SIMULATION SOFTWARE – DEVELOPMENT AND TRENDS

F. Breitenecker and **I. Troch**
Vienna University of TechnologyVienna, Austria

Keywords: Simulation, Algebraic loop, Block diagram, CACSD tool, Computer experiments, Continuous simulation, CSSL-Standard, Differential-algebraic equation, Differential equation, Discrete simulation, Experimental frame, Hybrid simulation, Identification, Implicit model, Integration algorithms, Model frame, Modeling, ODE solver, Random number generator, Simulation engine, Simulation environments, Simulation software, Simulation system, Simulators, State, State vector, Validation, Verification

Contents

Summary

This contribution presents development and trends of simulation software, with emphasis on control systems, robotics, and automation. Simulation emerged in the 1960' in order to be able to analyze nonlinear dynamic system and to synthesize nonlinear control systems, and in order to investigate complex automation systems. Since that time simulation as problem solving tool has been developed towards the third pillar of science (beneath

theory and experiment), and in parallel simulation software has been developed further on. The paper first follows the roots from analog computation and control system simulation, followed by the introduction of the CSSL standard for simulation languages. The next chapter is devoted to numerical algorithms for solving differential equations, which build up the simulation engine of a simulator.

Of interest is the special relation between simulation software tools and computer aided control system design tools (CACSD tools), discussed in the following chapter. Recent developments are discussed in next chapters: advanced analysis methods in simulators, simulation with implicit models, hybrid simulation, simulation in specific domains, and hybrid and combined modeling and simulation.

The second part of the contribution deals first with discrete simulation software for e.g. automation of manufacturing systems and with object-oriented approaches and software implementation of simulators, which allow real hybrid simulation, and ends with an overview on choice and comparison of simulation software.

1. Introduction

Simulation software plays a major role in analysis of nonlinear control systems and complex automation systems. Simulation software, simulation languages, simulators, simulation systems, and simulation environments is computer software intended for simulation of dynamic systems at a higher level than programming languages can do. The different terms used reflect more or less the development of this special kind of software, and unfortunately, in the literature these terms are very often mixed.

A “simulation” is a method for solving a problem in dynamical systems, which investigates instead of the real system a model of the system. The method “simulation” itself consists of several steps or tasks, which are: (1) formulation of the problem, (2) data collection, (3) mathematical modeling, (4) computer implementation, (5) model validation, (6) model identification, (7) experiments with the model, (8) representation of results, and (9) interpretation of results.

Thereby the experiments with the model (7) are usually investigations in the time domain, as the system under investigation is dynamic. The ideal simulation system should support all these steps, but up to now no simulation system supports all steps sufficiently. Roughly speaking, a simulation language emphasizes on implementation and partly on experimenting and modeling, “simulation software” is a little bit more general supporting implementation and experimenting and partly modeling, a simulation system additionally offers help for data gathering and data pre-processing, for validation and for identification, and for data post-processing (result representation) - a simulation system may consist of several parts.

Finally, a simulation environment is an integrated software environment combining the various parts of a simulation system, with database interface, increased support for modeling. The term “simulator” does not fit in this pattern. In common understanding, previously it was used for a hardware-oriented simulator, like a flight simulator or any other training simulator, nowadays it stands more or less for an abbreviation of all other

terms.

Almost everywhere one meets the distinction between continuous simulation and discrete simulation. First, again we are faced with a veritable tower of Babylon. Roughly speaking, continuous modeling and simulation is based on time domain analysis of nonlinear system described by ODEs using ODE solvers, whereas discrete modeling and simulation is based on time domain analysis of queuing systems described by process flow using time event handling.

But in continuous modeling and simulation we meet discrete elements, like discrete controllers described by difference equations, or hybrid systems, where at some times the states change in discrete patterns. As discussed later on, the basic algorithm of a continuous simulator as well as of a discrete simulator is an event handling algorithm with more or less complicated state changes in the events. The distinction of and the difference in discrete and continuous modeling is caused by the sometimes very different education of people in discrete and continuous simulation.

