EVOLUTION OF CONTINUOUS-TIME MODELING AND SIMULATION

Karl Johan Åström
Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
E-mail: kja@control.LTH.se

Hilding Elmqvist
Dynasim AB
Research Park Ideon
SE-223 70 Lund, Sweden
E-mail: Elmqvist@Dynasim.se

Sven Erik Mattsson
Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
E-mail: SvenErik@control.LTH.se

KEYWORDS

History, simulation, modeling, differential equations, differential-algebraic equations, object-orientation

ABSTRACT

Modeling and simulation have experienced an amazing development since its beginning in the 1920s. At that time, the technology was available only at a handful of university groups. Today it is available on the desk of all engineer who needs it. The paper presents the current status of modeling and simulation. It draws on the historical perspective to explain how the field has developed. Particular emphasis is given to shifts in technology and paradigms.

INTRODUCTION

Modeling and simulation are indispensable when dealing with complex engineering systems. It makes it possible to do essential assessment before systems are built, it can alleviate the need for expensive experiments and it can provide support in all stages of a project from conceptual design, through commissioning and operations. The following quote from one of the early pioneers Prof. Vannevar Bush, who worked on problems in power systems, is still highly relevant:

> "Engineering can proceed no faster than the mathematical analysis on which it is based. Formal mathematics is frequently inadequate for numerous problems pressing for solution, and in the absence of radically new mathematics, a mechanical solution offers the most promising and powerful attack wherever a solution in graphical form is adequate for the purpose. This is usually the case in engineering problems."

Technology has naturally been an important factor in the development of simulation. Analog techniques were predominant from 1920 to 1950. Major changes took place when digital computers were available and simulation techniques have then exploited the advances in digital computers and software techniques such as computer graphics.

There is a large literature on simulation in wide range of engineering journals. Early developments are described in Brennan and Linebarger (1964) and Tiechroew *et al.* (1967). More recent overviews found in the books Kreutzer (1986), Kheir (1988), Cellier (1991) and Linkens (1993) and the survey papers Otter and Cellier (1995), Cellier *et al.* (1995) and Marquardt (1996). Lists of software are published yearly by the Society for Computer Simulation.

In this paper we will essentially follow the historical development. We will start with analog techniques which were based on ordinary differential equations and block diagrams. A family of digital simulators which have inherited many of the properties of analog computing are then treated. The advantages and the limitations of the analog heritage are discussed. Domain oriented special purpose simulators are then described. This is a natural way to discuss issues such as efficiency and user friendlyness. Then we will discuss a new generation of simulators which are based on object oriented modeling. They cover multiple domains and permit multiple views of the system. They also have efficient ways to deal with decomposition and aggregation.

ANALOG SIMULATION

The first simulators were analog. The idea is to model a system in terms of ordinary differential equations and then make a physical device that obeys the equations. The physical system is initialized with proper initial values and its development over time then mimics the differential equation.

Simulation of an ordinary differential equation (ODE)

$$\frac{dx}{dt} = f(t,x) \tag{1}$$

can be accomplished by integrators and function generation. It was actually shown by Kolmogorov (1957) that continuous functions of several variables could be approximated by combinations of scalar products and generation of scalar functions. This idea was used for function generation in early analog simulation although it was not known at the time that the method was generally applicable.

The mechanical differential analyzer developed by Vannevar Bush at MIT was the first general purpose tool to simulate dynamical systems [Bush (1931)]. Variables were represented by angles. Integration was performed by the ball and disc integrator, which had been used

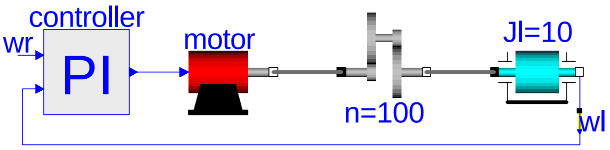

Figure 1 Schematic picture of a motor drive.

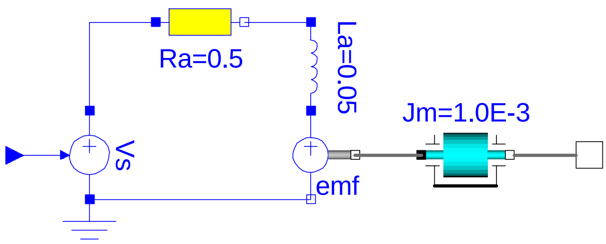

Figure 2 A motor model.

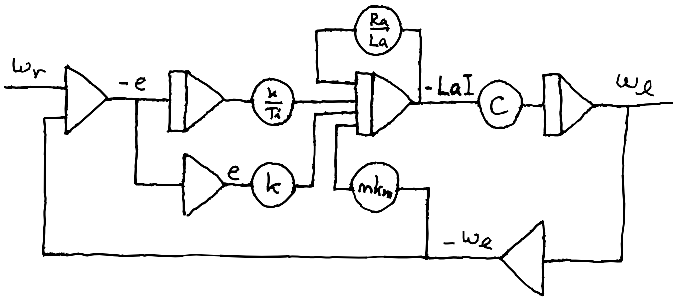

Figure 3 Schematic for simulating the motor drive on an analog computer. The constant $C = nk_m/(J_\ell + n^2 J_m)/L_a$.

in planimeters for a long time. Function generation was made by gear boxes and cams. Torque amplifiers were used for amplification. A major shift in technology occured with the publication of the paper Ragazzini *et al.* (1947), which demonstrated that that simulation could be done electronically. Variables were represented as voltages in the electronic simulators. This made it easy to plot variables and to set up a problem. It also paved the way for industrial production and wide spread use of analog computing, see Paynter (1989). Companies that produced electronic simulators also emerged e.g. Philbrick, Applied Dynamics and Electronic Associates were some of the major actors. Aerospace companies were major customers. A good overview of analog techniques is given in Jackson (1960).

How to Perform Analog Simulation

In analog computing, a differential equation (1) must be represented in terms of the fundamental operations, integration, addition, multiplication, and function generation. Since the analog computer has limited range and resolution the variables must be scaled. Scaling is tedious but it also gives useful insight into the structure of the problem, see Canon (1973). It is also necessary make the interconnections required to represent the function $f(t, x)$ in (1). In electronic analog computers the interconnections were made by patching cables in a board and parameters were set with potentiometers. The whole procedure of setting up a problem was tedious. Execution was fast but precision was limited. The analog computers were highly interactive, because parameters could be changed during the operation.

