Equation-Based Modeling with Modelica – Principles and Future Challenges

Dirk Zimmer

Inst. of System Dynamics and Control, German Aerospace Center DLR, Münchener Straße 20, 82234 Weßling, Germany; *dirk.zimmer@dlr.de*

Simulation Notes Europe SNE 26(2), 2016, 67 - 74
DOI: 10.11128/sne.26.on.10332
Received: June 20, 2016 (Invited Overview Note);
Accepted: June 25, 2016;

Abstract. Modelica is a well-established, open standard for the modeling and simulation of cyber-physical systems. Since it is based on equations, this modeling language is applicable to a multitude of physical domains and especially suited for complex physical systems and their control. This paper provides a brief introduction on the kind of equation-based modelling promoted by Modelica and its underlying core principles. The paper then describes its current state in development and outlines the most important technology trends for its future development.

Introduction

Modelling and simulation is today one of the most prevalent methods for the design of systems and their control. A large variety of specialized tools have been developed and are continually improved that take into account the specifics of each physical domain.

However, many systems combine components of different physical domains. Their design consequently represents an optimization process that cannot be mastered by any domain-specific tool alone. Fortunately, many methods for modelling and simulation of physical systems build on the same principals and can be shared across different physical domains. This is what has led to a multitude of generic simulation languages for physical systems such as MIMIC, ACSL, CSMP, gPROMS, VHDL-AMS, Matlab-Simulink, etc. [16].

This paper presents one of the more recent and meanwhile well-established languages: Modelica. This is an openly standardized modelling language, primarily aimed at the modelling and simulation of physical systems and their control.

From its founding in 1997, the language developed with a steadily growing user-base both in academia and industry.

Being a discussion paper, this text presents the author's view on Modelica:

- How to introduce Modelica with its basic principles?
- How has Modelica matured and established itself?
- What will be the future challenges and main development trends?

Each of these questions is addressed in a separate section. Going through these questions aims at providing a concise overview on Modelica.

1 Basic Principles of Modelica

There are five core principles that define the design of the Modelica modelling language:

- It is an **equation-based** language.
- It enables the **acausal** formulation of systems.
- It uses **physical connectors** to connect different components of a model.
- It is an **object-oriented** language that enables the reuse of once developed models.
- Although being a textual language, it embraces a second layer of **graphical modeling**.

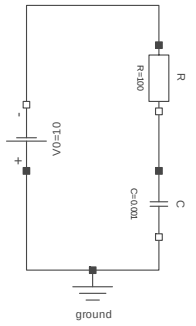
This list represents the author's choice. There are many other factors that have influenced Modelica and that need to be taken into account when designing a language. Nevertheless, these 5 principles cover the most vital aspects. Let us go through them one by one.

1.1 Equation-based Modeling

As an equation-based language, Modelica enables the modeller to formulate the system directly by the means of differential algebraic equations (DAEs). The following Listing represents the equations of a simple RC circuit in a corresponding Modelica model.

A Modelica model has a header that contains declarations of parameters (constant over simulation time) and variables. The subsequent equation part then contains algebraic and differential equations. The operator `der()` represents the time derivative:

```
model SimpleCircuit
  parameter Real C;
  parameter Real R;
  parameter Real V0;
  Real i;
  Real uC;
equations
  V0-uC = R*i;
  der(uC)*C = i;
end SimpleCircuit;
```



Listing 1: A simple Modelica model for an RC circuit.

Listing 1 presents basic elements of a Modelica model: parameter, variables, and equations. None of these elements is bound to physics in any way. Yet, it is meaningful, to use variables of physical quantities where applicable and to add description texts. This makes the code far easier to understand and safer to use:

```
model SimpleCircuit
 "A simple RC circuit"
 import SI = Modelica.SIunits;
 parameter SI.Capacitance C=0.001
 "Capacity";
 parameter SI.Resistance R = 100
 "Resistance";
 parameter SI.Voltage V0 = 10
 "Source Voltage";
 SI.Current i "Current" ;
 SI.Voltage uC "Capacitor Voltage";

initial equation
  uC = 0;
equations
  V0-uC = R*i;
  der(uC)*C = i;
end SimpleCircuit;
```

Listing 2: Polished version of listing 1, using description texts and physical units.