2. Continuous Roots of Simulation

The classical analysis of dynamic systems, especially of control systems, consists of eigenvalue analysis of the linearized mathematical model. As systems became more and more complex, this linear analysis was not sufficient, and the need of analyzing the nonlinear system became evident. It became necessary to analyze the model in the time domain - by means of a simulator. The first simulators were purely hardware; they worked with the principle of analogy.

The mathematical model of the system was mimicked by a physical device that obeys the equations. The mechanical differential analyzer developed at MIT was the first hardware simulator for dynamical systems. States were represented by angles; integration was performed by the ball and disc integrator. In 1947, Ragazzini demonstrated that this simulation could be done electronically. States and variables were represented as voltages in an electronic circuit. So the analogue computers were borne, which were widely used until the 1980s.

Simulation with an analog computer was a time-consuming: choosing proper state variables (represented by voltages), creating the circuit diagram from the equations (very similar the control diagrams), scaling the states, patching the circuit, and recording the simulation output. In the late days of analog simulation, there were attempts to support this procedure by software: digital scaling programs, automatic patching devices driven by a block-oriented scaled analog model description, and prototype programs for transforming the equations into the block-oriented description.

Furthermore, a digital computer was coupled with the analogue simulator (called hybrid simulator), in order to control, perform and document experiments, and also to model digital control. In parallel, as soon as digital computers appeared, it was tried to use them for simulation. The basic idea was to simulate the analogue computer at the digital computer. Models could be entered in the form of analog computer diagrams (block-oriented description) or control loops, and previous working practices could be

used. In the late sixties there were about twenty different programs available, called simulation languages or simple “simulators” (because of the analogue heritage, e.g. MIMIC, DYNASAR, DSL, and CSMP.

Also at European Universities projects for "digital" simulators were started, based on the experience with analog and hybrid simulation. The HYBSYS project at Vienna University of Technology started as automatic patch system for an analog simulator. Next step was to replace the analog computer by a library of ODE solvers, to extend the model description appropriately, and to incorporate a runtime environment (simulation system HYBSYS). HYBSYS continued with a submodel structure similar to an object-oriented structure, which was very modern for those times. Further developments of HYBSYS partly implemented the Model- Method – Experiment – Concept – for details see later, and a parallelization concept.

3. CSSL Structure in Continuous Simulation

Simulation (i.e. time domain analysis) supported various developments in engineering and other areas, and simulation groups and societies were founded. One main effort of such groups was to standardize digital simulation programs and to work with a new basis: not any longer simulating the analog computer, but a self standing structure for simulation systems.

There were some unsuccessful attempts, but in 1968 the CSSL Standard (The CSSL Report commissioned by the Simulation Council Inc - Sci) became the milestone in the development: it unified the concepts and language structures of the available simulation programs, it defined a structure for the model, and it describes minimal features for a runtime environment.

The CSSL standard suggests structures and features for a model frame and for an experimental frame. This distinction is based on Zeigler’s concept of a strict separation of these two frames:

- The model frame has to provide a comfortable way for model description - on a much higher level than within a programming language
- The experimental frame should allow experimenting with the implemented model, either interactively, or with a sequence of batch commands

Model frame and experimental frame are the user interfaces for the heart of the simulation system, for the simulator kernel or simulation engine. The simulation engine drives the calculations in the time domain:

- it is started from commands of the model frame,
- it first initializes parameters and states (calculations may be parts of the model frame),
- then it calls and controls an ODE solver, which itself evaluates the model equations (given in the dynamic part of the model frame) and in parallel data are logged,
- and at the end it does final calculations and data recording (calculations may be

parts of the model frame)

This basic structure of a simulator - due to CSSL standard – is illustrated in Figure 1, an extended structure with service of discrete elements (see later) is given in Figure 5.