An Example

A simple example will be used for illustration throughout the paper. The system, which is shown in Fig. 1–2, is a motor drive with an electric motor, a gearbox, a load and a controller. Equations are given in Appendix A.

To make an analog simulation of the motor drive the equations have to be represented by the operations that can be executed by an analog computer, i.e., integration, summation and multiplication by constants.

To obtain such a representation we must first introduce suitable state variables, i.e., variables that account for storage. They are typically the variables that appear differentiated in the equations. For the particular model these variables are ω_ℓ, ω_m, I and x. Since ω_m and ω_ℓ are related through an algebraic equation (A.5) one of the variables has to be eliminated. Elimination of ω_m using equations (A.2) and (A.5) gives.

$$\begin{aligned}
\frac{d\omega_\ell}{dt} &= nk_m I/(J_\ell + n^2 J_m) \\
\frac{dI}{dt} &= (-R_a I + k(\omega_r - \omega_\ell + \frac{1}{T_i}x) - nk_m\omega_\ell)/L_a \\
\frac{dx}{dt} &= \omega_r - \omega_\ell
\end{aligned} \qquad (2)$$

which is an explicit state space representation. To obtain an analog simulation diagram we assume that the state variables are available as output voltages of the integrators. Voltages representing the derivatives as expressed by equation (2) can then be obtained by multiplication by constants and addition and introduced as inputs to the integrators as shown in Fig. 3. Multiplication by a constant is done by a potentiometer, addition and integration by operational amplifiers with feedback. In practice there are additional complications. Potentiometers can only represent multiplication with numbers that are smaller than one. All variables have to be scaled. In the figure potentiometers are represented by circles, summers by triangles and integrators by a triangle with a rectangle.

Algebraic Loops

Several manual steps were required to transform the equations given in Appendix A to the form (1). These calculations are easy to do in the specific case but they are quite tedious and error prone for more complex systems. The connection to the physical processes are partially lost in the transformations. It is easy to recognize the controller in the analog simulation diagram in Fig. 3, but the gearbox and the inertias are no longer visible. They appear combined in the coefficient C. The reason for this is that analog simulation cannot deal with differential algebraic equations. If it is attempted to simulate the basic equations in Appendix A directly there will be a loop which only contains algebraic equations. This is caused by the relation (A4). This is not easy to discover without analysing the equations. The phenomenon which is well known in analog simulation is called the algebraic loop problem. One way of dealing with it in analog computing was to introduce a small capacitor in the algebraic loop. This amounts to replacing an algebraic equation with a differential equation that settles quickly. This could give large initial transients but it often worked well.

NUMERICAL INTEGRATION

Numerical solution of differential equation is an essential ingredient of digital simulators.

Ordinary Differential Equations (ODE)

There are many ways to find approximate numerical solutions to an ordinary differential equations such as (1). The methods are based on the idea of replacing the differential equations by a difference equation. Eulers method is based on approximation of the derivative by a first order difference. There are more efficient techniques such as Runge-Kutta and multi-step methods. These methods were well known when digital simulators emerged in the 1960s. This field of numerical mathematics experienced a revival because of the impact of digital computers. Important contributions were given to stability of difference approximations, see Dahlquist (1959) and Henrichi (1962). Automatic step length adjustment was another important contribution, see Fehlberg (1964). Systems with both fast and slow modes (stiff systems) posed a particular difficulty for explicit methods. It is necessary to choose a very short step length to have numerical stability, which gives a very slow simulation.

Differential Algebraic Equations (DAE)

The natural models for dynamical systems are differential algebraic equations (DAE), i.e. a mixture of differential and algebraic equations. This is true even for the simple servo in Appendix A. A general form of a DAE is

$$g(t, x, \dot{x}) = 0 \tag{3}$$

It is not always possible task to convert such an equation to an ordinary differential equation because the Jacobian $\partial g/\partial \dot{x}$ may not be invertible.

Numerical methods for differential algebraic equations appeared in 1970. The paper Gear (1971) is one of the early publications. Efficient codes came later, see Brenan *et al.* (1989) and Hairer *et al.* (1989).

Numerical integration of ODEs and DAEs are very active research fields which continue to have strong impact on modeling and simulation, see Hairer *et al.* (1987) and Hairer and Wanner (1991). Among the interesting development are improved algorithms, a better structuring of the code where algorithms and error control are separated, see Gustafsson (1993) and Olsson (1996). Algorithms for differential algebraic equations are still not as well developed as algorithms for ordinary differential equations.

THE ANALOG SIMULATION HERITAGE

When digital computers appeared it was natural to explore if they could be used for simulation. The development was triggered by Selfridge (1955) which showed how a digital computer can emulate a differential analyzer. There was a very intense activity, see Brennan and Linebarger (1964) and Tiechroew *et al.* (1967). By 1967 there were more than 23 different programs available. Typical examples are MIMIC from Wright Patterson [Peterson and Sansom (1965)], DYNASAR [Lucke (1965)] from General Electric, DSL/90 [Syn and Linebarger (1966)] and CSMP [Brennan and Silberberg (1968)] from IBM. One reason for the intense development was that the a problem could be entered in the form of analog computer diagrams and previous working practices could be reused. It seemed easier to change the technology than to change the paradigm.

The CSSL Standard

The CSSL report [Strauss (ed.) (1967)], commissioned by the Simulation Council Inc (SCi), was a major milestone since it unified the concepts and language structures of the available simulation programs.

In CSSL a system can be described in three different ways, as an interconnection of blocks as in MIDAS and DYNASAR, by mathematic expressions as in MIMIC and DSL/90 and by conventional programming construct as in FORTRAN. CSSL defined a set of operators like INTEG which emulates the integrator of the analog computer. Other built-in operators are IMPL for breaking algebraic loops and applying an iterative scheme for its solution, DELAY for time delays, HYST for hysteresis. Automatic sorting of the equations to proper order of the calculation is another feature of CSSL that was inherited from MIMIC. The reason was to avoid spurious delays when the inherent parallelism in the modeled devices are mapped on a sequential machine.

