Palensky (2012b). “Simulating complex energy systems with Modelica: A primary evaluation”. In: *DEST’2012: The 6th IEEE International Conference on Digital Ecosystems and Technologies*. Campione d’Italia, Italy (cit. on p. 89).
Elsheikh, Atiyah et al. (2013). “Modelica-enabled rapid prototyping via TRNSYS”. In: *BS’2013. The 13th International Conference of the International Building Performance Simulation Association*. Chambéry, France (cit. on p. 96).
EMPHYSIS (2017-2020). *Embedded systems with physical models in the production code software*. https://emphysis.github.io (cit. on p. 69).
Enge, Olaf et al. (Sept. 2006). “Quasi-stationary AC analysis using phasor description with Modelica”. In: *Modelica’2006: The 5th International Modelica Conference*. Vienna, Austria (cit. on p. 76).
EUROSYSLIB (2007-2010). *European Leaderhip in System Modeling and Simulation through advanced Modelica Libraries*. https://itea3.org/project/eurosyslib.html (cit. on p. 68).
Fernström, A. et al. (Mar. 2006). “OMNotebook - Interactive WYSIWYG Book Software for Teaching MetaProgramming”. In: *In Proceedings of the Workshop on Developing Computer Science Education – How Can It Be Done?* (Cit. on p. 72).
Ferreira, J.A. and J.P. Estima de Oliveira (1999). “Modelling hybrid systems using statecharts and Modelica”. In: *The 7th IEEE International Conference on Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA ’99.* Vol. 2, 1063 –1069 vol.2 (cit. on pp. 63, 65).
Filippo, Jenny Montbrun-Di et al. (1991). “A survey of bond graphs : Theory, applications and programs”. In: *Journal of the Franklin Institute* 328.5, pp. 565 –606 (cit. on p. 27).
Floros, X., F. E. Cellier, and E. Kofman (2010). “Discretizing Time or States?: A Comparative Study between DASSL and QSS (Work in Progress Paper)”. In: *EOOLT’2010: The 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Oslo, Norway (cit. on p. 94).
Floros, Xenofon et al. (2014). “Simulation of Smart-Grid Models using Quantization-Based Integration Methods”. In: *Modelica’2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 94).
Forrester, J. W. (1961). *Industrial Dynamics*. M.I.T. Press (cit. on p. 82).
Forrester, J. W. (1969). *Urban Dynamics*. M.I.T. Press (cit. on p. 84).
Forrester, J. W. (1971a). *Principles of Systems*. M.I.T. Press (cit. on p. 84).
Forrester, J. W. (1971b). *World Dynamics*. M.I.T. Press (cit. on p. 84).
Franke, Rüdiger and Hansjürg Wiesmann (2014). “Flexible modeling of electrical power systems – the Modelica PowerSystems library”. In: *Modelica’2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on pp. 33, 58, 77, 78).

Franke, Rüdiger et al. (2009a). "Standardization of themo-fluid modeling in Modelica.Fluid". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 65).

Franke, Rüdiger et al. (2009b). "Stream connectors – An extension of Modelica for device-oriented modeling of convective transport phenomena". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on pp. 39, 63).

Franke, Rüdiger et al. (2015). "Model-based Control with FMI and a C++ runtime for Modelica". In: *Modelica'2015: The 11th International Modelica Conference*. Paris, France (cit. on pp. 66, 97).

Frayssinet, Loïc et al. (2017). "Adaptation of building envelope models for energy simulation at district scale". In: *Energy Procedia* 122. CISBAT 2017 International ConferenceFuture Buildings & Districts – Energy Efficiency from Nano to Urban Scale, pp. 307–312 (cit. on p. 80).

Fritzson, Peter (2015). *Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach*. IEEE Press Wiley (cit. on pp. 71, 72).

Fritzson, Peter et al. (Mar. 2019). "MetaModelica – A Symbolic-Numeric Modelica Language and Comparison to Julia". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 98).

Fritzson, Peter et al. (2020). "The OpenModelica Integrated Environment for Modeling, Simulation, and Model-Based Development". In: *Modeling, Identification and Control* 41.4, pp. 241–295 (cit. on pp. 55, 69, 70, 72).

Gebremedhin, Mahder et al. (2012). "A data-parallel algorithmic Modelica extension for efficient execution on multi-core platforms". In: *Modelica'2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 98).