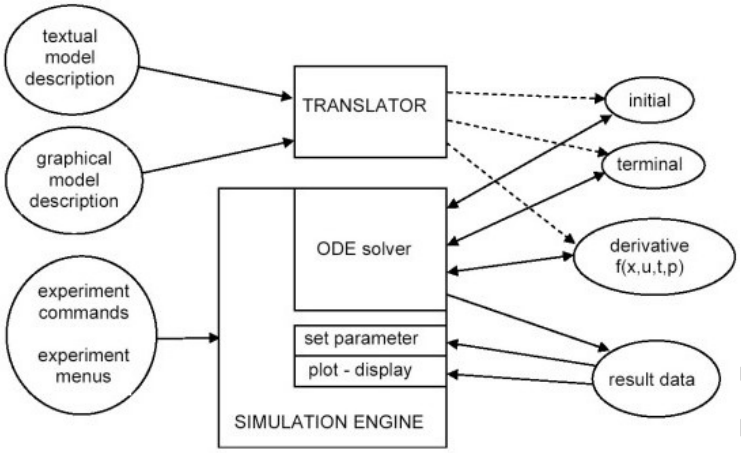

Figure 1: Basic structure of a simulation language due to CSSL standard

3.1. Structure of the Model Frame

In principle, in CSSL' model frame, a system can be described in three different ways, as an interconnection of blocks as in MIDAS, DYNASAR, ACSL Graphic Modeler, and SIMULINK, by mathematical expressions as in MIMIC, ESL, DSL, and ACSL, and by conventional programming constructs as in FORTRAN or C.

Mathematical basis is the state space description

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t, \mathbf{p}), \quad \mathbf{x}(t_0) = \mathbf{x}_0$$

Any kind of textual model formulation, of graphical blocks or structured mathematical description or host languages constructs must be transformed to an internal state equation of the structure given above, so that the vector of derivatives $\mathbf{f}(\mathbf{x}, \mathbf{u}, t, \mathbf{p})$ can be calculated for a certain time instant $\mathbf{f}_i = \mathbf{f}_i(\mathbf{x}(t_i), \mathbf{u}(t_i), t_i, \mathbf{p})$. This vector of derivates is fed into an ODE solver in order to calculate a state update $\mathbf{x}_{i+1} = \Phi(\mathbf{x}_i . \mathbf{f}_i, h)$, h stepsize (all controlled by the simulation engine). Essential is CSSL's concept of SECTIONs or REGIONs, giving a certain structure to the model description. First, CSSL defines a set of operators like INTEG which formulates parts of the state space description for the system governing ODEs (and which emulates the integrator of the analog computer). Other memory operators like DELAY for time delays and HYST for hysteresis, TABLE functions for generating (technical) tables, and transfer functions of arbitrary order complete dynamic modeling parts (strongly influenced by requirements for control systems). The dynamic model description builds up the DYNAMIC or DERIVATIVE section of the model description, for simple models the only one section. Automatic sorting of the equations (blocks) to proper order of the calculation is another very essential feature of CSSL. The model can be formulated in arbitrary sequence – allowing

structured parts, the translator puts the statements in the proper order for calculating the derivative vector. Furthermore, the user can define new block types by means of a MACRO definition. The macro feature is a predecessor of object-oriented modeling: the macro definition is similar to a class definition; the macro invocation is the instantiation of an object of the class defined. The CSSL standard also extends the classical model description in state space form, aiming for better structured modeling and more efficient implementations. Sometimes together with the state space equations we also meet parameter equations, parameter dependent initial values, and calculations with the terminal values (e.g. for cost functions in an optimization):

$$\mathbf{C(p) = 0}, \quad \mathbf{I}(\mathbf{p}, \mathbf{x}(t_0)) = \mathbf{0}, \quad E(\mathbf{p}, \mathbf{x}(t_f)) = \mathbf{0}$$

In principle, all this calculations could be done in the dynamic model description, but then they are calculated at each evaluation of the derivative vector of the ODE solver – although they have to be calculated only once.