The user can define new block types by means of a MACRO definition. A macro has a list of formal parameters. Their appearances in expressions are textually substituted when the macro is invoked. There is a special REDEFINE statement to generate unique names so that each invocation could use its own local variables. However, there was no naming convention to access local variables. The macro feature can be seen as a poorman's class description. It is more powerful than a function in a programming language since it has local storage in built-in operators like INTEG and DELAY and the REDEFINE statement to obtain instance variables. In an object-oriented language such instance variables are typically accessed by dot-notation. Macro handling is done without using information about the textual content of the macro. This can lead to severe errors just like macros in C programs.

In addition to modeling features, CSSL also contains statements for selecting integration routines and their parameters, for controlling the simulation, and for documentation of results. A perspective on the CSSL standard is given in Rimvall and Cellier (1986).

ACSL

A number of software products were based on the CSSL definition. One example is ACSL from Mitchell and Gauthier Associates, Mitchell and Gauthier (1976) which was the defacto standard for simulation for a long time. ACSL is based on CSSL but certain modifications and many enhancements were done. In particular, the macro language, the set of built-in operators and the set of control statements are considerably stronger than in CSSL. Constructs for combined continuous/discrete modeling were later added. It is thus possible to schedule events when certain variables crosses limits. The definitions were made in such a way that integration routines can accurately find the time of the event by utilizing zero-crossing functions and a root finder. ACSL was implemented as a preprocessor to Fortran. Fortran statements can be part of the model.

To illustrate the style of textual modeling in CSSL and ACSL we give the following ACSL model of the motor drive.

```
PROGRAM drive

MACRO motor(T, V, w, Ra, La, km)
  MACRO REDEFINE I
  T = km*I
  I = INTEG((-Ra*I + V - km*w)/La, 0.0)
MACRO END

INITIAL
  CONSTANT km=1.1616, Ra=0.5, La=0.02
  CONSTANT k=1, Ti=1, wr = 1
  CONSTANT n=100, Jl=10, Jm=2
  J = Jl + Jm*n**2
END ! of initial

DYNAMIC
DERIVATIVE
  T = motor(Vs, n*wl, Ra, La, km)
  wl = INTEG(n*T/J, 0.0)          ! Load
  e = wr - wl                     ! Control error
  Vs = k*(e + INTEG(e, 0.0)/Ti)   ! PI controller
END ! of derivative
  TERMT(t .ge. 1)   ! Terminate after 1 second
END ! of dynamic
END ! of program
```

A macro has been defined for the motor. All characters after ! are considered as comments by ACSL. Notice that due to the problem of constraints in the shaft (algebraic loop) it is necessary to make the same manual reduction of the equations as was done for pure analog simulation. The actual formulas for the combined inertia J is calculated in an INITIAL section.

Simnon

A different approach than using textual macros for structuring was taken in the program Simnon which was developed at Lund University starting 1972 [Elmqvist (1975)] and ported to PC in 1985 [Elmqvist *et al.* (1985)]. Simnon was part of a research program in computer aided control engineering, see Åström (1983).

Simnon uses continuous systems, discrete systems and connecting systems. Continuous and discrete systems are described in state space form. Variables are named locally in each system with names categorized as input, output and state. The functions which give the rate of change of states, the next state function, and the output are specified by assignment statements. The connecting system is a list of assignments of the form u[sys2] = y[sys1] that tells how the inputs are defined from outputs. The notation u[sys2] means the same as the dot-notation sys2.u. A restriction is that only two level hierarchies are supported. Simnon had a nice feature for modeling mixed sampled and continuous systems. Discrete systems where provided with a special variable which tells the next time a discrete system should be executed. This variable can be updated with any expression. This gives a simple way to deal with many different sampling schemes. It was also possible to define systems in Fortran or Pascal. This was however complicated to use.

Simnon was initially developed for interactive simulation on a PDP-15 environment. There was a strong separation between the modeling language and the command language for executing the simulation and for analysing the results. Basic simulation is executed by six command only: compile systems, initialize variables, change parameters, execute simulations, plot results and organize plots. There is an extensive error checking. Simnon uses global sorting of the assignments to find the proper calculation order during compilation. Simnon has its own machine code generator, which gave fast recompilation. Parameters could be changed without recompilation. There is a macro command facility so that several commands can be grouped together to form a new command. This was very useful for documentation.

Graphical Block Diagram Modeling

Graphical representations of the type shown in Fig. 3 were used to represent models in terms of integrators, adders and potentiometers in the early days of analog simulation. Because of the limited input-output facilities of early digital computers it was necessary to revert to textual representations in the digital simulators. Prototype graphical environments were designed in the mid 1970s using a cathode ray tube (CRT) and light pen for drawing block diagrams [van den Bosch and Bruijn (1977)]. However, graphical modeling was not widely used until modern work stations and the PC with raster graphics became generally available.

Boeings simulator EASY5 from 1976 was provided with

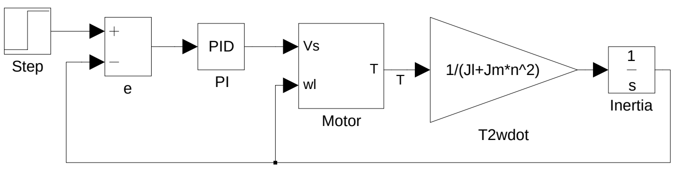

Figure 4 A SIMULINK model for the motor drive in Fig. 1.

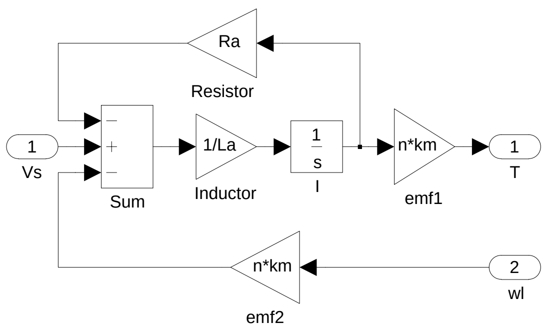

Figure 5 A SIMULINK model for the motor in Fig. 2.

a graphical user interface. The matrix environments MATLAB and MATRIX_X which appeared in the 1980s were provided with modeling tools. SystemBuild [Shah *et al.* (1985)] which is integrated with MATRIX_X appeared in 1984 and SIMULINK (originally called SIMULAB) which is integrated with MATLAB appeared in 1991 [Grace (1991)]. A new PC based system VisSim [Darnell and Kolk (1990)] appeared in 1990. Mitchell and Gauthier introduced the ACSL Graphics Modeller in 1993.