The equations in the examples of this paper are only used to describe continuous processes but Modelica also contains means to deal with events and conditional expressions which enable the formulation of discrete processes.

1.2 Acausality

Modelica uses acausal equations and not causal assignments. This means that the modeller can focus on what he wants to model and does not need to state how to compute the system. For instance, Ohm's law of Listing 1 can also be formulated in either of the following forms:

- `uC + R*i = V0;`
- `(uC-V0)/R = -i;`

Acausality is however more than the freedom on how to form an equation. It becomes an essential feature as soon as equations are reused, as typical for object-oriented modelling. Let us consider the following electric circuit that contains two instances of Ohm's law (Figure 1)

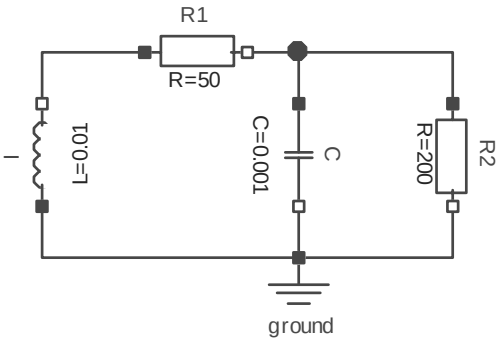


Figure 1: An electric circuit with two resistors R1 and R2 both representing Ohm's law.

and its corresponding computational realization in Matlab Simulink® (Figure 2):

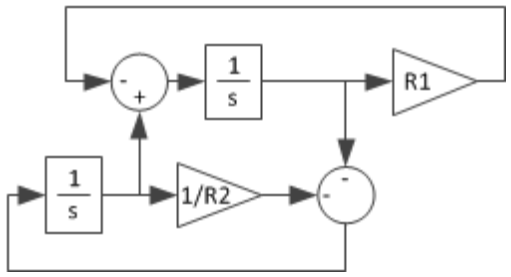


Figure 2: Computational realization of the electric circuit in a Simulink Block Diagram.

Evidently, R1 is used to compute the voltage out of the current, while R2 is used to compute the current out of the voltage. In Modelica, you do not have to care about this. Ohm's law is valid for both resistors.

More formally, within Modelica you describe systems according to the implicit DAE form:

$$\mathbf{0} = \mathrm{F}(\boldsymbol{x_p}, \dot{\boldsymbol{x}}_{\boldsymbol{p}}, \boldsymbol{u}, t)$$

It is then the task for a Modelica tool to bring this implicit DAE form into an explicit ODE form, typically more suitable for simulation:

$$\dot{\boldsymbol{x}} = \mathrm{f}(\boldsymbol{x}, \boldsymbol{u}, t)$$

where $\boldsymbol{x}$ is a subset of $\boldsymbol{x_p}$. This transformation is called index-reduction [3], with the index denoting the complexity of the transformation. Many physical systems, such as multibody systems typically are higher-index systems.

Index reduction is also useful for more advanced applications. It enables model inversion: instead of prescribing the forces and computing the trajectory, the modeller can prescribe the trajectory and compute the required forces. Such inverted models can then be used in a non-linear control loop to derive modern model-based control laws [12].

1.3 Physical connectors

The example of listing 2 contains a complete model, with as many equations as variables. This approach however is only feasible for very small models. For larger models with thousands of equations, an object-oriented approach is mandatory. Here, a model is composed out of sub-models, also denoted as components. The sub-models contain fewer equations than variables. The missing equations then are added by connecting the components. For example, Figure 3 displays the model diagram of an electrical actuated inverted pendulum.