Gevorkyan, Migran et al. (2016). "The Stochastic Processes Generation in OpenModelica". In: *Distributed Computer and Communication Networks*. Ed. by Vladimir M. Vishnevskiy, Konstantin E. Samouylov, and Dmitry V. Kozyrev. Cham: Springer International Publishing, pp. 538–552 (cit. on p. 98).

Gomes, Cláudio et al. (May 2018). "Co-Simulation: A Survey". In: *ACM Computing Surveys* 51 (cit. on pp. 15, 96).

Gómez, Francisco J. et al. (2020). "Software requirements for interoperable and standard-based power system modeling tools". In: *Simulation Modelling Practice and Theory* 103, p. 102095 (cit. on pp. 67, 98).

Gottelt, F., T. Hoppe, and L. Nielsen (2017). "Applying the Power Plant Library ClaRa for Control Optimisation". In: *Modelica'2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on p. 76).

gPROMS. *Process Systems Enterprise*. http://www.psenterprise.com/gproms/index.html (cit. on p. 16).

Gräber, Manual et al. (2012). "Using functional mocku-up units for nonlinear model predictive control". In: *Modelica'2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 98).

Gupta, B. R. (2005). *Power Systems Analysis and Design*. S Chand & Co Ltd (cit. on p. 73).

Hafez, Ahmed and Andrew Forsyth (Apr. 2009). "A Review of More-Electric Aircraft". In: *13th International Conference on Aerospace Sciences and Aviation Technology* 13 (cit. on p. 15).

Halder, Asim, Nitai Pal, and Debasish Mondal (2018). "Transient Stability Analysis of a Multimachine Power System with TCSC Controller – A Zero Dynamic Design Approach". In: *International Journal of Electrical Power & Energy Systems* 97, pp. 51 –71 (cit. on p. 21).

Haumer, A. et al. (2008). "Quasi-Stationary modeling and simulation of electrical circuits using complex phasors". In: *Modelica'2008: the 6th International Modelica Conference*. Bielefeld, Germany (cit. on p. 64).

Haumer, A. et al. (2009). "The AdvancedMachines library: Loss models for electric". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 64).

Heckel, Jan-Peter and Christian Becker (Mar. 2019). "Advanced Modeling of Electric Components in Integrated EnergySystems with the TransiEnt Library". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 78).

Henningsson, Erik, Hans Olsson, and Luigi Vanfretti (2019). "DAE Solvers for Large-Scale Hybrid Models". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 92).

Hiskens, Ian A. and M. A. Pai (Feb. 2000). "Trajectory Sensitivity Analysis of Hybrid Systems". In: *IEEE Transactions on Circuits and Systems – Part I: Fundamental Theory and Applications* 47.2 (cit. on p. 22).

Höger, C. (Sept. 2013). "Sparse Causalisation of Differential Algebraic Equations for Efficient Event Detection". In: *The 8th EUROSIM Congress on Modelling and Simulation*. Cardiff, UK: IEEE Computer Society, pp. 351–356 (cit. on p. 94).