In the following, a non-structured and structured model description for the pendulum is given. The well known equations (length l, mass m, damping coefficient d) are

$$\ddot{\varphi}(t) = -\frac{g}{l}\sin\varphi - \frac{d}{m}\dot{\varphi}, \quad \varphi(0) = \varphi_0 = \frac{\pi}{n}, \dot{\varphi}(0) = \dot{\varphi}_0$$

where now parameter equations are added (in parentheses the intrinsic parameter equations of the model are given):

$$\varphi_0 = \frac{\pi}{n}, d = \sqrt{2D}, \ \varphi^{\deg}(t_f) = \frac{\pi}{180}\varphi(t_f) \quad (a = \frac{g}{l}, b = \frac{d}{m})$$

In ACSL (Advanced Continuous Simulation Language; Mitchell and Gauthier 1976, 1981) a model description can be formulated structured and non-structured. In table 1 the non-structured model is given:

```
PROGRAM math_pendulum
! --- unstructured CSSL model
---------------------
! --- model parameters
---------------------------
  CONSTANT m=1, l=1, d=0.3 ! kg, m, N*s/m
  CONSTANT g=9.81, pi=3.141592653
  CONSTANT dphi0=0, phi0=0.78539816 !
! --- model equations
---------------------------
    phi0=pi/4
  phi = integ ( dphi,        phi0)
  dphi = integ (-(g/l)*sin(phi)-(d/m)*dphi,
dphi0)
    phi_grad = phi*180/pi
END ! of Program
---------------------------------
```

Table 1: Non-structured textual model description

This unstructured model description given above is translated totally into the derivative vector, so that initial value and final value are calculated at each evaluation of the derivative vector.

The following structured model description in ACSL generates more efficient code: only the DERIVATIVE section is translated into the derivative vector function, while INITIAL section and TERMINAL section are translated into functions called immediately before and after the ODE solver, as shown in table 2.

```
PROGRAM math_pendulum
! --- structured CSSL model ------------------------
! --- model parameters -----------------------------
  CONSTANT m=1, l=1, d=0.3 ! kg, m, N*s/m
  CONSTANT g=9.81, pi=3.141592653; dphi0=0, pintel=2
INITIAL ! calculation with parameters--------------
    phi0 = pi/pintel; a = g/l; b = d/m
END ! of INITIAL -----------------------------------
     DERIVATIVE ! ODE model -----------------------
       phi = integ ( dphi,        phi0)
      dphi = integ ( -gdl*sin(phi) - ddm*dphi, dphi0)
     END ! of DERIVATIVE -------------------------
TERMINAL ! calculations with final states ---------
    phi_grad = phi*180/pi
END ! of TERMINAL ----------------------------------
END ! of Program -----------------------------------
```

Table 2: Structured textual model description

With graphical window systems, graphical model descriptions became important. Here the roots go back on the one side to analog computation using patching diagrams, and on the other side to control techniques with signal flow diagrams.

Consequently, simulation systems offered this kind of model description, either as stand-alone model frame, or as extension of a textual model frame. The following picture shows the model for the pendulum using SIMULINK, one of the most used graphical simulators.

But in the graphical SIMULINK model (Figure 2) one disadvantage appears: the graphical structure consisting of directed dynamic signal flow allows almost no structure for dynamic calculations and static calculations; e. g. one finds the calculation of the final value and the calculation of the dependent parameters as dynamic block, which is evaluated at each call of the ODE solver (except the calculations for the initial value).

From 2000 on, graphical model frames were enriched by structures discussed before; in SIMULINK, triggered subsystems can be used for such purposes (Figure 3, possible triggered subsystems shown in subwindow).

