M

Multi-domain Modeling and Simulation

Martin Otter
Institute of System Dynamics and Control, German Aerospace Center (DLR), Wessling, Germany

Abstract

One starting point for the analysis and design of a control system is the block diagram representation of a plant. Since it is nontrivial to convert a physical model of a plant into a block diagram, this can be performed manually only for small plant models. Based on research from the last 40 years, more and more mature tools are available to achieve this transformation fully automatically. As a result, multi-domain plants, for example, systems with electrical, mechanical, thermal, and fluid parts, can be modeled in a unified way and can be used directly as input–output blocks for control system design. An overview of the basic principles of this approach is given, and it is shown how to utilize nonlinear, multi-domain plant models directly in a controller. Finally, the low-level "Functional Mockup Interface" standard is sketched to exchange multi-domain models between many different modeling and simulation environments.

Keywords

Block diagram · Differential-algebraic equation (DAE) system · Flow variable · Functional Mockup Interface (FMI) · Inverse models · Modelica · Modia · Object-oriented modeling · Potential variable · Stream variable · Symbolic transformation · VHDL-AMS

Introduction

Methods and tools for control system analysis and design usually require an input–output block diagram description of the plant to be controlled. Apart from small systems, it is nontrivial to derive such models from first principles of physics. Since a long time, methods and tools are available to construct such models automatically for one domain, for example, a mechanical model, an electronic, or a hydraulic circuit. These domain-specific methods and tools are, however, only of limited use for the modeling of multi-domain systems.

In the dissertation (Elmqvist 1978), a suitable approach for multi-domain, object-oriented modeling has been developed by introducing a modeling language to define models on a high level based on first principles. The resulting differential-algebraic equation (DAE) systems are automatically transformed with proper algorithms in a block diagram description with input and output signals based on ordinary differential equations (ODEs), where the algebraic variables

J. Baillieul, T. Samad (eds.), *Encyclopedia of Systems and Control*,
https://doi.org/10.1007/978-1-4471-5102-9_140-2

have been eliminated and all derivatives are explicitly solved for.

In 1978, computers were not powerful enough to apply this method on systems with more as a few hundred equations. In the 1990s the technology has been substantially improved, many different modeling languages appeared (and also disappeared), and object-oriented modeling was introduced in commercial simulation environments.

In Table 1, an overview of the most important standards, languages, and tools in the year 2019 for multi-domain modeling is given:

- The *Modelica* language is a standard from the Modelica Association (2017). The first version was released in 1997. The language is accompanied with the *Modelica Standard Library* (https://github.com/modelica/ModelicaStandardLibrary) – an open-source Modelica library with about 1600 model components and 1350 functions from many domains. There are several free and commercial software tools supporting the Modelica language and the Modelica Standard Library.
- The *VHDL-AMS* language is a standard from IEEE (IEEE 1076.1-2017 2017), first released in 1999. It is an extension of the widely used VHDL hardware description language. This language is especially used in the electronics community.
- There are several (proprietary) vendor-specific modeling languages, notably *EcosimPro* and

Multi-domain Modeling and Simulation, Table 1 Multi-domain modeling and simulation environments

Environments supporting the ***Modelica*®** ***Language Standard*** (https://www.modelica.org/)	
Altair Activate®	https://solidthinking.com/product/activate
ANSYS® Twin Builder	https://www.ansys.com/products/systems
Dymola®	https://www.dymola.com/
JModelica.org	https://jmodelica.org/
MapleSim	https://www.maplesoft.com/products/maplesim/
MWorks	http://en.tongyuan.cc/
OpenModelica	https://www.openmodelica.org/
OPTIMICA® Compiler Toolkit	https://www.modelon.com/
Simcenter® Amesim	https://www.plm.automation.siemens.com
SimulationX®	https://www.simulationx.com/
Wolfram SystemModeler	https://www.wolfram.com/system-modeler/
Environments supporting the ***VHDL-AMS Standard*** (https://standards.ieee.org)	
AMS Designer	https://www.cadence.com/
ANSYS® Twin Builder, Simplorer®	https://www.ansys.com/products/systems
Saber	https://www.synopsys.com
SMASH®	https://www.dolphin-integration.com
SystemVision®	https://www.mentor.com/products/sm/
Environments supporting ***vendor-specific*** *multi-domain modeling languages*	
EcosimPro®	https://www.ecosimpro.com/
gPROMS	https://www.psenterprise.com/products/gproms
Simscape	https://www.mathworks.com/products/simscape
20-sim	https://www.20sim.com/
Open-source *environments supporting other multi-domain modeling languages*	
Modia	https://github.com/ModiaSim

gRPOMS for modeling of thermodynamic processes, *Simscape* from MathWorks as an extension to Simulink®, *MAST* the underlying modeling language of *Saber* (Mantooth and Vlach 1992), and the language of *20-sim*, which supports *bond graphs* (Karnopp et al. 2012). Bond graphs are a special graphical notation to define multi-domain systems based on energy flows. It was invented in 1959 by Henry M. Paynter.

- *Modia* and its supporting packages form an open-source prototyping platform based on the *Julia* programming language (Bezanson et al. 2017). Modia is the name of a Julia package that supports a modeling language called "Modia language" as a domain-specific extension of Julia and provides algorithms to symbolically transform Modia models into DAE systems that can be numerically solved by standard DAE integrators. The Modia language and the new algorithms used in the Modia package are intended as inspiration for the development of the next Modelica generation.