To illustrate the graphical modeling systems we will present a SIMULINK model for the simple motor drive shown in Fig. 4– 5. It is interesting to note the similarities with the analog diagram in Fig. 3.

The graphical modeling tools share many properties. The model is constructed from graphical blocks with input and output ports that are connected by drawing lines. Parameters can be set in dialog windows which appear by clicking on the blocks. There are extensive block libraries for simple arithmetic operations, functions, transfer functions, linear systems in state space form, controllers, etc. It is possible to define aggregate models with inputs and outputs and to reuse such composite models in any number of hierarchical levels. Connections represent either scalar or vector variables with fixed causality. EASY5 and ACSL also have textual representations of expressions which means that the behavior of a new block can be defined textually. SIMULINK does not have any textual representation of expressions except Matlab notation, but it may slow down the simulation because the expressions are interpreted. SIMULINK blocks can be defined as C-code, but this method is quite complicated.

The analog computing paradigm with its requirement of explicit state models (ODE) is a fundamental limitation of block diagram modeling. The blocks have a unidirectional data flow from inputs to outputs. This is the reason why an object like a gearbox in the simple motor drive cannot be dealt with directly. It is also the reason why motor and load inertia appear in the mixed expression in the SIMULINK model in Fig. 4. A severe consequence is that it is cumbersome to build physics based model libraries in the block diagram languages. In EASY5 there is a special connector which permits connections with bidirectional dataflow. A general solution to this problem required a paradigm shift.

MODELING IN SPECIFIC DOMAINS

It is possible to design modeling environments that are very user friendly by restricting the domain of the models. A large number of tools of this type have been developed in several branches of engineering. A model is assembled simply by connecting components from predefined libraries. The idea is to relieve the user from model development by providing readymade models or model components, which can be assembled to a complete model. A brief discussion of some tools of this type is given in this section.

Electrical systems

The system SPICE [Nagel and Pederson (1973); Nagel (1975)], which was developed for analog modeling of electrical circuits is a typical example. Electrical circuits are formed simply by connecting resistors, capacitors, inductors and transistors. VHDL-AMS [IEEE (1997)] is an extension of the discrete circuit modeling language VHDL for combined continuous and discrete models. VHDL-AMS is a large and rich modeling language targeted mainly at the application domain of electronics hardware.

EMTP (Electro Magnetic Transients Program) [Ele (1989)] and its relatives ATP and EMTDC, are industry standard for electro-magnetic transients in power systems. It was developed in the late 1960s by Dommel at the Bonneville Power Administration. The program PSS/E (Power System Simulator) is a widely used program for simulation of transmission networks which was developed in the mid 1970s.

Mechanical systems

Multi-body systems are used to model 3-dimensional mechanical systems, such as robots, satellites and vehicles. Vectors and matrices are natural elements for modeling. Interfaces to CAD databases are needed to generate models automatically from a CAD topology and to animate the results.

The first tools appeared in the end of the 1970s. DADS was developed at the University of Iowa appeared around 1984. Several commercial tools are now available. ADAMS is a popular software. The German Aerospace Establishment, DLR has a long tradition from Fadyna (1977) and Medyna (1984) to SIMPACK. An interesting feature is that the code for simulation of multi-body systems use both symbolic and numeric computing.

Energy and process systems

SpeedUp [Sargent and Westerberg (1964); Perkins and Sargent (1982)] from the Centre for Process Systems Engineering, Imperial College in London, is widely used for dynamic simulation in chemical engineering. The same group also developed gPROMS (general Process Modelling System) [Barton and Pantelides (1994)]. SpeedUp and gPROMS exploit numerical DAE solvers and automatic differentiation for analytical Jacobian calculations.

The Modular Modeling System (MMS) is a system for simulation of nuclear and fossil power plants developed by EPRI [Divakaruni (1986)]. It consists of a user interface and modules for ACSL or EASY5. Since ACSL requires the models to be on explicit state space form, there are many approximations to avoid algebraic loops. This make it very difficult to modify existing models and add new ones.

There are several tools, for simulation of heating, ventilation and air conditioning e.g., HVACSIM$^+$ [Clark (1985)] and TRNSYS (1983). These languages are in the spirit of the CSSL languages. To support exchange of models between tools a standard Neutral Model Format (NMF) was proposed to the building and energy systems simulation community in 1989 [Sahlin and Sowell (1989); Sahlin *et al.* (1996)]. The language is formally controlled by a committee within Am. Soc. for Heating, Refrigerating and Air-Conditioning Engineers. Several independently developed NMF tools and libraries exist.

Summary

Software for specific domains is very easy to use if the problem fits the tool directly. It is however often very difficult to modify the tools and to add new features. Some things that can be learned from the modeling environment is that model libraries are very useful and that the analog computing paradigm is too limited.

PHYSICAL MODELING

A typical procedure for physical modeling is to cut a system into subsystems and to account for the behavior at the interfaces. Each subsystem is modeled by balances of mass, energy and momentum and material equations. The complete model is obtained by combining the descriptions of the subsystems and the interfaces. This approach requires a different paradigm for modeling. A model is considered as a constraint between system variables. This leads naturally to DAE descriptions. The approach is very convenient for building reusable model libraries.