Hölemann, Sebastian and Dirk Abel (Sept. 2009). "Modelica predective control - An MPC library for Modelica". In: *Automatisierungstechnik* 57.4, pp. 187–194 (cit. on p. 83).
Huang, Qiuhua (Nov. 2016). "Electromagnetic Transient and Electromechanical Transient Stability Hybrid Simulation: Design, Development and its Applications". PhD thesis (cit. on p. 21).
IBPSA Project 1 (2018-2022). *BIM/GIS and Modelica Framework for building and community energy system design and operation*. https://ibpsa.github.io/project1/index.html (cit. on p. 69).
IEA EBC Annex 60 (2012-2017). *New generation computational tools for building and community energy systems based on the Modelica and Functional Mockup Interface standards*. www.iea-annex60.org/index.htmll (cit. on p. 69).
INDIGO (2016-2020). *New Generation of Intelligent Efficient District Cooling Systems*. https://www.indigo-project.eu/ (cit. on p. 69).
ITESLA (2012-2016). *Innovative Tools for Electrical System Security within Large Areas*. https://cordis.europa.eu/project/id/283012 (cit. on p. 69).
Joos, A., K. Dietl, and G. Schmitz (2009). "Thermal Separation: An Approach for a Modelica Library for Absorption, Adsorption and Rectification". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 84).
Jorissen, F. et al. (2018). "Implementation and Verification of the IDEAS Building Energy Simulation Library". In: *Journal of Building Performance Simulation* 11.6, pp. 669–688 (cit. on p. 81).
Kalaschnikow, Sergej N. (2002). "PQLib - A Modelica library for power quality analysis in networks". In: *Modelica'2002: The 2nd International Modelica Conference*. Munich, Germany (cit. on p. 97).
Karnavas, Yannis L and Eleytherios I Lygouras (2018). "Synchronous machine analysis and modelling in LabVIEW: An educational tool for transient stability studies". In: *The International Journal of Electrical Engineering & Education* 57.3, pp. 202–229 (cit. on p. 30).
Klöckner, Andreas, Andreas Knoblach, and Andreas Heckmann (2017). "How to shape noise spectra for continuous system simulation". In: *Mathematical and Computer Modelling of Dynamical Systems* 23.3, pp. 284–300 (cit. on p. 82).
Klökner, Andreas, Franciscus L. J. van der Linden, and Dirk Zimmer (2014). "Noise Generation for Continuous System Simulation". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 64).
Kofman, E. (2006). "A Third Order Discrete Event Simulation Method for Continuous System Simulation". In: *Latin American Applied Research* 36.2, pp. 101–108 (cit. on p. 94).
Kofman, Ernesto (2002). "A Second-Order Approximation for DEVS Simulation of Continuous Systems". In: *Simulation* 78.2, pp. 76–89 (cit. on p. 94).
Kofman, Ernesto (May 2004). "Discrete Event Simulation of Hybrid Systems". In: *SIAM Journal of Scientific Computing* 25.5, pp. 1771–1797 (cit. on p. 94).
Kofman, Ernesto and Sergio Junco (2001). "Quantized-state systems: A DEVS approach for continuous system simulation". In: *Simulation: Transactions of the Society for Computer Simulation International* 18.3, pp. 123–132 (cit. on p. 94).
Köhler, Jochen et al. (May 2016). "Modelica Association Project System Structure and Parameterization – Early Insights". In: *The First Japanese Modelica Conferences*. Tokyo, Japan (cit. on p. 68).
Kolmogorov, Andrey (1956). "On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables". In: *Proceedings of the USSR Academy of Sciences* 108. (Russian), pp. 179–182 (cit. on p. 25).
Kothari, D. P. and I. J. Nagrath (2014). *Modern Power System Analysis*. Mcgraw Hill Education Private Limited, New Delhi (cit. on p. 73).
Kral, C. and A. Haumer (2005). "Modelica libraries for dc machines, three phase and polyphase machines". In: *Modelica'2005: The 4th International Modelica Conference*. Hamburg, Germany (cit. on p. 64).
Kral, C. and A. Haumer (2011a). "Object Oriented Modeling of Rotating Electrical Machines". In: InTech. Chap. Advances in Computer Science and Engineering, pp. 135–160 (cit. on p. 30).
Kral, C. and A. Haumer (2011b). "The new FundamentalWave library for modeling rotating electrical three phase machines". In: *Modelica'2011: The 8th International Modelica Conference*. Dresden, Germany (cit. on p. 65).
Krammer, Martin et al. (Mar. 2019). "Standardized Integration of Real-Time andNon-Real-Time Systems: The Distributed Co-SimulationProtocol". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 68).

Kron, G. (1963). *Duakoptics - The Piecewise Solution of Large-scale Systems*. London, UK: MacDonald & Co. (cit. on p. 28).

Kundur, P., N.J. Balu, and M.G. Lauby (1994). *Power System Stability and Control*. Discussion Paper Series. McGraw-Hill Education (cit. on p. 15).

Lange, Ralph et al. (2021). "Integrating the Functional Mock-Up Interface with ROS and Gazebo". In: *Robot Operating System (ROS): The Complete Reference (Volume 5)*. Ed. by Anis Koubaa. Cham: Springer International Publishing, pp. 187–231 (cit. on p. 97).

Larsson, Mats (2000). "ObjectStab - A Modelica library for power system stability studies". In: *Modelica'2000: The Modelica Workshop*. Lund, Sweden (cit. on pp. 30, 76).

Leitner, Benedikt et al. (Sept. 2019). "A method for technical assessment of power-to-heat use cases to couple local district heating and electrical distribution grids". In: *Energy* 182, pp. 729–738 (cit. on pp. 22, 80).

Leitold, A. and Katalin M. Hangos (2001). "Structural solvability analysis of dynamic process models". In: *Computers & Chemical Engineering* 25, pp. 1633–1646 (cit. on p. 28).

Linden, Franciscus L. J. van der (2014). "General fault triggering architecture to trigger model faults in Modelica using a standardized blockset". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 83).