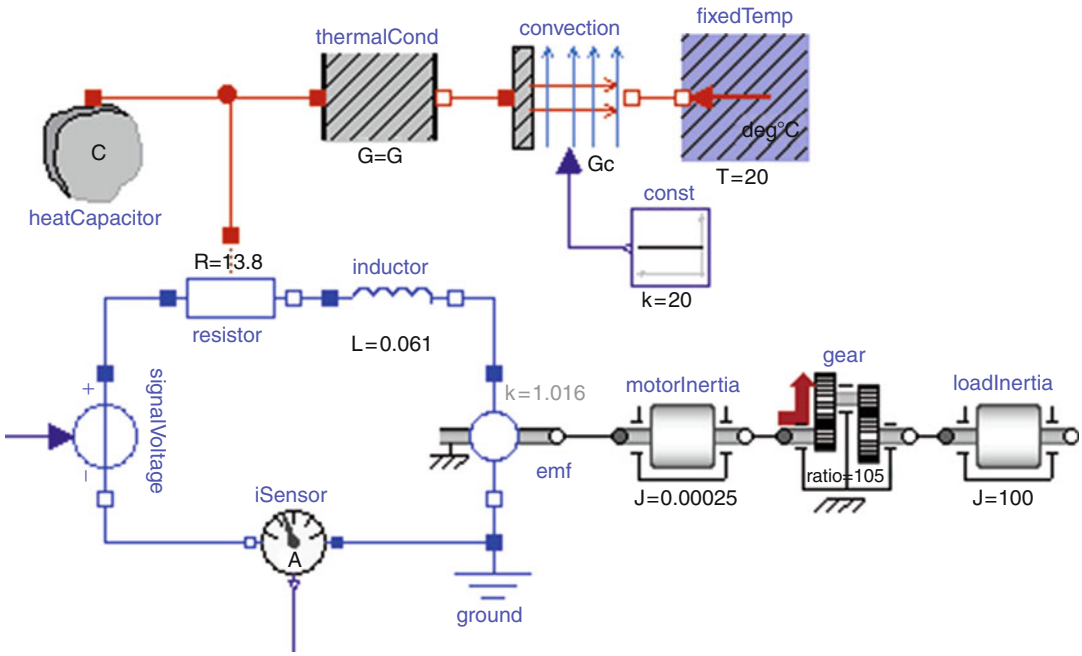
In section "Modeling Language Principles", the principles of multi-domain modeling based on a modeling language are summarized. In section "Models for Control Systems", it is shown how such models can be used not only for simulation but also as components in nonlinear control systems. Finally, in section "The Functional Mockup Interface", an overview about a low-level standard for the exchange of multi-domain systems is described.

Modeling Language Principles

Schematics: The Graphical View

Modelers nowadays require a simple to use graphical environment to build up models. With very few exceptions, multi-domain environments define models by schematic diagrams. A typical example is given in Fig. 1, showing a simple direct current electrical motor in Modelica.

In the lower left part, the electrical circuit diagram of the DC motor is visible, consisting mainly of the armature resistance and inductance



Multi-domain Modeling and Simulation, Fig. 1 Modelica schematic of DC motor with mechanical load and heat losses

of the motor, a voltage source, and component "emf" to model in an idealized way the electromotoric forces in the air gap. On the lower right part, the motor inertia, a gear box, and a load inertia are present. In the upper part, the heat transfer of the resistor losses to the environment is modeled with lumped elements.

A component, like a resistor, rotational inertia, or convective heat transfer, is shown as an icon in the diagram. On the border of a component, small rectangular or circular signs are present representing the "physical ports." Ports are connected by lines and model the (idealized) physical or signal interaction between ports of different components, for example, the flow of electrical current or heat or the rigid mechanical coupling. Components are built up hierarchically from other components. On the lowest level, components are described textually with the respective modeling language (see section "Component Equations").

Coupling Components by Ports

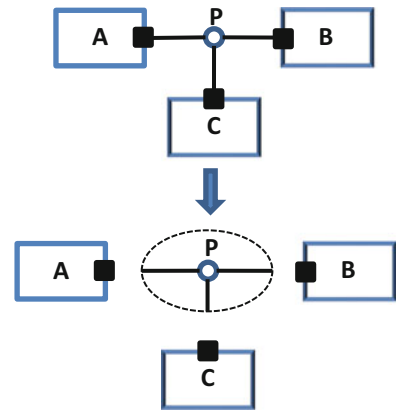
The ports define how a component can interact with other components. A port contains a definition of the variables that describe the interface and defines in which way a tool can automatically construct the equations for the connections. A typical scenario is shown in Fig. 2 where the ports of the three components A, B, C are connected together at one point P.

When cutting away the connection lines, the resulting system consists of three decoupled components A, B, C and a new component around P describing the infinitesimally small connection point. The balance equations and the boundary conditions of the respective domain must hold at all these components. When drawing the connection lines, enough information must be available in the port definitions so that the tool can construct the equations of the infinitesimally small connection points automatically.

To summarize, the component developer is responsible that the balance equations and boundary conditions are fulfilled for every component (A, B, C in Fig. 2), and the tool is responsible that the balance equations and boundary conditions are also fulfilled at the points where the components are connected together (P in Fig. 2). As a consequence, the balance equations and boundary conditions are fulfilled in the overall model containing all components and all connections.