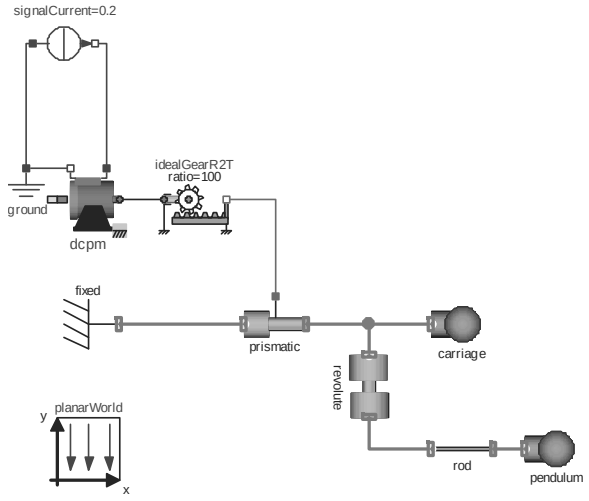


Figure 3: Modelica model diagram of an electric driven inverse pendulum.

The individual components of this diagram feature domain-specific connectors (green squares for translatory mechanical hinges, blue squares for electric pins, etc.). These connectors are declaring pairs of potential and flow variables. By connecting them with lines, a junction is formed. For each of these junctions, equations are generated: potential variables are all set to be equal whereas the sum of flow variables has to be zero.

The modeller is free to use whatever variables for potential and flow as desired. For many physical domains however, Table 1 already provides suitable pairings:

Domain	Potential	Flow
Translational Mechanics	Velocity: *v* [m/s]	Force: *f* [N]
Rotational Mechanics	Angular velocity: ω [1/s]	Torque: τ [Nm]
Electrics	Voltage potential v [V]	Current i [A]
Magnetics	Magnetomotive force: Θ [A]	Time-derivative of magnetic flux: Φ [V]
Hydraulics	Pressure p [Pa]	Volume flow rate V [m^3/s]
Thermal	Temperature T[K]	Entropy flow rate S [J/Ks]
Chemical	Chem. potential: μ [J/mol]	Molar flow rate *v* [mol/s]

Table 1: Pairs of potential and flow variables for different physical domains inherited from bond graphs

These pairings of potential and flow variables are known from bond graph modelling [7]. The product of each this pairs represents the flow of energy. Hence the connection via these pairings will represent flows of energy going in and out of components.

The typical use in Modelica may partly deviate from this table. For instance position may be favoured over velocity, and specific enthalpy maybe more practical than temperature. Modelica is hence less dogmatic than bond graphs but nevertheless still profits from the same underlying thermodynamic principles.

1.4 Object Orientation

The combination of physical connectors with pairs of potential and flow and the ability to formulate acausal DAEs then enables a fully object-oriented modelling approach.

The equations are distributed over several compo-

nents. Components of any domain such as resistors, dampers, wheels, joints, batteries can be declared in the same way as simple variables. Listing 3 presents the object-oriented code corresponding for the following electric circuit.

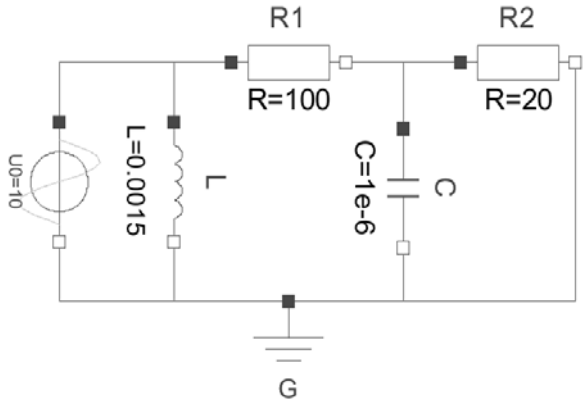


Figure 4: An example electric circuit.

```
model Circuit
  import E = Modelica.Electrical.Analog
  E.Basic.Resistor R1(R=100);
  E.Basic.Resistor R2(R=20);
  E.Basic.Capacitor C(C=1e-6);
  E.Basic.Inductor L(L=0.0015);
  E.Sources.SineVSource S(Ampl=15, Freq=50);
  E.Basic.Ground G;
equations
  connect(G.p,S.n)
  connect(G.p,L.n)
  connect(G.p,R2.n)
  connect(G.p,C.n)
  connect(S.p,R1.p)
  connect(S.p,L.p)
  connect(R1.n,R2.p)
  connect(R1.n,C.p)
end Circuit;
```

Listing 3: Modelica Model of Figure 4.