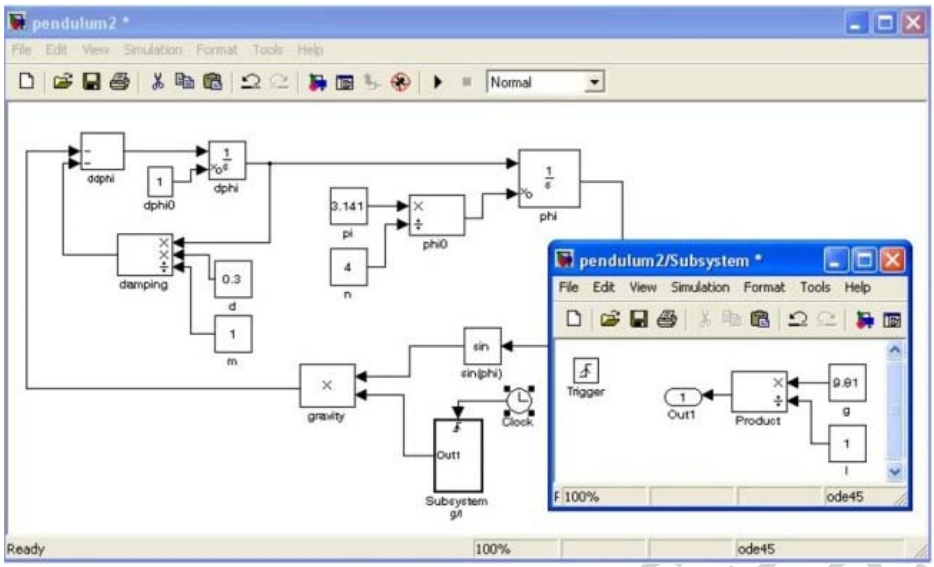

Figure 2: Graphical model description of pendulum in SIMULINK

3.2. Requirements for the Experimental Frame

In principle, the experimental frame has to set parameters, it has to control and perform the "simulation" of the model (the solution of the system governing ODEs), and it should support documentation of the results. In the CSSL standard, minimal requirements for runtime environment are given as follows:

- availability of certain ODE Solvers (Euler, RK4, RK-Fehlberg, and Gear or BDF algorithms for stiff systems
- possibility for change of parameters, eventually also calculations with parameters,
- and documentation of results in a plotting system

From 1968 on all simulation languages tried to meet the CSSL standard. But the implementations were and are different. First, the structure with sections or regions can be given explicitly by definition of this section, semi-implicitly by type declarations of all variables, and full implicitly by recognition of the translator (which must run at last a two-pass parser). For instance, SIMNON defines parameters and states, so that for sorting the section structure is determined; for the pendulum example, these definitions are:

```
    parameter m, l,
g, …….
    state     phi,
phidot
    derivative
dphi, ddphi
```

These definitions determine the structure of the equations, but they also show a (minor) problem: a state cannot be a derivative, and vice versa; in principle, `dphi` is both, so it has to be renamed, e.g. as state with `phidot`.

There are many different ways to implement a simulation system. In a pure interpreter

implementation, first the runtime system – an interpreter – recognizes the commands, and starts the simulation engine - in most cases the start of the simulation run in the time domain. The engine (sometimes compiled, sometimes working in interpreter mode) calls an interpreter algorithm for solving the ODE, which itself has to calculate the derivative vector by interpretation of the sorted equations. In a pure compiled implementation, first a translator transforms the model description into a host language resulting in a function call for the derivative vector (and for additional functions). A compiled library system (with simulation engine, with ODE solvers, with parameter change facilities, and with a plotting system) is linked to this compiled model, which then can be handled from a compiled runtime environment. One meets various mixtures of compiled and interpreted implementations, as well as on the one side really strict distinctions of model frame and experimental frame, and on the other hand definitions of model frame and experimental frame in a common deck for compilation. Clearly, there are advantages and disadvantages in all kinds of implementation. Generally holds: the more compiled, the faster, but the more inflexible for changes, and the more interpreted, the slower, but the more flexible for changes. In addition to modeling features, CSSL also suggests statements for selecting integration routines and their parameters, for controlling the simulation, and for documentation of results.

4. Numerical Algorithms in Simulation Systems

With the first digital computers the development of numerical software started. This software was implemented in libraries, written usually in FORTRAN. The first numerical libraries concentrated on linear analysis and linear algebra, like the BLAS library. The Basic Linear Algebra System was mainly used in control applications, and in applied mathematics. Soon libraries with other algorithms became available, amongst them algorithms for solving ODE solving. Most of these algorithms replace the differential quotient by difference quotients. Euler's 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. Automatic step length adjustment was another important contribution.