Bond Graphs

Bond Graphs are directed graphs where the subsystems are the nodes and the power flow in the system is shown by the branches, see Karnopp and Rosenberg (1968). The connections are called power bonds and have asso-

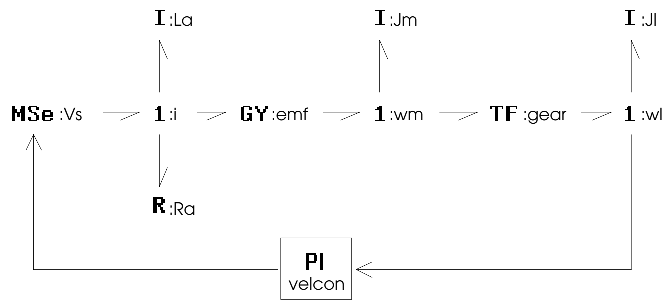

Figure 6 A bond graph model for the motor drive in Fig. 1.

ciated effort variables (such as voltage and torque) and flow variables (current and angular velocity). A bond graph, developed by the tool 20-SIM [Broenink (1997)], of the simple motor drive is given in Fig. 6. The voltage source for the motor is described as a "modulated effort source" (MSe). Its power flow is divided between the resistor, inductor and electro-motorical force (emf). Since these elements are connected in series they have the same current (flow). Such a distribution of power with retained flow, is accomplished by a so called 1-junction in the bond graph. There are also 0-junctions which maintain the same effort. The conversion from electric power to mechanical power is accomplished with a gyrator (GY). The power from the gyrator is stored in the rotating intertia (Jm) and flows through the gear box, which is modeled by a transformer (TF), to the energy storage (Jl).

There are algorithms to assign causalities in a bond graph. The bond graph of Fig. 6 is non-trivial because of the constraint between wm and wl. A tool will typically warn about "derivative causality". In most cases, it is possible to use a good DAE-solver which is robust enough to handle such high index problems.

Dymola

The Dynamic Modeling Language (Dymola) by Elmqvist (1978), was an early effort to support physical modeling. Its designer, who also wrote Simnon had experienced the limitations of the analog computing paradigm in modeling thermal power stations. There was too large a gap between the user's problem and the model description that the computer would understand. The user had to collect all the equations of the different components and manually transform them in order to give the expressions for the derivatives. Modeling should be much closer to the way an engineer builds a real system, first trying to find standard components like motors, pumps and valves from manufacturers' catalogues with appropriate specifications and interfaces. Only if there does not exist a particular subsystem, the engineer would actually construct it.

The design of Dymola was strongly influenced by the first object-oriented language, Simula, see Birtwistle *et al.* (1973) and by experience of the power of symbolic computations. Simula with its hierarchical problem decomposition and use of classes for reuse provided a methodology for orienting a model to its physical subsystem. The behavior description in Simula was designed for discrete event simulation with sequential programming with coroutines and methods. This was not suitable for continuous modeling, since the laws of physics are equations. Coupling between objects were in Simula expressed by calling methods in other objects. Physical coupling is typically symmetric and it imposes constraints on the interface variables, like Kirchhoff's voltage and current laws.

Dymola introduced model classes, which at the time were called model types, and a submodel statement to invoke classes similar to the ref-construct in Simula. Submodels were described by equations. A construct called cut was introduced to name connection mechanisms such as wires, pipes and shafts. It declares variables that were associated with the cut and are constrained when cuts are connected to each other to define the topology. These constructs made it possible to describe models in many different domains like electrical circuits, mechanics, thermo-dynamics, etc. in a uniform way. Such a model description was equivalent to the collection of the equations of each instantiated model class and the formed connection equations, i.e., a DAE.

Symbolic formula manipulation was used extensively to convert the DAE to ODEs. The graph theoretic methods in Tarjan (1972) and Wiberg (1977) were found to be very effective. They were typically designed for sparse linear problems but were applicable to non-linear DAEs since only structural information was utilized. The resulting linear equations were solved symbolically and remaining nonlinear equations numerically.

The parser, the structural analysis and the formula manipulation was originally implemented in Simula. The program was not useful for large problems at the time due to limited memory capacity (64 kwords) on machines like Univac 1108. The largest problem handled was a thermal power plant with 300 equations, i.e, it was not industrially applicable. Cellier used Dymola in modeling classes in the late 1980s and in his modeling book Cellier (1991). Development of Dymola was resumed in 1992 when Elmqvist founded Dynasim AB in Lund and made a commercial version. The major computer architecture and operating systems then supported large linear address space suitable for symbolic computation on large models. Hardware and software had finally caught up with the ideas. In the mean time a few other developments had taken place. They will be discussed below.

Omola and OmSim

By the mid 1980s there had been major developments in hardware, software and numerics that made it worth while to continue the modeling effort that started with Dymola. Work stations and personal computers with powerful graphics were available. Object oriented programming and software had advanced considerably. Numerical algorithms for solving DAEs were available [Brenan *et al.* (1989); Hairer *et al.* (1989)]. Powerful computing environments were also available.

A research program in object oriented modeling was initiated at the Department of Automatic Control in Lund. Preliminary work was done on modeling, see Åström and Kreutzer (1986) and interactive graphics Elmqvist and Mattsson (1989). The project soon focused on modeling languages. Omola (Object-oriented modeling language) appeared in late 1988, see Andersson (1989). The first prototypes were written in CommonLisp and KEE. As the language design stabilized it was written in C++, see Mattsson *et al.* (1993). Models can be decomposed hierarchically with well defined interfaces that describe interaction. All model components are represented as classes. Inheritance and specialization support easy modification. Omola supports behavioral descriptions in terms of DAEs and difference equations.

Omola has primitives for describing discrete events which allows definition of classes to support high level descriptions as finite state machines and Petri nets [Andersson (1994)]. A kernel for model representation based on Omola and an interactive environment, OmSim, was also implemented see Mattsson *et al.* (1993) and Andersson (1994). The complete environment includes a graphical model editor, consistency analysis, symbolic analysis and manipulation, ODE and DAE solvers and interactive plotting. The software is implemented in C++.

Several applications were made in parallel with the development of Omola. The idea was to explore if the language constructs were suitable for different domains. Applications studied include chemical processes, power generation and power networks, see Nilsson (1993). This work resulted in guidelines for structure and class hierarchy decomposition and organization of model libraries. Issues relating to component composition versus multiple inheritance were also raised. Language extensions were suggested. Typical examples are constructs for system structuring using arrays of components and language elements to define regular connections patterns. It is useful for modeling of a distillation column which consists of a set of trays connected in series. Component arrays are also useful for spatial discretization. Medium and machine decomposition was proposed as method to separate the description of the process media from the processing units. This can be supported by allowing model classes to be parameters.