Link, K., H. Steuer, and A. Butterlin (2009). "Deficiencies of Modelica and its simulation environments for large fluid systems". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 91).

Lu, Xing et al. (Apr. 2019). "An Open Source Modeling Framework for Interdependent Energy-Transportation-Communication Infrastructure in Smart and Connected Communities". In: *IEEE Access* 7 (cit. on p. 80).

M. Otter, K.-E. and I. Dressler Årzén (2005). "StateGraph-A Modelica Library for Hierarchical State Machines". In: *Modelica'2005: The 4th International Modelica Conference*. Hamburg, Germany (cit. on pp. 63, 65).

Maffezzoni, Claudio, Roberto Girelli, and Petrika Lluka (1996). "Generating efficient computational procedures from declarative models". In: *Simulation Practice and Theory* (cit. on pp. 28, 58).

Majetta, K. et al. (2009a). "Improvement of MSL Electrical Analog Library". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 64).

Majetta, K. et al. (2009b). "SPICE3 Modelica library". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 64).

Majetta, K. et al. (2011). "MSL Electrical Spice3 - Status and further development". In: *Modelica'2011: The 8th International Modelica Conference*. Dresden, Germany (cit. on p. 64).

Mans, Michael et al. (Mar. 2019). "Automated model generation and simplification for district heating and cooling netwroks". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 80).

Marquardt, W. (1996). "Trends in computer-aided process modeling". In: *Computers & Chemical Engineering* 20.6. Fifth International Symposium on Process Systems Engineering, pp. 591 –609 (cit. on p. 29).

Marx-Schubach, Thomas and Gerhard Schmitz (2019). "Modeling and simulation of the start-up process of coal fired power plants with post-combustion CO2 capture". In: *International Journal of Greenhouse Gas Control* 87, pp. 44 –57 (cit. on pp. 76, 84).

Mattsson, Sven Erik and Gustaf Söderlind (1993). "Index reduction in differential-algebraic equations using dummy derivatives". In: *SIAM Journal on Scientific Computing* 14.3, pp. 677 –692 (cit. on p. 28).

Mehrmann, Volker and Lena Wunderlich (2009). "Hybrid systems of differential-algebraic equations – Analysis and numerical solution". In: *Journal of Process Control* 19.8. Special Section on Hybrid Systems: Modeling, Simulation and Optimization, pp. 1218 –1228 (cit. on p. 92).

Migoni, G. and E. Kofman (Dec. 2009). "Linearly ImplicitDiscrete Event Methods for Stiff ODEs". In: *Latin American Applied Research* (cit. on p. 94).

Migoni, G. et al. (2016). "Efficient simulation of Hybrid Renewable Energy Systems". In: *International Journal of Hydrogen Energy* 41.32, pp. 13934 –13949 (cit. on p. 94).

Minz, Markus, Linus Netze, and Antonello Monti (June 2016). "A multi-level approach to power system Modelica models". In: *2016 IEEE 17th Workshop on Control and Modeling for Power Electronics (COMPEL)*. Norwegian University of Science and Technology. Trondheim, Norway (cit. on p. 76).

MODELISAR (2008-2011). *From System Modeling to S/W running on the Vehicle*. https://itea3.org/project/modelisar.html (cit. on p. 68).

MODRIO (2012-2016). *Model Driven Physical Systems Operation*. https://itea3.org/project/modrio.html (cit. on p. 69).

Mosterman, Pieter J., Martin Otter, and Hilding Elmqvist (1998). "Modeling Petri Nets As Local Constraint Equations For Hybrid Systems Using Modelica". In: *In: proceedings of the Summer Computer Simulation Conference -98*, pp. 314–319 (cit. on p. 65).

Moudgalya, K. K. et al. (2017). "Large scale training through spoken tutorials to promote and use OpenModelica". In: *Modelica'2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on p. 72).

Moudgalya, K. M. (Oct. 2020). "ICT in Education and Implications for the Belt and Road Initiative". In: Singapore: Springer. Chap. ICT for Education, Employment and Empowerment in India, pp. 53–71 (cit. on p. 73).

Müller, D. et al. (Sept. 2016). "AixLib - An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework". In: *BAUSIM'2016 IBPSA Germany*. Dresden, Germany (cit. on p. 80).

Murad, M. A. A. et al. (2017). "Enhancing engineering studies in developing countries using OpenModelica". In: *2017 4th International Conference on Advances in Electrical Engineering (ICAEE)* (cit. on p. 77).