In order that a tool can automatically construct the equations at a connection point, every port variable needs to be associated to a port variable type. In Table 2, some port variable types of Modelica are shown. Hereby it is assumed that $u_1, u_2, \ldots, u_n, y, v_1, v_2, \ldots, v_n, f_1, f_2, \ldots, f_n, s_1, s_2, \ldots, s_n$ are corresponding port variables from different components that are connected together at the same point P.

Port variable types "input" and "output" define the usual signal connections in block diagrams. "Potential variables" and "flow variables" are used to define standard physical connections. For example, an electrical port contains the electrical potential and the electrical current at the port, and when connecting electrical ports together, the electrical potentials are identical, and the sum of the electrical currents is zero; see Table 2. These equations correspond exactly to Kirchhoff's voltage and current laws. "Stream variables" are used to describe the connection semantics of intensive quantities in bidirectional fluid flow, such as specific enthalpy or mass fraction. Here, the idealized balance equation at a connection point states, for example, that the sum of the port enthalpy flow rates is zero and the port



Multi-domain Modeling and Simulation, Fig. 2 Cutting the connections around the connection point P results in three decoupled components A, B, C and a new component around P describing the infinitesimally small connection point

Multi-domain Modeling and Simulation, Table 2 Some port variable types in Modelica

Port variable type	Connection semantics
Input variables u_i, output variable y	$u_1 = u_2 = \ldots = u_n = y$ (exactly one output variable can be connected to n input variables)
Potential variables v_i	$v_1 = v_2 = \ldots = v_n$
Flow variables f_i	$0 = \sum f_i$
Stream variables s_i (with associated flow variables f_i)	$0 = \sum f_i \hat{s}_i;\ \hat{s}_i = \begin{cases} s_{mix} & \text{if } f_i > 0 \\ s_i & \text{if } f_i \leq 0 \end{cases}$ $(0 = \sum f_i)$

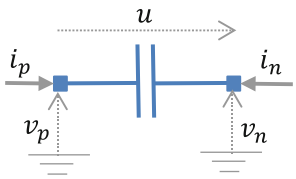
Multi-domain Modeling and Simulation, Table 3 Some port definitions from Modelica

Domain	Port variables
Electrical analog	Electrical potential in [V] *(pot.)* electrical current in [A] (*flow*)
Electrical multiphase	Vector of electrical ports
Electrical quasi-stationary	Complex electrical potential *(pot.)* complex electrical current (*flow*)
Magnetic flux tubes	Magnetic potential in [A] *(pot.)* magnetic flux in [Wb] (*flow*)
Translational (one-dimensional mechanics)	Distance in [m] *(pot.)* cut-force in [N] (*flow*)
Rotational (one-dimensional mechanics)	Absolute angle in [rad] *(pot.)* cut-torque in [Nm] (*flow*)
Two-dimensional mechanics	Position in x-direction in [m] *(pot.)* position in y-direction in [m] *(pot.)* absolute angle in [rad] *(pot.)* cut-force in x-direction in [N] (*flow*) cut-force in y-direction in [N] (*flow*) cut-torque in z-direction in [Nm] (*flow*)
Three-dimensional mechanics	Position vector in [m] *(pot.)* transformation matrix in [**1**] *(pot.)* cut-force vector in [N] (*flow*) cut-torque vector in [Nm] (*flow*)
One-dimensional heat transfer	Temperature in [K] *(pot.)* heat flow rate in [W] (*flow*)
One-dimensional thermo-fluid pipe flow	Pressure in [Pa] *(pot.)* mass flow rate in [kg/s] (*flow*) spec. enthalpy in [J/kg] *(stream)* mass fractions in [**1**] *(stream)*

enthalpy flow rate is computed as the product of the mass flow rate (a flow variable f_i) and the directional specific enthalpy s_i, which is either the (yet unknown) mixing-specific enthalpy s_{mix} when the flow is from the connection point to the port or the specific enthalpy s_i in the port when the flow is from the port to the connection point. More details and explanations are available from Franke et al. (2009). In Table 3 some of the port definitions are shown that are provided by the Modelica Standard Library and the PlanarMechanics library.

Component Equations

Implementing a component in a modeling language means to define the ports of the component and to provide the equations describing the relationships between the port variables. For example, an electrical capacitor with constant capacitance C can be defined by the equations in the right side of Fig. 3.



$$0 = i_p + i_n$$
$$u = v_p - v_n$$
$$C\frac{du}{dt} = i_p$$

Multi-domain Modeling and Simulation, Fig. 3 Equations of a capacitor component

Such a component has two ports, the pins "p" and "n," and the port variables are the electrical currents i_p, i_n flowing into the respective ports and the electrical potentials v_p, v_n at the ports. The first component equation states that if the current i_p at port "p" is positive, then the current i_n at port "n" is negative (therefore, the current flowing into "p" is flowing out of "n"). Furthermore, the two remaining equations state that the derivative of the difference of the port potentials is proportional to the current flowing into port "p."

One important question is how many equations are needed to describe such a component? For an input–output block, the answer is simple: all input variables are known, and for all other variables, one equation per unknown is needed. Counting equations for physical components, such as a capacitor, are more involved: the requirement that *any type of component connections shall always result in identical numbers of unknowns and equations of the overall system* leads to the following counting rule (for a proof, see Olsson et al. 2008):

1. The number of potential and the number of flow variables in a port must be identical.
2. Input variables and variables that appear differentiated are treated as known variables.
3. The number of equations of a component must be equal to the number of unknowns minus the number of flow variables.