Systems with both fast and slow modes (stiff systems) posed a particular difficulty for explicit ODE solvers. It is necessary to choose a very short step length to have numerical stability, which gives a very slow simulation and results in numerical errors for the slow model parts. Here implicit integration schemes offer a help, allowing a bigger stepsize. Numerical solvers for this purpose appeared in 1970. It turned out, that implicit solvers can also handle implicit model equations and model equations with algebraic constraints, so that the conquest for the best DAE (Differential-Algebraic Equations) solvers started (for comments on implementation see section on implicit model descriptions). Implicit model equations very often occur e.g. in modeling robots (if inversion of the mass matrix is inappropriate or impossible).Nowadays, FORTRAN has lost dominance, user find libraries with ODE and DAE solvers in C, C++, Java, etc. Standardized libraries like IMSL and NAG library offer more then one host language. The kernel of a simulation system makes use of these ODE and DAE solvers.

5. Simulation Software and CACSD Tools

The fore-mentioned BLAS library was also the basis of another type of software, of the so-called CACSD tools (**C**omputer-**A**ided **C**ontrol **S**ystem **D**esign). The most prominent tool is the MATLAB system with all its toolboxes. MATLAB itself offers ODE solvers - to be called from the MATLAB environment - , which frequently must call the evaluation of the derivative vector – to be supplied as function in an m-file. Here MATLAB does not fulfill the requirements for a simulator, because the model description, the m-file with the derivative vector, follows classic programming rules (no sorting, fixed state space, etc.) – consequently one cannot call MATLAB a simulator. But after the first MATLAB versions a special toolbox for simulation was released, the SIMULINK system (previously called SIMULAB). SIMULINK is a model frame for graphical model representation, which on the one side offers a simple experimental frame by pull-down menus in the model layout, but which on the other side is driven by MATLAB, the up to now most powerful experimental frame. SIMULINK can be seen as simulator, with graphical model frame and a small experimental frame consisting of pop-down menus. SIMULINK / MATLAB can be seen as powerful simulator with the graphical model frame SIMULINK (without use of the pop-down menus for experimentation), and with the very powerful experimental frame MATLAB (without direct MATLAB calls to ODE solvers).

In general, graphical model representations of the SIMULINK type shown in Figure 2 were used to represent models in terms of integrators, adders and potentiometers in the early days of analog simulation and to represent control loops and transfer functions. However, graphical modeling was not widely used until modern work stations and the PC with raster graphics became generally available. Boeing's simulator EASY5 from 1976 was provided with a graphical user interface. Not only MATLAB, but also the other matrix environments $MATRIX_X$ and CTRL_C were provided with graphical modeling tools: SystemBuild, integrated in MATRIXX, and SYSTEM-C integrated in CTRL-C appeared in 1984, whereby the first versions made use of the graphical system GEM, and not of the WINDOWS environment (at that time really GEM was much more stable than WINDOS 1.x). MATLAB started SIMULINK in 1991, implemented in WINDOWS in 1991, interestingly first called SIMULAB.

These three CACSD systems became simulation systems by means of the graphical model frames with weak structures. But neither MATLAB nor MATRIXX and CNTRL-C offer better structured textual model frames, so that up to now MATLAB / SIMULINK is a simulator with relatively weakly structured model frame, but with the most powerful experimental frame, with MATLAB itself (from MATLAB's point of view simulation is only one of the analysis methods offered). The MATLAB / SIMULINK system dominates clearly the market for continuous simulation, many simulators of the 1980s and 1990s have vanished, and competitors could only survive by concentrating on special areas, e.g. special model frames. But 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. A severe consequence is that it is cumbersome to build physics based model libraries in the block diagram languages. Despite all of these disadvantages in SIMULINK, the MATLAB / SIMULINK system became the most used simulator –

because of the very powerful experimental frame MATLAB (see also next chapter). Indeed, up to now the experimental frames of many simulators must be called very poor.