Modelica

In addition to Dymola and Omola, there are several other languages with similar ideas defined, such as: ASCEND [Piela *et al.* (1991)], gPROMS, NMF [Sahlin *et al.* (1996)], ObjectMath [Fritzson *et al.* (1995)], SIDOPS+ [Breunese and Broenink (1997), Smile [Kloas *et al.* (1995)], and U.L.M. [Jeandel *et al.* (1996)]. The situation was thus similar to the mid 1960s when CSSL was defined as a unification of the techniques and ideas of many different simulation programs. An international effort was initiated in September 1996 for the

```
model MotorDrive
  Motor      motor;
  PI         controller;
  Gearbox    gearbox(n=100);
  Shaft      Jl(J=10);
  Tachometer wl;
equation
  connect(controller.out, motor.in);
  connect(motor.flange  , gearbox.a);
  connect(gearbox.b     , Jl.a);
  connect(Jl.b          , wl.a);
  connect(wl.w          , controller.in);
end MotorDrive;
```

Figure 7 A Modelica model of the system in Fig. 1.

purpose of bringing together expertise in object-oriented physial modeling and defining a modern uniform modeling language. The language is called Modelica[1]. Version 1.0 was finished in September 1997.

Modelica is intended for modeling within many application domains such as electrical circuits, multibody systems, drive trains, hydraulics, thermodynamical systems, and chemical processes etc. It supports several formalisms: ordinary differential equations (ODE), differential-algebraic equations (DAE), bond graphs, finite state automata, and Petri nets etc. Modelica is intended to serve as a standard format so that models arising in different domains can be exchanged between tools and users. More information about Modelica can be found in Elmqvist *et al.* (1998).

Fig. 1 and Fig. 2 are actually Modelica models of the motor drive presented in Dymola. The physical components and their interconnections are shown graphically. Parameters can be set by clicking on the components. The textual Modelica representation is shown in Fig. 7. Modelica has constructs for storing and exchanging graphical information (positions, connection lines, icons). This has, however, been omitted in Fig. 7.

CONCLUSIONS

Simulation is essential for dealing with complex systems. Techniques for modeling and simulation have advanced substantially since the mid 1920s. In this paper we have attempted to capture the historical development. It started with mechanical differential analyzers which were able so solve a few ordinary differential equations. These systems were available only to a small group of researchers. The replacement of mechanics by electronics in the 1950s was a major advance which led to industrialization and a significant increase of the availability of modeling and simulation. Another major advance occured when the analog computers were replaced by digital computers in the 1960s. It is interesting that only one of the analog computing company made the transition to digital simulators, see Gilbert and Howe (1978). Another major advance happened in the 1990s when personal computers and computer graphics became generally available. It is interesting to observe that ideas change slower than technology. The analog simulation paradigm is still prevailing. It is only in the 1990s that it has been more widely realised that a paradigm shift is needed. This is driven by demands from users to be able to simulate complex multi-domain models and advances in object oriented programming, software for differential algebraic systems, symbolic computing and advanced graphics. The modern approaches build on non-causal modeling with mathematical equations and the use of object-oriented constructs to facilitate reuse of modeling knowledge.

[1]Modelica™ is a trade mark of the Modelica Design Group

REFERENCES

ANDERSSON, M. (1989): "An object-oriented modeling environment." In IAZEOLLA *et al.*, Eds., *Simulation Methodologies, Languages and Architectures and AI and Graphics for Simulation*, 1989 European Simulation Multiconference, Rome, pp. 77–82. The Society for Computer Simulation International.

ANDERSSON, M. (1994): *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis ISRN LUTFD2/TFRT--1043--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ÅSTRÖM, K. J. (1983): "Computer aided modeling, analysis and design of control systems — A perspective." *IEEE Control Systems Magazine*, **3**, pp. 4–16.

ÅSTRÖM, K. J. and W. KREUTZER (1986): "System representations." In *Proc. IEEE Control Systems Society Third Symposium on Computer-Aided Control Systems Design (CACSD)*. Arlington, Virginia.

BARTON, P. and C. PANTELIDES (1994): "Modeling of combined discrete/continuous processes." *AIChE J.*, **40**, pp. 966–979.

BIRTWISTLE, G. M., O. J. DAHL, B. MYHRHAUG, and K. NYGAARD (1973): *SIMULA BEGIN*. Auerbach Publishers Inc.

BRENAN, K. E., S. L. CAMPBELL, and L. R. PETZOLD (1989): *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. North-Holland, Amsterdam. Also available in SIAM's Classics in Applied Mathemathics series, No. 14, 1996.

BRENNAN, R. D. and R. N. LINEBARGER (1964): "A survey of digital simulation—Digital analog simulator programs." *Simulation*, **3**.

BRENNAN, R. D. and M. Y. SILBERBERG (1968): "The system/360 continuous system modeling program." *Simulation*, **11**, pp. 301–308.

BREUNESE, A. P. and J. F. BROENINK (1997): "Modeling mechatronic systems using the SIDOPS+ language." In *Proceedings of ICBGM'97, 3rd International Conference on Bond Graph Modeling and Simulation*, Simulation Series, Vol.29, No.1, pp. 301–306. The Society for Computer Simulation International.

BROENINK, J. F. (1997): "Modelling, simulation and analysis with 20-sim." *Journal A, Benelux Quarterly Journal on Automatic Control*, **38:3**, pp. 20–25. Special issue on Computer Aided Control System Design, CACSD.

BUSH, V. (1931): "The Differential Analyzer: A new machine for solving differential equations." *Journal of the Franklin Institute*, **212**, pp. 447–488.

CANON, M. R. (1973): "Magnitude and time scaling of state-variable equations for analog/hybrid computing." *Simulation*, **21**, pp. 23–28.

CELLIER, F., H. ELMQVIST, and M. OTTER (1995): "Modeling from physical principles." In LEVINE, Ed., *The Control Handbook*, pp. 99–108. CRC Press, Boca Raton, FL, USA.