In the example of the capacitor, there are five unknowns (i_p, i_n, v_p, v_n, du/dt) and two flow variables (i_p, i_n). Therefore, $5 - 2 = 3$ equations are needed to define the component in Fig. 3.

```
type Voltage = Real (unit="V");
type Current = Real (unit="A");

connector  Pin
        Voltage v;
   flow Current i;
end Pin;

model Capacitor
    parameter Real C(unit="F");
    Pin p,n;
    Voltage u;

equation
    0 = p.i + n.i;
    u = p.v - n.v;
    C*der(u) = p.i;
end Capacitor;
```

Multi-domain Modeling and Simulation, Fig. 4 Modelica model of capacitor component

```
subtype voltage is real;
subtype current is real;
nature electrical is
        voltage across
        current through
        electrical_ref reference;

entity CapacitorInterface IS
   generic(C: real);
   port (terminal p, n: electrical);
end entity CapacitorInterface;

architecture SimpleCapacitor of
             CapacitorInterface is
   quantity u across i through p to n;
begin
   i == C*u'dot;
end architecture SimpleCapacitor;
```

Multi-domain Modeling and Simulation, Fig. 5 VHDL-AMS model of capacitor component

Modeling languages are used to provide a textual description of the ports and of the equations in a specific syntax. For example, in Modelica the capacitor from Fig. 3 can be defined as shown in Fig. 4 (keywords of the language are written in boldface). In VHDL-AMS the capacitor model can be defined as shown in Fig. 5.

One difference between Modelica and VHDL-AMS is that in Modelica, all equations need to be explicitly given, and port variables (such as p.i) can be directly accessed in the model (Fig. 4).

Instead, in VHDL-AMS (and some other modeling languages), port variables cannot be accessed in a model, and instead via the "**quantity** .. **across** .. **through** .. **to** .." construction, the relationships between the port variables are implicitly defined and correspond to the Modelica equations "0 = p.i + n.i" and "u = p.v − n. v."

Simulation of Multi-domain Systems

Collecting all the component equations of a multi-domain system model together with all connection equations results in a DAE system:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{w}, \mathbf{y}, \mathbf{u}, t) \tag{1}$$

where $t \in \mathbb{R}$ is time, $\mathbf{x}(t) \in \mathbb{R}^{n_x}$ is a vector which contains variables that appear differentiated, $\mathbf{w}(t) \in \mathbb{R}^{n_w}$ is a vector of algebraic variables, $\mathbf{y}(t) \in \mathbb{R}^{n_y}$ is a vector of output variables, $\mathbf{u}(t) \in \mathbb{R}^{n_u}$ is a vector of input variables, and $\mathbf{f} \in \mathbb{R}^{n_x+n_w+n_y}$ is a vector of functions that represent the right-hand sides of the equations of the DAE system. The equations in (1) can be solved numerically with an integrator for DAE systems; see, for example, Brenan et al. (1996). For DAEs that are linear in their unknowns, a complete theory for solvability is available based on *matrix pencils*, see, for example, Brenan et al. (1996), and also reliable software for their analysis (Varga 2000).

Unfortunately, only certain classes of *nonlinear* DAEs can be *directly* solved numerically in a reliable way. For example, there are DAEs where no unique solution exists anymore when the step-size of the integrator approaches zero. For such systems step-size control is difficult, and numerical integration may easily fail. Domain-specific software, as, e.g., for mechanical systems, transforms the underlying DAE system into a form that can be more reliably solved using domain-specific knowledge. Hereby, certain equations of the DAE system are analytically differentiated, and special integration methods are used to solve the resulting overdetermined set of DAEs. In multi-domain simulation software, the following approaches are used:

(a) The DAE system (1) is directly solved numerically using an implicit integration method, such as a linear multistep method. Typically, all VHDL-AMS simulators use this approach.
(b) The DAE system (1) is symbolically transformed in a form that is equivalent to a set of ODEs, and then either explicit or implicit ODE or DAE integration methods are used to numerically solve the transformed system. The transformation is usually based on the algorithms of Pantelides (1988) and of Mattsson and Söderlind (1993) or some variants of them and might require to analytically differentiate equations. Typically, Modelica-based simulators, but also EcosimPro, use this approach.

For many models both approaches can be applied successfully. There are, however, systems where approach (a) is successful and fails for (b) or vice versa.

DAEs (1) derived from modeling languages usually have a large number of equations but with only a few unknowns in every equation. In order to solve DAEs of this kind efficiently, both with (a) or (b), typically graph theory and/or sparse matrix methods are utilized. For method (b) the fundamental algorithms have been developed by Elmqvist (1978) and later improved in further publications. For a recent survey and comparison of some of the algorithms, see Frenkel et al. (2012).