Murota, K. (1987). *Systems Analysis by Graphs and Matroids*. Berlin: Springer, p. 281 (cit. on p. 28).

Müller, Sven et al. (Mar. 2016). "Interfacing Power System and ICT Simulators: Challenges, State-of-the-Art, and Case Studies". In: *IEEE Transactions on Smart Grid* PP, pp. 1–1 (cit. on pp. 15, 96).

Nasar, S. A and F. C. Trutt (1998). *Electric Power Systems*. 1st. CRC Press (cit. on p. 73).

Nouidui, Thierry Stephane and Michael Wetter (2014). "Tool coupling for the design and operation of building energy and control systems based on the functional mock-up interface standard". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 98).

Nytsch-Geusen, Christoph and et al. (Sept. 2016). "BuildingSystems – Eine modular hierarchische Modell-Bibliothek zur energetischen Gebäude- und Anlagensimulation". In: *BAUSIM'2016 IBPSA Germany*. Dresden, Germany (cit. on p. 80).

Ochel, Lennart and et al. (2019). "OMSimulator – Integrated FMI and TLM-based Co-simulation with Composite Model Editing and SSP". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 16).

Oeding, Dietrich and Bernd Rüdiger Oswald (2016). *Elektrische Kraftwerke und Netze*. 8th Edition. Springer (cit. on p. 55).

Olsson, Hans et al. (2009). "Operator Overloading in Modelica 3.1". In: *Modelica'2009: The 7th International Modelica Conference*. Como, Italy (cit. on p. 63).

OPENCPS (2015-2018). *Open Cyber-Physical System Model-Driven Certified Development*. https://www.opencps.eu/ (cit. on p. 69).

OPENPROD (2009-2012). *Open Model-Driven Whole-Product Development and Simulation Environment*. https://itea3.org/project/openprod.html (cit. on p. 68).

Otter, M. and F. E. Cellier (1996). "Software for Modeling and Simulating Control Systems". In: ed. by ed W.S. Levine. Boca Raton, FL, US: CRC Press. Chap. The Control Handbook, pp. 415–428 (cit. on p. 26).

Otter, Martin (Sept. 2006). "The LinearSystems library for continuous and discrete control systems". In: *Modelica'2006: The 5th International Modelica Conference*. Vienna, Austria (cit. on p. 83).

Otter, Martin and et al. (2015). "Formal requirement modeling for Simulation-based verification". In: *Modelica'2015: The 11th International Modelica Conference*. Paris, France (cit. on p. 83).

Palensky, P. et al. (2017). "Cosimulation of Intelligent Power Systems: Fundamentals, Software Architecture, Numerics, and Coupling". In: *IEEE Industrial Electronics Magazine* 11.1, pp. 34–50 (cit. on p. 96).

Palensky, Peter, Edmund Widl, and Atiyah Elsheikh (2012). "Simulating cyber-physical energy systems: challenges, tools and methods". In: *IEEE Transactions on Systems, Man and Cybernetics – Part C: Applications and Reviews, S. I. on Industrial Applications of Distributed Intelligent Systems* (cit. on p. 89).

Palensky, Peter et al. (2013). "Modeling inteligent energy systems: Co-Simulation platform for validating flexible-demand EV charging management". In: *IEEE Transactions on Smart Grid, SI: Real-Time Demand Response* (cit. on p. 90).

Pantelides, Constantinos C. (Mar. 1988). "The Consistent Initialization of Differential-Algebraic Systems". In: *SIAM Journal on Scientific and Statistical Computing* 9.2, pp. 213–231 (cit. on p. 28).

Paynter, Henry M. (1961). *Analysis and Design of Engineering Systems*. Cambridge, Massachusetts: MIT Press (cit. on p. 26).

PEGASE (2008-2012). *Pan European Grid Advanced Simulation and state Estimation*. https://ieeexplore.ieee.org/document/6465783 (cit. on p. 68).

Perfumo, Cristian et al. (2012). "Load management: Model-based control of aggregate power for populations of thermostatically controlled loads". In: *Energy Conversion and Management* 55, pp. 36 –48 (cit. on p. 94).

Pfeiffer, A. et al. (2012). "PySimulator – A simulation and analysis environment in Python with plugin infrastructure". In: *Modelica'2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 67).

Pollok, Alexander, Andreas Klöckner, and Dirk Zimmer (2019). "Psychological aspects of equation-based modelling". In: *Mathematical and Computer Modelling of Dynamical Systems* 25.2 (cit. on p. 30).