Models for one domain can be collected in packages, Modelica's name for its software libraries. The package for analogue electrical components is imported in the listed example. The components of this package are then accessed by dot notation and declared just as variables. The equation section does not contain direct equations anymore but just the connect statements.

There is more to object-orientation than the basic use of components and its collection in packages. Modelica supports concepts of inheritance, even multiple inheritance. Partial models are the counterpart to abstract classes in equation-based models and can be used to define component interfaces.

The structural type system then enables a flexible replacement of models or model classes.

1.5 Graphical modelling

The manual coding of physical systems as presented in Listing 3 is a laborious and potentially error-prone task. Instead, engineers prefer to model graphically. Most Modelica modelling tools hence offer a diagram editor that can be used to compose systems such as in Figure 1, 2, or 4 in a purely graphical way by using drag and drop.

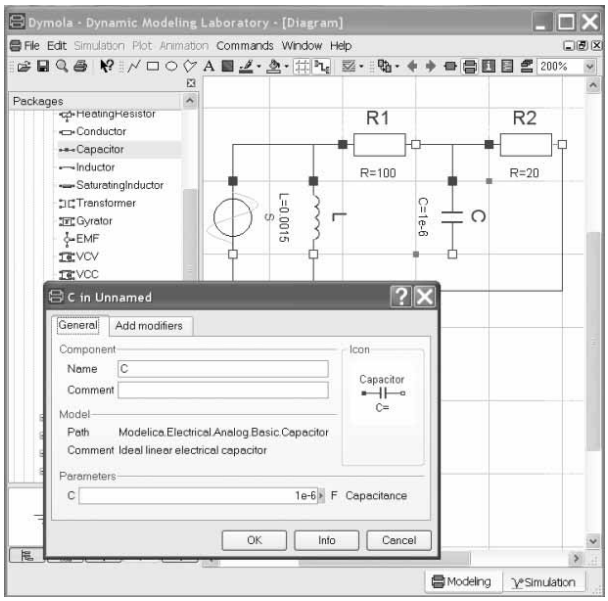


Figure 5: GUI of Dymola, one possible Modelica tool, used to model the circuit of Figure 4.

The Modelica language provides annotations that act as a container for the resulting meta-information. These are used to store the graphical information about the position, orientation and scale of the components in the diagram layer. Most Modelica editors hide the content of annotations by default so that the modeller can focus on the essential parts.

1.6 Resulting modelling style

What results of the 5 principles is a declarative modelling language that enables the creation of self-contained models.

Declarative means that the modeller can focus on *what* he wants to model rather on *how* to compute it.

Self-contained means that the models alone are valuable information source, even without any simulation tool at hand.

2 Establishment of Modelica

2.1 Language and tools

Modelica is not the first language to be based on the outlined principles. Many academic predecessors or tools such as OMOLA, or 20-sim have helped to path the way. Yet it is one of the few openly specified languages that meanwhile found significant industry acceptance and tool support. Table 2 provides an incomplete list of commercial and free tools supporting the Modelica standard.

Tool	Developer	Type
Dymola	Dassault Systèmes	commercial
OpenModelica	OSMC /Linköping Univ.	free
SystemModeller	Wolfram	commercial
JModelica.org	Modelon AB / Lund Univ.	free
SimulationX	ITI GmbH	commercial
MapleSim	MapleSoft	commercial
LMS Imagine Lab Amesim	Siemens PLM	commercial
MWorks	Suzhou Tongyuan	commercial
CyModelica	CyDesign Labs /ESI	commercial
Modelicac	SciLab Enterprises	free

Table 2: List of Modelica simulation environments. Complete tool list available at [1].