Solving the DAE system (1) means to solve an initial value problem. In order that integration can be started, a consistent set of initial variables $\dot{\mathbf{x}}_0 = \dot{\mathbf{x}}(t_0)$, $\mathbf{x}_0 = \mathbf{x}(t_0)$, $\mathbf{w}_0 = \mathbf{w}(t_0)$, $\mathbf{y}_0 = \mathbf{y}(t_0)$, $and\ \mathbf{u}_0 = \mathbf{u}(t_0)$ has to be determined first at the initial time t_0 which is a nontrivial task. For example, often (1) shall start in steady state, that is, it is required that $\dot{\mathbf{x}}_0 := \mathbf{0}$ and therefore at the initial time (1) is required to satisfy

$$\mathbf{0} = \mathbf{f}(\mathbf{0},\ \mathbf{x}_0,\ \mathbf{w}_0,\ \mathbf{y}_0,\ \mathbf{u}_0,\ t_0) \tag{2}$$

(2) is an algebraic system of nonlinear equations in the unknowns $\mathbf{x}_0$, $\mathbf{w}_0$, $\mathbf{y}_0, and\ \mathbf{u}_0$. These are $n_x + n_w + n_y$ equations for $n_x + n_w + n_y + n_u$

unknowns. Therefore, n_u further conditions must be provided (usually some elements of $\mathbf{u_0}$ and/or $\mathbf{y_0}$ are fixed to desired physical values). Solving (2) for the unknowns is also called "DC operating point calculation" or "trimming." Nonlinear equation solvers are based on iterative methods that require usually a sufficiently accurate initial guess for all unknowns. In a large multi-domain system model, it is not practical that reasonable initial values must be provided for every DAE variable and therefore methods are needed to solve (2) even if generic guess values in a library are provided that might be far from the solution of the system at hand.

For analog electronic circuit simulations, a large body of theory, algorithms, and software is available to solve (2) based on homotopy methods. The basic idea is to solve a sequence of nonlinear algebraic equation systems by starting with an easy to solve simplified system, characterized by the homotopy parameter $\lambda = 0$. This system is continuously "deformed" until the desired one is reached at $\lambda = 1$. The solution at iteration i is used as guess value for iteration $i+1$, and at every iteration, the solution is usually computed with a Newton–Raphson method. The simplest such approach is "source stepping": the initial guess values of all electrical components are set to "zero voltage" and/or "zero current." All (voltage and current) sources start at zero, and their values are gradually increased until the desired source values are reached. This method may not converge, typically due to the severe nonlinearities at switching thresholds in logical circuits.

There are several, more involved approaches, called "probability-one homotopy" methods. For these method classes, proofs exist that they converge for all initial conditions globally with probability one (so practically always). These algorithms can only be applied for certain classes of DAEs; see, for example, the "variable stimulus probability-one homotopy" of Melville et al. (1993).

Although strong results exist for analog electrical circuit simulators, it is difficult to generalize them to the large class of multi-domain systems covered by a modeling language. In Modelica a "homotopy" operator was introduced into the language (Sielemann et al. 2011) in order that a library developer can formulate simple homotopy methods like the "source stepping" in a component library. A generalization of probability-one methods for multi-domain systems was developed in the dissertation of Sielemann (2012) and was successfully applied to air distribution systems described as one-dimensional thermo-fluid pipe flow.

Models for Control Systems

Models for Analysis

The multi-domain models from section "Modeling Language Principles" can be utilized to evaluate the properties of a control system by simulation. Also control systems can be designed by nonlinear optimization where at every optimization step one or several simulations of a plant model are executed. Furthermore, modeling environments usually provide pre- and/or post-processing methods to linearize the nonlinear DAE system (1) around a steady-state operating point or an operating point along the simulation trajectory

$$\begin{aligned} \mathbf{x}(t) &\approx \mathbf{x}_{op} + \Delta\mathbf{x}(t), \quad & \mathbf{w}(t) &\approx \mathbf{w}_{op} + \Delta\mathbf{w}(t), \\ \mathbf{y}(t) &\approx \mathbf{y}_{op} + \Delta\mathbf{y}(t), \quad & \mathbf{u}(t) &\approx \mathbf{u}_{op} + \Delta\mathbf{u}(t) \end{aligned} \tag{3}$$

resulting in

$$\begin{aligned} \Delta\dot{\mathbf{x}}_{\text{red}} &= \mathbf{A}\,\Delta\mathbf{x}_{\text{red}} + \mathbf{B}\,\Delta\mathbf{u} \\ \Delta\mathbf{y} &= \mathbf{C}\,\Delta\mathbf{x}_{\text{red}} + \mathbf{D}\,\Delta\mathbf{u} \end{aligned} \tag{4}$$

where $\mathbf{x}_{op}, \mathbf{w}_{op}, \mathbf{y}_{op}, and\, \mathbf{u}_{op}$ define the operating point, $\Delta\mathbf{x}_{\text{red}}$ is a vector consisting of elements of $\Delta\mathbf{x}$, vector $\Delta\mathbf{w}$ is eliminated by exploiting the algebraic constraints, and **A, B, C,** and **D** are constant matrices. Simulation tools provide linear analysis and synthesis methods on this linearized system and/or export it for usage in an environment such as Julia, Maple, Mathematica®, MATLAB®, or Python®.

Multi-domain models might also be used directly in nonlinear Kalman filters, moving horizon estimators, or nonlinear model predictive

control. For example, the company ABB is using moving horizon estimation and nonlinear model predictive control based on Modelica models in its product OPTIMAX® to significantly improve the start-up process of power plants (Franke and Doppelhamer 2006), for the control of virtual power plants and for other purposes.