Pop, A. et al. (2012). "Static and Dynamic Debugging of Modelica Models". In: *Modelica'2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 83).

Pop, A. et al. (Mar. 2019). "A new OpenModelica compiler high performance frontend". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 92).

Prabhu, J. A. X. et al. (2016). "Design of electrical system based on load flow analysis using ETAP for IEC projects". In: *2016 IEEE 6th International Conference on Power Systems (ICPS)*, pp. 1–6 (cit. on p. 20).

Prat, Victorino S., Alfonso Urquia, and Sebastian Dormido (Sept. 2006). "ARENALib: A Modelica library for discrete-event system simulation". In: *Modelica'2006: The 5th International Modelica Conference*. Vienna, Austria (cit. on p. 82).

Protopapadaki, Christina and Dirk Saelens (2017). "Heat pump and PV impact on residential low-voltage distribution grids as a function of building and district properties". In: *Applied Energy* 192, pp. 268 –281 (cit. on p. 81).

Protopapadaki, Christina and Dirk Saelens (2019). "Towards metamodeling the neighborhood-level grid impact of low-carbon technologies". In: *Energy and Buildings* 194, pp. 273 –288 (cit. on p. 81).

Ragazzini, J. R., R. H. Randall, and F. A. Russell (1947). "Analysis of Problems in Dynamics by Electronic Circuits". In: *Proceedings of the IRE* 35.5, pp. 444–452 (cit. on p. 25).

Rahrovi, Babak and Mehrdad Ehsani (Feb. 2019). "A Review of the More Electric Aircraft Power Electronics". In: pp. 1–6 (cit. on p. 15).

Rande, Akshay and F. Casella (Apr. 2014). "Multi-rate integration algorithms: a path towards efficient simulation of object-oriented models of very large systems". In: *EOOLT'2014: The 6h International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Berin, Germany (cit. on p. 93).

ResiliEntEE (2017-2021). *Resilience of integrated energy networks with a high share of renewable Energies*. https://www.tuhh.de/transient-ee/en/projectdescription.html (cit. on p. 69).

Rice, John R. (July 1960). "Split Runga-Kutta Method for Simultaneous Equations". In: *Journal of Research of the National Bureau of Standards – B. Mathematics and Mathematical Physics* 64B.3 (cit. on p. 93).

Richter, Marcel, Gerd Oeljeklaus, and Klaus Görner (2019). "Improving the load flexibility of coal-fired power plants by the integration of a thermal energy storage". In: *Applied Energy* 236, pp. 607 –621 (cit. on p. 76).

Roberts, Steven, Arash Sarshar, and Adrian Sandu (Nov. 2018). "Coupled Multirate Infinitesimal GARK Schemes for Stiff Systems with Multiple Time Scales". In: *SIAM Journal on Scientific Computing* 42, A1609–A1638 (cit. on p. 93).

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Philadelphia, Pennsylvania, USA: SIAM Press (cit. on p. 92).

Sabeeh, Bashar and Chin Gan (May 2016). "Power System Frequency Stability and Control: Survey". In: 11, pp. 5688–5695 (cit. on p. 22).

Saldamli, Levon, Peter Fritzson, and Bernhard Bachmann (2002). "Extending Modelica for partial differential equations". In: *Modelica'2002: The 2nd International Modelica Conference*. Munich, Germany (cit. on p. 98).

Sandelin, E. et al. (2003). "DrModelica an interactive tutoring environment for Modelica". In: *Modelica'2003: The 3rd International Modelica Conference*. Linköping, Sweden (cit. on p. 72).

Sanz, Victorino, Federico Bergero, and Alfonso Urquia (Apr. 2018). "An approach to agent-based modeling with Modelica". In: *Simulation Modelling Practive and Theory* 83, pp. 65–74 (cit. on p. 82).

Sanz, Victorino and Alfonso Urquia (Apr. 2016). "Modelica extensions for supporting message passing communication and dynamic data structures". In: *EOOLT'2016: The 7h International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Milan, Italy (cit. on p. 98).

Sanz, Victorino, Alfonso Urquia, and Francesco Casella (Apr. 2014). "Improving Efficiency of Hybrid System Simulation in Modelica". In: *EOOLT'2014: The 6h International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*. Berin, Germany (cit. on p. 94).

Sanz, Victorino, Alfonso Urquia, and Alberto Leva (2014). "1D/2D Cellular Automata Modeling with Modelica". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 82).