Whereas Listings 1-3 only presented toy examples, many realistic models of various application fields have been created, often with more than 100,000 equations.

Automotive companies were among the early adopters. Models for vehicle dynamics but also for cabin climatization and powertrain modelling are in use at the automotive industry. Also motorcycles, trucks, trains and heavy equipment of all kinds are frequently modelled in Modelica.

Meanwhile also the conservative aviation business is increasingly using Modelica. Especially the design of energy systems for modern more electric aircraft is a demanding application field.

The energy sector in general is highly relevant for Modelica. Models of various power plants (from solar thermal to coal fired) have been created and their integration into a common energy grid is studied. In this way, a substantial amount of intellectual property has meanwhile been encoded in Modelica.

It would, however, be wrong to reduce Modelica just to the language and its tools. Equally important are the available Modelica libraries, especially the extensive, free Modelica Standard Library (MSL). Furthermore there is the Modelica Association. This is a non-profit association that engages in development of the standard, corresponding libraries and the scientific and industrial community.

The Modelica language, the Modelica libraries and the Modelica Association consequently form a powerful triangle that has enabled the recent success of this technology.

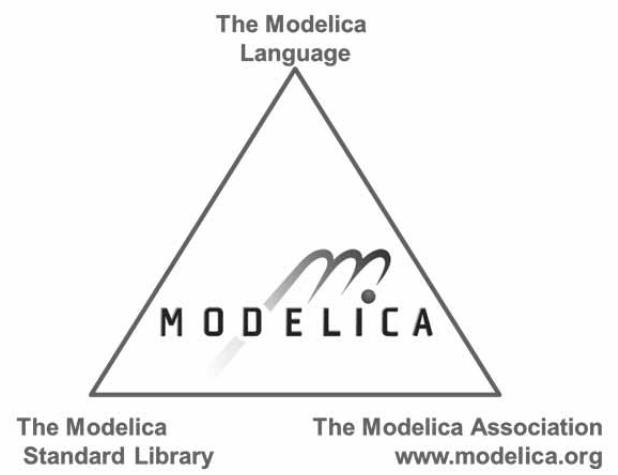


Figure 6: Illustration of Modelica's version of trinity: combining an open language, with open libraries and an open association.

2.2 Modelica Standard Library

Most modelling tasks do not start from scratch but build upon pre-existing models. The Modelica Standard Library provides therefore suitable building blocks. For the most relevant domains in physics and control it offers ready-to-use components, corresponding documentation and explanatory examples.

The recent development of the MSL has undergone steady growth. So has the number of code lines doubled to more than 250,000 from MSL v2.2.2 (2008) to MSL v3.2.1 (2013) (including comments and meta-data).

Where the MSL proves to be insufficient, the modeller can choose from a long list of free and or commercial Modelica libraries. The Modelica website [1] lists all these libraries together and offers compliance checkers. For the collaborative development of free libraries, GitHub offers a popular and well suited platform.

2.3 Modelica Association

The Modelica Association is a non-profit organization formed out of more than 20 organizational and more than 100 individual members. This community organizes its work in internal projects. Two of them are devoted for refinement and development of the language specification and for the development of the MSL.

Since both the specification as well as the MSL have meanwhile reached a high level of complexity, further development or request for changes or clarification build upon dedicated processes. To illustrate this, Figure 7 shows the size of the specification document (in terms of page number). From roughly 50 pages the size almost 6-folded to 300. The specification of Modelica is in plain text and in most parts not formal. Hence, the rise of specification length does not only express the growing complexity of the language but also the stronger need for clarifications. The larger number of tools supporting Modelica fortifies this need.