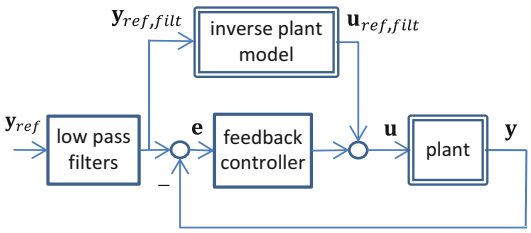
Inverse Models

A large body of literature exists about the theory of nonlinear control systems that are based on *inverse* plant models; see, for example, (Isidori 1995). Methods such as feedback linearization, nonlinear dynamic inversion, or flat systems use an inverse plant model in the control loop. However, a major obstacle is how to *automatically* utilize an inverse plant model in a controller without being forced to manually set up the equations in the needed form which is not practical for larger systems. Modeling languages can solve this problem as discussed below.

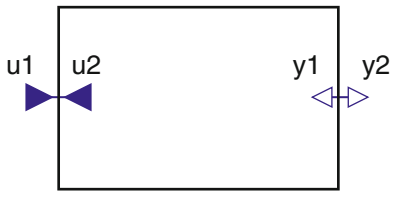
Nonlinear inverse models can be utilized in various ways in a control system. The simplest approach, as feed forward controller, is shown in Fig. 6. Under the assumption that identical equations (1) are used for the plant and the inverse plant model (but the plant model has $\mathbf{u}$ as input and $\mathbf{y}$ as output, whereas the inverse plant model has $\mathbf{y}$ as input and $\mathbf{u}$ as output) and start at the same initial state, then from the construction, the control error $\mathbf{e}$ is zero and $\mathbf{y} = \mathbf{y}_{\text{ref,filt}}$, so $\mathbf{y}$ is identical to the low-pass filtered reference signals. In other words, $\mathbf{y} \approx \mathbf{y}_{\text{ref}}$ for reference signals that have a frequency spectrum below the cutoff frequency of the low-pass filters. Since the assumption is actually not fulfilled, there will be a nonzero control error $\mathbf{e}$, and the feedback controller has to cope with it. This controller structure with a nonlinear inverse plant model has the advantage that the feed forward part is useful over the complete operating range of the plant.

Various other structures with nonlinear plant models are discussed in Looye et al. (2005), such as compensation controllers, feedback linearization controllers, and nonlinear disturbance observers. In Olsson et al. (2017), a Modelica extension is defined to directly construct (sampled data) feedback linearization controllers from (continuous-time) Modelica plant models. Furthermore, integration algorithms are proposed to solve nonlinear inverse models with hard real-time requirements.

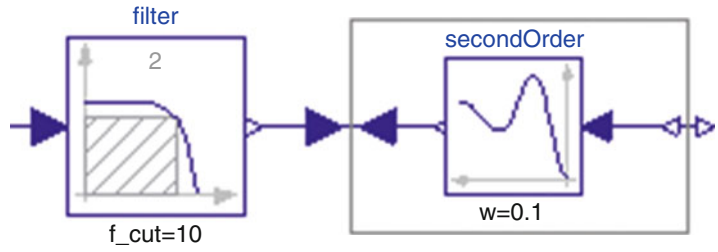
It turns out that nonlinear inverse plant models can be generated automatically with the techniques that have been developed for modeling languages; see section "Modeling Language Principles." In particular, constructing an inverse model from (1) means that the inputs u are defined to be outputs, so they are no longer knowns but unknowns, and outputs y are defined to be inputs, so they are no longer unknowns but knowns. The resulting system is still a DAE system and can therefore be handled as any other DAE system. Therefore, defining an inverse model with a modeling language just requires exchanging the definition of input and output signals. In Modelica, this can be graphically performed with the nonstandard input–output block from Fig. 7. This block has two inputs and two outputs and is described by the equations:



Multi-domain Modeling and Simulation, Fig. 6 Controller with inverse plant model in the feed forward path. The inverse plant model needs usually also derivatives of $\mathbf{y}_{\text{ref}}$ as inputs. These derivatives are provided by appropriate low-pass filters



Multi-domain Modeling and Simulation, Fig. 7 Modelica InverseBlockConstraint block



Multi-domain Modeling and Simulation, Fig. 8 Inversion of a second-order system in Modelica

$$u1 = u2; \quad y1 = y2;$$

From a block diagram point of view, this looks strange. However, from a DAE point of view, this just states constraints between two input and two output signals. In Fig. 8, it is shown how this block can be used to invert a simple second-order system: the output of the low-pass filter is connected to the output of the second-order system (note, a direction connection of this kind violates the input–output block semantics, but an indirect connection via the InverseBlockConstraint block is possible) and therefore this model computes the input of the second-order system, from the input of the filter by inverting the second-order system.

A Modelica environment will generate from this type of definition the inverse model, thereby differentiating equations analytically and solving algebraic variables of the model in a different way as for a simulation model. The whole transformation is nontrivial, but it is just the standard method used by Modelica tools as for any other type of DAE system.

The question arises whether a solution of the inverse model exists or is unique and whether the model is stable (otherwise, it cannot be applied in a control system). In general, a nonlinear inverse model consists of linear and/or nonlinear algebraic equation systems and of linear and/or nonlinear differential equations. Therefore, from a formal point of view, the same theorems as for a general DAE system applies, see Brenan et al. (1996). Furthermore, all these equations need to be solved with a numerical method. For some classes of systems, it can be shown that a unique mathematical solution exists and that the system is stable. However, in general, one cannot expect that it is possible to provide such a proof for complex inverse plant models. In some cases it is possible to approximate the plant model so that its inverse is guaranteed to be stable. In more involved cases, it might only be possible to perform simulations at many operating points to show that the probability of a stable inverse model is high. Note, nonlinear inverse plant models have been successfully utilized by automatic generation from a Modelica plant model, in particular for robots, satellites, aircrafts, vehicles, and thermo-fluid systems.