Sanz, Victorino, Alfonso Urquia, and Alberto Leva (May 2016). "CellularAutomataLib2 : improving the support for cellular automata modelling in Modelica". In: *Mathematical and Computer Modelling of Dynamical Systems* 22.3, pp. 244–264 (cit. on p. 82).

Sanz, Victorino et al. (Jan. 2012). "Modeling of hybrid control systems using the DEVSLIB Modelica library". In: *Control Engineering Practice* 20.1, pp. 24–34 (cit. on p. 82).

Sanz, Victorino et al. (Sept. 2013). "Hybrid system modeling using the SIMANLib and ARENALib Modelica libraries". In: *Simulation Modelling Practice and Theory* 37, pp. 1–17 (cit. on p. 82).

Savcenco, V. (2009). "Construction of a multirate RODAS method for stiff ODEs". In: *Journal of Computational and Applied Mathematics* 225.2, pp. 323–337 (cit. on p. 93).

Schiavo, Francesco and Francesco Casella (May 2007). "Object-oriented modelling and simulation of heat exchangers with finite element methods". In: *Mathematical and Computer Modelling of Dynamical Systems* 13.3, pp. 211–235 (cit. on p. 78).

Schneider, Georg Ferdinand et al. (2015). "Hardware-in-the-Loop-Simulation of a building energy and control system to investigate circulating pump control using Modelica". In: *Modelica'2015: The 11th International Modelica Conference*. Paris, France (cit. on p. 80).

Schweiger, G. et al. (June 2020a). "Co-Simulation - An Empirical Survey: Applications, Recent Developments and Future Challenges". In: *Simulation Notes Europe* 30.2, pp. 73–76 (cit. on p. 96).

Schweiger, Gerald et al. (2020b). "Modeling and simulation of large-scale systems: A systematic comparison of modeling paradigms". In: *Applied Mathematics and Computation* 365 (cit. on p. 30).

Sielemann, M., F. Casella, and M. Otter (Nov. 2013). "Robustness of declarative modeling languages: Improvements via probability-one homotopy". In: *Simulation Modelling Practice and Theory* 38, pp. 38–57 (cit. on p. 82).

Sielemann, M. and G. Schmitz (2011). "A quantitative metric for robustness of nonlinear algebraic equation solvers". In: *Mathematics and Computers in Simulation* 81.12, pp. 2673 –2687 (cit. on pp. 29, 82).

Sielemann, Michael (2012). "probability-one homotopy for Robust initialization of Differential-Algebraic Equations". In: *Modelica'2012: The 9th International Modelica Conference*. Munich, Germany (cit. on p. 82).

Steinmetz, Charles Proteus (1893). "Complex quantities and their use in Electrical Engineering". In: *Proceedings of the International Electrical Congress, Conferece of AIEE (American Institute of Electrical Engineers Proceedings)*. Chicago, USA, pp. 33–74 (cit. on p. 79).

Stevenson, William D. (2014). *Elements Of Power System Analysis*. 4th. Mcgraw Hill Education Private Limited, New Delhi (cit. on p. 73).

Sulligoi, G., A. Vicenzutti, and R. Menis (2016). "All-Electric Ship Design: From Electrical Propulsion to Integrated Electrical and Electronic Power Systems". In: *IEEE Transactions on Transportation Electrification* 2.4, pp. 507–521 (cit. on p. 15).

Thiele, B., M. Otter, and S. E. Mattsson (2014). "Modular Multi-Rate and Multi-Method Real-Time Simulation". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 93).

Thiele, B. et al. (2017). "Towards a standard-conform platform-generic and feature-rich Modelica device drivers library". In: *Modelica'2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on p. 83).

Thongam, J. S. et al. (2013). "All-electric ships – A review of the present state of the art". In: *2013 Eighth International Conference and Exhibition on Ecological Vehicles and Renewable Energies (EVER)*, pp. 1–8 (cit. on p. 15).

Tiller, Michael and Dietmar Winkler (2014). "impact – A Modelica Package Manager". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 75).

TransiEnt (2013-2017). *Transient Behavior of Integrated Energy Networks with a High Share of Renewable Energies*. https://www.tuhh.de/transient-ee/en/project.html (cit. on p. 69).

Ulbig, Andreas, Theodor S. Borsche, and Göran Andersson (2014). "Impact of Low Rotational Inertia on Power System Stability and Operation". In: *IFAC Proceedings Volumes* 47.3. 19th IFAC World Congress, pp. 7290 –7297 (cit. on p. 15).