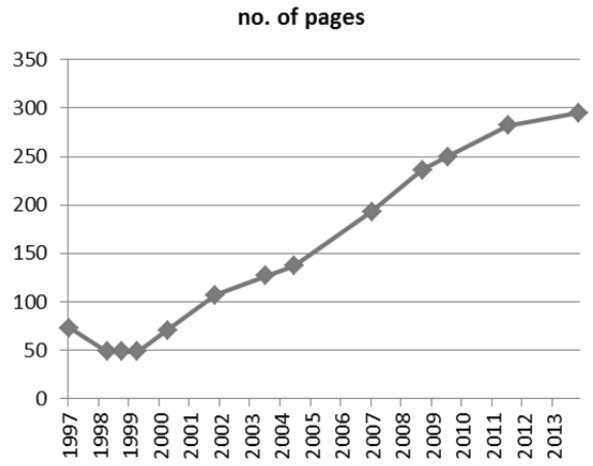


Figure 7: Length of the Modelica Specification Document in number of pages from 1997 (v1.0) to 2014 (v3.3rev1).

Furthermore, the Modelica Association is also organizing the development of the FMI Standard which goes beyond Modelica in its applicability (see chapter 3).

Internal meetings are organized in form of regular design meetings, roughly 4 times a year. To reach out to a larger community, international Modelica conferences are organized every one or two years. These conferences bring together industry and academia. Their scope ranges from concrete modelling applications to new language concepts. For newbies to Modelica, these conferences are an excellent learning and networking opportunity.

3 Modelica – Future Challenges

Being a naturally readable and openly standardized language, Modelica has established itself as an excellent storage format for mathematical models. This alone is of major importance. Model libraries often contain the result from years of development, validated data from expensive test rigs and in general models often represent key intellectual property of industrial companies. Hence, it is vital that the format of these models is a tool-independent and mature standard that guarantees ongoing usability.

This usability of a system dynamics model is also what generates the upcoming demands on the Modelica language and its tools. The primary and established application fields are the early design optimization of systems and the corresponding design of controllers.

These two fields form the seeds for two corresponding development trends in today's industry. The first trend is the increasing use of models within systems engineering also frequently denoted as model-based systems engineering (MBSE). The second trend of cyber-physical systems is where controllers and the physical system are modelled as a whole and the models are used more directly for the controller development.

Figure 8 provides an overview of typical tasks arising from a stronger integration of Modelica in either systems engineering or cyber-physical systems.

3.1 Towards MBSE

In system engineering, the use of Modelica and its models is not an isolated activity but part of a larger product development process. The trend is to use more and more models for this. This confronts Modelica with new demands for the use of its models such as the formulation of requirements or the need for failure analysis.

Additionally, the term systems engineering correlates often with the on-going bureaucratization of engineering. The trend is hence driven by large industrial companies, often part of even larger conglomerates in the need to cooperate with each other. Since these large entities, are strongly bureaucratic [6], so their engineering processes become. For Modelica, this means that generic interfaces are needed to integrate the models or the tools within the foreseen (if not prescribed) industrial tool-chains. The key development in this direction was the development of the functional mock-up interface (FMI) standard [2].