CELLIER, F. E. (1991): *Continuous System Modeling*. Springer-Verlag, New York, USA.

CLARK, D. R. (1985): "HVACSIM$^+$ Building systems and equipment simulation program." Reference Manual NBSIR 85-3243. U.S. Department of Commerce, National Bureau of Standards, Washington, DC.

DAHLQUIST, G. (1959): *Stability and error bounds in the numerical integration of ordinary differential equations*. Transactions No. 130. The Royal Institute of Technology, Stockholm, Sweden.

DARNELL, P. A. and R. A. KOLK (1990): "An interactive simulation and control design environment." In *Proceedings of the 1990 European Simulation Symposium*, pp. 56–60. SCS.

DIVAKARUNI, S. M. (1986): "The application of simulation in large energy system analysis." *Modeling, Identification and Control*, **6**, pp. 231–247.

ELECTRIC POWER RESEARCH INSTITUTE, INC. (EPRI) (1989): *Electromagnetic Transients Program (EMTP) Revised Rule Book Version 2.0*.

ELMQVIST, H. (1975): "SIMNON— An interactive simulation program for nonlinear systems — User's manual." Technical Report TFRT-7502. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1978): *A Structured Model Language for Large Continuous Systems*. PhD thesis TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H., K. J. ÅSTRÖM, and T. SCHÖNTHAL (1985): *Simnon — User's Guide for MS-DOS Computers*. Studentlitteratur, Lund, Sweden.

ELMQVIST, H. and S. E. MATTSSON (1989): "A simulator for dynamical systems using graphics and equations for modelling." *IEEE Control Systems Magazine*, **9:1**, pp. 53–58.

ELMQVIST, H., S. E. MATTSSON, and M. OTTER (1998): "Modelica — The new object-oriented modeling language." In *Proceedings of the 12th European Simulation Multiconference (ESM'98)*. SCS, The Society for Computer Simulation, Manchester, UK.

FEHLBERG, E. (1964): "New high-orderRunge-Kutta formulas with step size control for systems of firts and second order differential equations." *ZAMM*, **44**.

FRITZSON, P., L. VIKLUND, D. FRITZSON, and J. HERBER (1995): "High-level mathematical modeling and programming." *IEEE Software*, **12:3**.

GEAR, C. W. (1971): "Simultaneously numerical solution of differential-algebraic equations." *IEEE Transactions on Circuit Theory*, **CT-18**, pp. 217–225.

GILBERT, E. O. and R. M. HOWE (1978): "Design considerations in a multi-computer." In *AFIPS Conference Proceedings*, vol. 47, pp. 385–393.

GRACE, A. C. W. (1991): "SIMULAB, An integrated environment for simulation and control." In *Proceedings of the 1991 American Control Conference*, pp. 1015–1020. American Autom. Control Council.

GUSTAFSSON, K. (1993): "Object oriented implementation of software for solving ordinary differential equations." *Scientific Programming*, **2**, pp. 217–225.

HAIRER, E., C. LUBICH, and M. ROCHE (1989): *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Lecture Notes in Mathematics No. 1409. Springer-Verlag, Berlin.

HAIRER, E., S. NØRSETT, and G. WANNER (1987): *Solving Ordinary Differential Equations I — Nonstiff Problems*. Computational Mathematics No. 8. Springer-Verlag, Berlin.

HAIRER, E. and G. WANNER (1991): *Solving Ordinary Differential Equations II — Stiff and Differential-Algebraic Problems*. Computational Mathematics No. 14. Springer-Verlag, Berlin.

HENRICHI, P. (1962): *Discrete variable methods in ordinary differential equations*. John Wiley & Sons, Inc.

IEEE (1997): "Standard VHDL Analog and Mixed-Signal Extensions." Technical Report IEEE 1076.1. IEEE.

JACKSON, A. S. (1960): *Analog Computation*. McGraw-Hill, New York.

JEANDEL, A., F. BOUDAUD, P. RAVIER, and A. BUHSING (1996): "U.L.M: Un Langage de Modélisation, a modelling language." In *Proceedings of the CESA'96 IMACS Multiconference*. IMACS, Lille, France.

KARNOPP, D. C. and R. C. ROSENBERG (1968): *Analysis and simulation of multiport systems — The bond graph approach to physical system dynamics*. MIT Press, Cambridge, MA, US.

KHEIR, N. A., Ed. (1988): *Systems Modeling and Computer Simulation*. Marcel Dekker, Inc, New York, USA, USA.

KLOAS, M., V. FRIESEN, and M. SIMONS (1995): "Smile — A simulation environment for energy systems." In SYDOW, Ed., *Proceedings of the 5th International IMACS-Symposium on Systems Analysis and Simulation (SAS'95)*, vol. 18–19 of *Systems Analysis Modelling Simulation*, pp. 503–506. Gordon and Breach Publishers.

KOLMOGOROV, A. N. (1957): "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition." *Dokl. Akad. Nauk USSR*, **114**, pp. 953–956.

KREUTZER, W. (1986): *System Simulation — Programming styles and Languages*. Addison-Wesley, Reading MA, USA.

LINKENS, D. A., Ed. (1993): *CAD for Control Systems*. Marcel Dekker, Inc.

LUCKE, VIRGIL, H. (1965): "Dynasar — Analysis methods developed for the dynamic system analyzer." In *Preprints of the 1965 Joint Automatic Control Conference*, pp. 780–786.

MARQUARDT, W. (1996): "Trends in computer-aided process modeling." *Computers chem. Engng*, **20**, pp. 591–607.

MATTSSON, S. E., M. ANDERSSON, and K. J. ÅSTRÖM (1993): "Object-oriented modelling and simulation." In LINKENS, Ed., *CAD for Control Systems*, pp. 31–69. Marcel Dekker, Inc.

MITCHELL, E. E. L. and J. S. GAUTHIER (1976): "Advanced continuous simulation language (ACSL)." *Simulation*, pp. 72–78.

NAGEL, L. (1975): "SPICE2: A computer program to simulate semiconductor circuits." Memorandum ERL-M520. Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, USA.

NAGEL, L. and D. O. PEDERSON (1973): "Simulation program with integrated circuit emphasis (SPICE)." Memorandum ERL-M382. Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA, USA.