The Functional Mockup Interface

Many different types of simulation environments are in use. One cannot expect that a generic approach as sketched in section "Modeling Language Principles" will replace all these environments with their rich set of domain-specific knowledge, analysis, and synthesis features. Practically, all simulation environments provide a vendor-specific interface in order that a user can import components that are not describable by the simulation environment itself. Typically, this requires to provide a component as a set of C or Fortran functions with a particular calling interface. In the control community, the most widely used approach of this kind is the S-Function interface from the MathWorks, where Simulink is used as integration platform, and model components from other environments are imported as S-Functions.

In 2010 the vendor-independent standard "Functional Mockup Interface 1.0" was developed, followed by a significantly improved version 2.0 in 2014 (Modelica Association 2014). An again significantly improved version 3.0 is under development as of June 2019 (see

https://fmi-standard.org/faq/). FMI is a low-level standard for the exchange of models between different simulation environments. This standard allows to exchange only the model equations (called "FMI for Model Exchange") and/or the model equations with an embedded solver (called "FMI for Co-Simulation"). This standard was quickly adopted by many simulation environments, and in 2019 there are more than 130 tools that support it (for an actual list of tools, see https://fmi-standard.org/tools/). In particular Modelica environments export Modelica models in this format, and therefore, Modelica multi-domain models can be imported in other environments with low effort.

A software component which implements the FMI is called Functional Mockup Unit (FMU). An FMU consists of one zip file with extension ".fmu" containing all necessary components to utilize the FMU either for Model Exchange, for Co-Simulation, or for both. The following summary is an adapted version from Blochwitz et al. (2012):

1. An *XML-file* contains the definition of all exposed variables of the FMU, as well as other model information. It is then possible to run the FMU on a target system without this information, that is, without unnecessary overhead. Furthermore, this allows determining all properties of an FMU from a text file, without actually loading and running the FMU.
2. A set of *C-functions* is provided to execute model equations for the Model Exchange case and to simulate the equations for the Co-Simulation case. These C-functions can be provided either in binary form for different platforms or in source code. The different forms can be included in the same model zip file.
3. Further data can be included in the FMU zip file, especially a model icon (bitmap file), documentation files, maps and tables needed by the model, and/or all object libraries or DLLs that are utilized.

The main application area of FMI is offline simulation, but it is also used for (soft) real-time applications, see, for example, the direct support of FMI in the real-time systems of dSPACE (https://www.dspace.com). In Brembeck (2019) a nonlinear (continuous-time) Modelica model of a highly maneuverable electric vehicle is used as starting point of a dedicated tool chain to generate automatically (sampled data) FMUs used as discrete-time prediction models for an extended moving horizon state estimator and an extended Kalman filter.

Summary and Future Directions

Multi-domain modeling based on a DAE description and defined with a modeling language is an established approach, and many tools support it. This allows to conveniently define plant models from many domains for the design and evaluation of control systems. Furthermore, nonlinear inverse plant models can be easily constructed with the same methodology and can be utilized in various ways in nonlinear control systems.

Current research focuses on the support of the complete life cycle: defining requirements of a system formally on a "high level," considerably improving testing by checking these requirements automatically when evaluating a system design by simulations, see, for example, Bouskela and Jardin (2018), and providing complete tool chains to embedded systems. For example, in the European ITEA project EMPHYSIS (embedded systems with physical models in the production code software; https://itea3.org/project/emphysis.html), the variant eFMI of FMI is being developed in the years 2017–2021 so that whole tool chains from multi-domain modeling environments to production code on automotive electronic control units and other embedded devices with *hard real-time requirements* become feasible. This will allow convenient and fast target code generation of nonlinear controllers, optimization-based controllers, extended and unscented Kalman filters, or moving horizon estimators from multi-domain models.

Furthermore, the methodology itself is further improved. Especially, with the prototyping

platform Modia (Elmqvist et al. 2017), new directions are evaluated, such as hybrid 2D schematic/3D graphical user interfaces, close integration of complex data structures with equation-based modeling (e.g., to model multiphase, multicomponent fluids or collision handling of 3D mechanical systems), improved and new symbolic transformation algorithms (Otter and Elmqvist 2017), code generation for large models, support of multi-domain Dirac impulses, and changing the number of equations during simulation.

Cross-References

- Computer-Aided Control Systems Design: Introduction and Historical Overview
- Descriptor System Techniques and Software Tools
- Extended Kalman Filters
- Feedback Linearization of Nonlinear Systems
- Interactive Environments and Software Tools for CACSD
- Model Building for Control System Synthesis
- Modeling of Dynamic Systems from First Principles
- Model-Predictive Control in Practice
- Moving Horizon Estimation
- Nonlinear Filters

Bibliography

Bezanson J, Edelman A, Karpinski S, Shah V (2017) Julia: a fresh approach to numerical computing. SIAM Rev 59:65–98. https://doi.org/10.1137/141000671

Blochwitz T, Otter M, Akesson J, Arnold M, Clauß C, Elmqvist H, Friedrich M, Junghanns A, Mauss J, Neumerkel D, Olsson H, Viel A (2012) Functional Mockup Interface 2.0: the standard for tool independent exchange of simulation models. In: Proceedings of the 9th international modelica conference, Munich, 3–5 Sept 2012, pp 173–184. https://doi.org/10.3384/ecp12076173