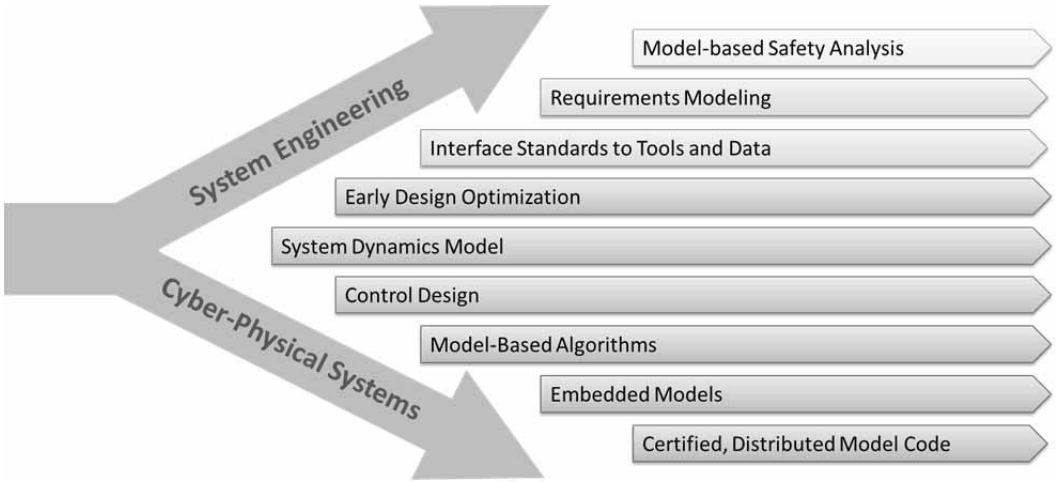


Figure 8: Illustration of two major development trends and their sub-topics.

FMI offers a tool independent standard for model exchange and co-simulation. The difference between these two forms is whether or not the code of the numerical ODE solver is included or not. Model-exchange is (simplistically) based on the mathematical form:

$$(\dot{\boldsymbol{x}}, \widehat{\boldsymbol{h}}, \boldsymbol{y}) \;=\; \mathrm{f}(\boldsymbol{x}, \boldsymbol{h}, \boldsymbol{u}, t)$$

with $\boldsymbol{x}$ being the continuous states, $\boldsymbol{h}$ the discrete states, $\boldsymbol{u}$ the input, $\boldsymbol{y}$ the output and t the time. The FMI then offers a suitable application programming interface that enables the connection to other models and the application of any hybrid ODE solver. In co-simulation, the numerical solver for the advance of time is already included in the exchanged code. This is suitable for non-stiff couplings between sub-systems and is also an option to combine classic tools for system simulation with 3D tools for fluid dynamics or finite elements.

One important aspect of FMI is that models do not need to be exchanged as white boxes. The model code can be obfuscated either by compilation or even by more effective means. In many cases, this represents a sufficient level of protection of intellectual property that companies are willing to mutually exchange some of their models, a process needed for the early design of today's systems. For instance, within the research project Clean Sky, the environmental control system model of an aircraft, the corresponding cabin model and the electrical system model could be exchanged by FMI and a total system simulation was performed [14].

The exchange of models is also important for other reasons. Models can also represent requirements.

In this way a company can communicate its specification to a supplier and the supplier can test his models against these specifications.

Ready-to-use libraries [8] help the Modelica developer to formulate its requirements and future language extensions [4] may ease the binding of requirements to the corresponding models in the near future.

The FMI does not only offer a standardized API, it also contains a standardized XML format for the description of hierarchical, object-oriented models. This format can be used to import or export meta-information for the corresponding models. Enhancements of the Modelica standard enable to include this or other meta-information directly within Modelica models [15]. In combination this allows advanced model-based methods to be performed using multiple tools.

For instance, models can be tagged with possible fault modes and corresponding failure rates. A tool can then extract this information and perform a series of simulation for a safety and reliability analysis. By extracting information about the connection structure of the model, this analysis can cover all relevant fault cases within reasonable effort [10]. Special Modelica libraries for fault modelling may help the modeller perform such a task [13].

In summary, the integration of Modelica into the processes of MBSE is an on-going process. However, the formulation of interface standards such as FMI has led to significant higher industry acceptance. New, practically explored language concepts support this development by enabling a better handling of meta-data within models.

3.2 Towards Cyber-Physical Systems

Cyber-Physical systems denote mechanisms in interaction with model-based algorithms. The goal is hence to develop model-based algorithms such as controllers, health-monitoring, fault-detection, etc. within Modelica, test these algorithms (also in discretized form) in a virtual environment with Modelica models. Ideally, code for the distributed embedded control units shall then be automatically generated in a certified form similar to standards such as DO-178. In its entirety this represents hence a challenging goal.

In order to better support this trend, Modelica 3.3 has been extended by Modelica Synchronous [5]. This language extension enables the modelling of clocked synchronous processes. In this way, controllers can be modelled in a discrete form and discrete control effects can properly be taken into account.