Valdivia-Guerrero, Virgilio et al. (Sept. 2016). "Modelling and Simulation Tools for Systems Integration on Aircraft". In: *SAE Technical Paper*. SAE International (cit. on p. 69).

Vanfretti, L., T. Bogodorova, and M. Baudette (2014). "A Modelica power system component library for model validation and parameter identification". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 67).

Vanfretti, L. et al. (2016a). "iTesla Power Systems Library (iPSL): A Modelica library for phasor time-domain simulations". In: *SoftwareX* 5, pp. 84–88 (cit. on p. 77).

Vanfretti, Luigi et al. (2016b). "RaPId: A modular and extensible toolbox for parameter estimation of Modelica and FMI compliant models". In: *SoftwareX* 5, pp. 144 –149 (cit. on pp. 67, 72).

Villella, F. et al. (Oct. 2012). "PEGASE pan-European test-beds for testing of algorithms on very large scale power systems". In: *2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe)*. Berlin, Germany (cit. on p. 68).

Šilar, Jan, Filip Ježek, and JiřÃ Kofránek (2018). "PDEModelica: a Modelica language extension for partial differential equations implemented in OpenModelica". In: *International Journal of Modelling and Simulation* 38.2, pp. 128–137 (cit. on p. 98).

Waurich, V. and J. Weber (2017). "Interactive FMU-based visualization for an early design experience". In: *Modelica'2017: The 12th International Modelica Conference*. Prague, Czech Republic (cit. on p. 56).

Webster, J. and C. Bode (Mar. 2019). "Implementation of a Non-Discretized Multiphysics PEM Electrolyzer Model in Modelica". In: *Modelica'2019: The 13th International Modelica Conference*. Regensburg, Germany (cit. on p. 78).

Wetter, M. and C. Haugstetter (2006). "Modelica versus TRNSYS – A Comparison Between an Equation-Based and a Procedural Modeling Language for Building Energy Simulation". In: *The 2nd SimBuild Conference*. Cambridge, MA, USA (cit. on p. 30).

Wetter, M. et al. (Sept. 2019). "IBPSA Project 1: BIM/GIS and Modelica framework for building and community energy system design and operation - Ongoing developments, lessons learned and challenges". English. In: *IOP Conference Series / Earth and Environmental Science* 323.1 (cit. on p. 69).

Wetter, Michael and et al. (Dec. 2015). "IEA EBC Annex 60 Modelica Library – An international collaboration to develop a free open-source model library for buildings and community energy systems". In: *14th IBPSA Conference*. Hyderabad, India (cit. on p. 81).

Wetter, Michael et al. (2014). "Modelica Buildings library". In: *Journal of Building Performance Simulation* 7.4, pp. 253–270 (cit. on p. 80).

Widl, Edmund et al. (2013). "The FMI++ Library: A High-level Utility Package for FMI for Model Exchange". In: *The IEEE Workshop on Modeling and Simulation of Cyber-Physical Energy Systems*. Berkeley, USA (cit. on p. 16).

Winkler, D. (Sept. 2017). "Electrical Power System Modelling in Modelica - Comparing Open-source Library Options". In: *SIMS2017: The 58th Conference on Simulation and Modelling*. Reykjavik, Iceland (cit. on p. 79).

Yang, Tao, Serhiy Bozhko, and Greg Asher (Apr. 2014). "Modelling of Electrical Power Systems with Dynamic Phasorsin Modelica". In: *Modelica'2014: The 10th International Modelica Conference*. Lund, Sweden (cit. on p. 85).

Zabala, Laura et al. (2020). "Virtual testbed for model predictive control development in district cooling systems". In: *Renewable and Sustainable Energy Reviews* 129, p. 109920 (cit. on p. 81).

Zeigler, Bernard P. and J. S. Lee (1998). "Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment". In: *Enabling Technology for Simulation Science II*. Ed. by Alex F. Sisti. Vol. 3369. International Society for Optics and Photonics. SPIE, pp. 49 –58 (cit. on p. 93).

Ziegler, B. P. (1976). *Theory of Modeling and Simulation*. New York, USA: John Wiley & Sons (cit. on p. 84).

Zimmer, Dirk (2013). "A new framework for the simulation of equation-based models with variable structure". In: *SIMULATION* 89.8, pp. 935–963 (cit. on p. 98).

Zimmer, Dirk (June 2016). "Equation-based Modeling with Modelica – Principles and Future Challenges". In: *Simulation Notes Europe* 26.2, pp. 67–74 (cit. on p. 98).