Bouskela D, Jardin A (2018) A new temporal language for the verification of cyber-physical systems. In: 2018 annual IEEE international systems conference (SysCon). https://doi.org/10.1109/SYSCON.2018.8369502

Brembeck J (2019) Nonlinear constrained moving horizon estimation applied to vehicle position estimation. Sensors 2019, 19, 2276. https://doi.org/10.3390/s19102276

Brenan KE, Campbel SL, Petzold LR (1996) Numerical solution of initial-value problems in differential-algebraic equations. SIAM, Philadelphia

Elmqvist H (1978) A structured model language for large continuous systems. Dissertation. Report CODEN:LUTFD2/(TFRT–1015), Department of Automatic Control, Lund Institute of Technology, Lund. https://lup.lub.lu.se/search/ws/files/4602422/8570492.pdf

Elmqvist H, Henningsson T, Otter M (2017) Innovations for future modelica. In: Proceedings of the 12th international modelica conference, Prague. https://doi.org/10.3384/ecp17132693

Franke R, Doppelhamer J (2006) Online application of modelica models in the industrial IT extended automation system 800xA. In: Proceedings of the 6th international modelica conference, Vienna, 4–5 Sept 2006, pp 293–302. https://modelica.org/events/modelica2006/Proceedings/sessions/Session3c2.pdf

Franke R, Casella F, Otter M, Sielemann M, Mattsson SE, Olsson H, Elmqvist H (2009) Stream connectors – an extension of modelica for device-oriented modeling of convective transport phenomena. In: Proceedings of the 7th international modelica conference, Como, pp 108–121. https://doi.org/10.3384/ecp09430078

Frenkel J, Kunze G, Fritzson P (2012) Survey of appropriate matching algorithms for large scale systems of differential algebraic equations. In: Proceedings of the 9th international modelica Conference, Munich, 3–5 Sept 2012, pp 433–442. https://doi.org/10.3384/ecp12076433

IEEE 1076.1-2017 (2017) IEEE standard VHDL analog and mixed-signal extensions. Standard of IEEE. https://standards.ieee.org/standard/1076_1-2017.html

Isidori A (1995) Nonlinear control systems, 3rd edn. Springer, Berlin/New York

Karnopp DC, Margolis DL, Rosenberg RC (2012) System dynamics: modeling, simulation, and control of mechatronic systems, 5th edn. Wiley, Hoboken

Looye G, Thümmel M, Kurze M, Otter M, Bals J (2005) Nonlinear inverse models for control. In: Proceedings of the 4th international modelica conference, Hamburg, 7–8 Mar 2005, pp 267–279. https://modelica.org/events/Conference2005/online_proceedings/Session3/Session3c3.pdf

Mantooth HA, Vlach M (1992) Beyond spice with saber and MAST. In: IEEE international symposium on circuits and systems, San Diego, vol 1, 10–13 May 1992, pp 77–80

Mattsson SE, Söderlind G (1993) Index reduction in differential-algebraic equations using dummy derivatives. SIAM J Sci Comput 14:677–692

Melville RC, Trajkovic L, Fang SC, Watson LT (1993) Artificial parameter homotopy methods for the DC

operating point problem. IEEE Trans Comput Aided Des Integr Circuits Syst 12:861–877. https://doi.org/10.1109/43.229761

Modelica Association (2014) Functional mock-up interface for model exchange and co-simulation, Version 2.0. https://fmi-standard.org/downloads/

Modelica Association (2017) Modelica – a unified object-oriented language for systems modeling. Language specification, Version 3.4. https://www.modelica.org/documents/ModelicaSpec34.pdf

Olsson H, Otter M, Mattsson SE, Elmqvist H (2008) Balanced models in modelica 3.0 for increased model quality. In: Proceedings of the 6th international modelica conference, Bielefeld, pp 21–33. https://www.modelica.org/events/modelica2008/Proceedings/sessions/session1a3.pdf

Olsson H, Mattsson SE, Otter M, Pfeiffer A, Brger C, Henriksson D (2017) Model-based embedded control using Rosenbrock integration methods. In: Proceedings of the 12th international modelica conference, Prague. https://doi.org/10.3384/ecp17132517

Otter M, Elmqvist H (2017) Transformation of differential algebraic array equations to index one form. In: Proceedings of the 12th international modelica conference, Prague. https://doi.org/10.3384/ecp17132565

Pantelides CC (1988) The consistent initialization of differential-algebraic systems. SIAM J Sci Stat Comput 9:213–233

Sieleman M (2012) Device-oriented modeling and simulation in aircraft energy systems design. Dissertation, Dr. Hut. ISBN:978-3-8439-0504-6

Sielemann M, Casella F, Otter M, Clauß C, Eborn J, Mattsson SE, Olsson H (2011) Robust initialization of differential-algebraic equations using homotopy. In: Proceedings of the 8th international modelica conference, Dresden. https://doi.org/10.3384/ecp1106375

Varga A (2000) A descriptor systems toolbox for Matlab. In: Proceedings of CACSD'2000 IEEE international symposium on computer-aided control system design, Anchorage, 25–27 Sept 2000, pp 150–155. https://ieeexplore.ieee.org/iel5/7209/19415/00900203.pdf