To generate code for the embedded control units, code generation of Modelica tools may require improvement. Also here the FMI standard may be useful to serve as a container for light-weight model and simulation code. The use of FMIs on rapid-prototyping hardware has meanwhile substantially improved [2].

Finally, for many safety critical applications in aviation, transport or energy, the certification of the applied controller code is of key importance. This will only be realistically possible with a well-defined subset of the Modelica language. First definitions in these directions have been undertaken by [9] and [11]. Nevertheless, the vision to automatically generate certified code for embedded systems out of Modelica models is still a far reaching goal but definitely worth pursuing.

4 Conclusions

Being solely based on equations with no pre-implemented physics, Modelica is a truly generic and universal modelling standard. Much freedom is given to the modeller and after almost 20 years of establishment, it is fair to say that modellers of many different backgrounds have endorsed this freedom. For the future development of Modelica, the resulting amount of variety in its usage and the rising complexity represents a vital challenge – a challenge worth to be taken.

About the author

Dr. Dirk Zimmer received his PhD from ETH Zurich in 2010. He is now research group leader at the German Aerospace Center DLR for the modelling and simulation of aircraft energy systems.

He is using Modelica since 2005 and is teaching as guest lecturer at the TU Munich since 2010. Also since then, he is regular member of the Modelica Association.

References

[1] www.modelica.org

[2] www.fmi-standard.org/literature

[3] Bujakiewicz P. *Maximum weighted matching for high index differential-algebraic equations*. PhD Thesis, Technische Universiteit Delft, 1993,147p.

[4] Elmqvist H, Olsson H, Otter M. Constructs for Meta Properties Modeling in Modelica. In *Proc. of the 11th Intern. Modelica Conf*; 2015 Sep, Paris, France.

[5] Elmqvist H, Otter M, Mattsson SE. Fundamentals of Synchronous Control in Modelica. *Proc. of the 9th Intern. Modelica Conf*; 2012 Sep, Munich, Germany.

[6] Graeber D. *The Utopia of Rules*, Melville House, 2015

[7] Karnopp DC, Margolis DL, Rosenberg RC. *System Dynamics: Modeling and Simulation of Mechatronic Systems*. 4th edition, John Wiley&Sons, 2006, New York, 576p.

[8] Kuhn M, Giese T, Otter M. Model Based Specifications in Aircraft Systems Design. In *Proc. of the 11th Intern. Modelica Conf*; 2015 Sep, Paris, France.

[9] Satabin L, Colaço JL, Andrieu O, Pagano B. Towards a Formalized Modelica Subset. *Proc. of the 11th Intern. Modelica Conf*; 2015 Sep, Paris, France.

[10] Schallert C. Automated Safety Analysis by Minimal Path Set Detection for Multi-Domain Object-Oriented Models. *Proc. of the 11th Intern. Modelica Conf*; 2015 Sep, Paris, France.

[11] Thiele B, Knoll A, Fritzson P. Towards Qualifiable Code Generation from a Clocked Synchronous Subset of Modelica. *Modeling, Identification and Control*; 2015 36(1):23-52

[12] Thümmel M, Looye G, Kurze, Otter M, Bals J. Nonlinear Inverse Models for Control In: *Proc. of the 4th Intern. Modelica Conf*, 2005, Hamburg, Germany

[13] van der Linden F. General fault triggering architecture to trigger model faults in Modelica using a standardized blockset. *Proc. of the 10th Intern. Modelica Conf*; 2014 Mar, Lund, Sweden.

[14] Zimmer D, Giese T, Crespo M, Vial S. Model Exchange in Industrial Practice, *Geener Aviation 2014;* 2014 Brussels, Belgium

[15] Zimmer D, Otter M, Elmqvist H, Kurzbach G. Custom Annotations: Handling Meta-Information . *Proc. of the 10th Intern. Modelica Conf*; 2014, Mar, Sweden.

[16] Zimmer D. (2010), *Equation-Based Modeling of Variable Structure Systems;* PhD Dissertation, 2010, ETH Zürich, 219 p.