NILSSON, B. (1993): *Object-Oriented Modeling of Chemical Processes*. PhD thesis ISRN LUTFD2/TFRT--1041--SE, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

OLSSON, H. (1996): "Object oriented solvers for initial value problems." In *Proceedings of the OONSCI96*.

OTTER, M. and F. E. CELLIER (1995): "Software for modeling and simulating control systems." In LEVINE, Ed., *The Control Handbook*, pp. 415–428. CRC Press, Boca Raton, FL, USA.

PAYNTER, H. M. (1989): "The differential analyzer as an active mathematical instrument." *IEEE Control Systems magazine*, **9**, pp. 3–8.

PERKINS, J. D. and R. W. H. SARGENT (1982): "SPEEDUP: A computer program for steady-state and dynamic simulation and design of chemical processes." In MAH AND REKLATIS, Eds., *Selected Topics in Computer-Aided Process Design and Analysis*, AIChE Symposium Series 78.

PETERSON, H. and F. SANSOM (1965): "MIMIC—A digital simulator program." SESCA Internal Memo No. 65-12. Wright-Patterson Air Force Base, Ohia, US.

PIELA, P., T. EPPERLY, K. WESTERBERG, and A. WESTERBERG (1991): "ASCEND: An object-oriented computer environment for modeling and analysis: the modeling language." *Computers and Chemical Engineering*, **15:1**, pp. 53–72.

RAGAZZINI, J. R., R. H. RANDALL, and F. A. RUSSELL (1947): "Analysis of problems in dynamics by electronic circuits." *Proc. IRE*, **35**, pp. 444–452.

RIMVALL, M. and F. CELLIER (1986): "Evolution and perspectives of simulation languages following the CSSL standard." *Modeling, Identification and Control*, **6**, pp. 181–199.

SAHLIN, P., A. BRING, and E.F.SOWELL (1996): "The Neutral Model Format for building simulation, Version 3.02." Technical Report. Department of Building Sciences, The Royal Institute of Technology, Stockholm, Sweden.

SAHLIN, P. and E. SOWELL (1989): "A neutral format for building simulation models." In *Proceedings of Building Simulation '89*. IBPSA, Vancouver, Canada.

SARGENT, R. W. H. and A. W. WESTERBERG (1964): "SPEED-UP in chemical engineering design." *Trans. Inst. Chem. Eng. (London)*, **42**, pp. 190–197.

SELFRIDGE, R. G. (1955): "Coding a general purpose digital computer to operate as a differential analyzer." In *Proceedings 1955 Western Joint Computer Conference, IRE*.

SHAH, S. C., M. A. FLOYD, and L. L. LEHMAN (1985): "MATRIX$_X$: Control design and model building CAE capability." In JAMSHIDI AND HERGET, Eds., *Computer-Aided Control Systems Engineering*, pp. 181–207. Elsevier Science Publishers B.V. (North-Holland).

STRAUSS (ED.), J. C. (1967): "The SCi continuous system simulation language (CSSL)." *Simulation*, **9**, pp. 281–303.

SYN, W. M. and R. N. LINEBARGER (1966): "DSL/90 — A digital simulation program for continuous system modeling." In *AFIPS Conference Proceedings*, vol. 28.

TARJAN, R. E. (1972): "Depth-first search and linear graph algorithms." *SIAM J. Computing*, **1**, pp. 146–160.

TIECHROEW, D., J. F. LUBIN, and T. D. TRUITT (1967): "Discussion of computer simulation and comparison of languages." *Simulation*, **9**, pp. 181–190.

TRNSYS (1983): "A transient simulation program." Reference Manual. Solar Energy Laboratory, University of Wisconsin, Wisconsin, US.

VAN DEN BOSCH, P. P. J. and P. BRUIJN (1977): "The directed digital computer as a teaching tool in control engineering; interactive instruction and design." In *Proceedings of the IFAC Symposium on Trends in Automatic Control Education*, pp. 260–271.

WIBERG, T. (1977): *Permutation of an Unsymmetric Matrix to Block Triangular Form*. PhD thesis, Department of Information Processing, University of Umeå, Umeå, Sweden.

APPENDIX A — Equations for the motor drive

The equations for the motor drive shown in Fig. 1 are given below.

The load is modeled as a simple rotating inertia. Let ω_ℓ be the angular velocity of the load, J_ℓ be the moment of inertia of the load and T_g the torque on the load from the gear box. The equation of motion of the load then becomes

$$J_\ell \frac{d\omega_\ell}{dt} = T_g \tag{A.1}$$

The electric motor is modeled as shown in Fig. 2. Let ω_m be the angular velocity of the motor, J_m its moment of inertia, k_m the torque constant, I the rotor current, and T_m the torque exerted on the motor axis by the gear box. The equation of motion of the motor rotor then becomes

$$J_m \frac{d\omega_m}{dt} = k_m I - T_m \tag{A.2}$$

The electrical properties of the rotor can be characterized by the rotor resistance R_a and inductance L_a. If V_s is the voltage applied to the rotor winding Kirchhoff's law for the motor rotor becomes.

$$L_a \frac{dI}{dt} + R_a I = V_s - k_m \omega_m \tag{A.3}$$

The gearbox has the gear ratio, n. Neglecting the moment of inertia of the gears the gearbox can be modeled by

$$\omega_m = n\omega_\ell \tag{A.4}$$
$$T_g = nT_m \tag{A.5}$$

The controller is assumed to be a PI controller. Let ω_r be the set point of the controller. The power amplifier and the controller can then be modeled by

$$V_s = k(\omega_r - \omega_\ell + \frac{1}{T_i}x) \tag{A.6}$$
$$\frac{dx}{dt} = \omega_r - \omega_\ell \tag{A.7}$$

where k is the gain of the controller and the power amplifier and T_i is the integration time of the controller.

Summary

The model of the system is described by eight variables (ω_ℓ, ω_m, ω_r, T_m, T_g, I, V_s and x), eight parameters (J_ℓ, J_m, k_m, n, L_a, R_a, k, and T_i), four ordinary differential equations and three algebraic equations. For a typical experiment, we need also to specify the set point $\omega_r(t)$.