-
-
-

Bibliography

Ascher U.M. and Petzold L. R. (1999). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia: SIAM. [Classical introduction to numerical algorithms for solving differential equations]

Banks J., (Ed). (1998): *Handbook of Simulation*. New York: John Wiley and Sons. [State-of-the-art handbook for discrete modeling and simulation]

Bergin Jr. T. J. and Gibson R. G. (Eds) (1996): *History of Programming Languages*. New York: ACM Press. [Nice overview on history of programming languages]

Breitenecker F. (1992).Models, methods and experiments - A new structure for simulation systems. Mathematics and Computers in Simulation 34, 1 – 30. Amsterdam: North Holland. [Introduction to Model-Method-Experiment - concept]

Breitenecker F. and Husinsky I. (Eds.) (1992 - now). SNE – Simulation News Europe, Vienna: ARGESIM Publisher. [This journal publishes the ARGESIM Comparisons on Simulation Techniques and Simulation Tools and reports about European Projects in modeling and Simulation]

Brenan, K. E., S. L. Campbell, and L. R. Petzold (1989). *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Amsterdam: North-Holland. [This is a classical survey on algorithms for solving differential-algebraic equations]

Cellier F. E. (1991): *Continuous System Modeling*. New York: Springer. [Up to now a state-of-the-art summary about continuous system modeling]

Forrester J. W. (1961). *Industrial Dynamics*. Cambridge, MA: M.I.T. Press. [The book on System Dynamics]

Hairer E., Norsett S., and Wanner G. (1987). Solving Ordinary Differential Equations I - Nonstiff Problems. Computational Mathematics No. 8. Berlin: Springer. [This book introduces into classical ODE solvers]

Hairer E. and Wanner G. (1991). Solving Ordinary Differential Equations II - Stiff and Differential-Algebraic Problems. Computational Mathematics No. 14. Berlin: Springer. [This book introduces into stiff DAE solvers]

Hellekalek P. (1998). Good random number generators are (not so) easy to find. Mathematics and Computers in Simulation 46, 485-505. Amsterdam: Elsevier. [Informative survey paper on random number generators]

Jackson A. S. (1960). *Analog Computation*. New York: McGraw-Hill. [Standard book on analog computation, worth reading dut to historical development]

Jansson B. (1966). *Random Number Generators*. Stockholm:Victor Pettersons. [Book on first developments of pseudo random number generators]

Karnopp D. C. and Rosenberg R. C. (1968): *Analysis and Simulation of Multiport Systems - The Bond Graph Approach to Physical System Dynamics*. Cambridge, MA: MIT Press. [The authors, known as fathers of bond graphs, introduce into the multiport systems concept, or bond graph concept, resp.]

Kleijnen J. P. C. (1975*). Statistical Techniques in Simulation*. New York: Marcel Dekker. [This book introduces into correct use of statistical techniques in discrete simulation]

L'Ecuyer P. (1998). Random number generation. In (J. Banks, ed.) Handbook on Simulation, 93 – 137. New York: John Wiley. [Good survey paper on random number generators by one of the fathers of this subject]

Sargent R. G. (1988). Event graph modeling for simulation with an application to flexible manufacturing systems. Management Science. 34(10) 1231 - 1251. Elkridge, MD: INFORMS. [Overview paper for application of event graphs in automation]

Schmidt B. (2001). *The Art of Modeling and Simulation: Introduction into the Simulation System SIMPLEX-3*. Erlangen: SCS European Publishing House. [Introduction into continuous, discrete and hybrid simulation using the simulator SIMPLEX]

Schriber T. J. (1991). *An Introduction to Simulation Using GPSS/H*. New York: John Wiley & Sons. [Introduction into discrete simulation using the simulator GPSS/H]

Solar D. and Breitenecker F. (1988): The Simulation System HYBSYS. In Proc.European Simulation Multiconferences ESM88, 312 – 318. San Diego: SCS Publishing. [Introduces a hybrid simulators based on the Model-method-Experiment - concept]

Strauss J. C. (1967) The SCi continuous system simulation language (CSSL). Simulation 9, 281-303. San Diego: SCS Publishing. [Paper defining the CSSL standard for simulation languages]

Zeigler B.P. (1976). *Theory of Modeling and Simulation*. New York: John Wiley & Sons. [This book is a standard book for discrete simulation, introducing the concept of model frame and experimental frame]

Biographical Sketches

Felix Breitenecker studied "Technical Mathematics" at Vienna University of Technology (VUT), finishing with Master of Technical Sciences (Dipl.-Ing.) in 1976 and graduated (Dr.techn.) with a thesis in Mathematics of Control.in 1979. From 1976 on he worked as research assistant at the VUT Mathematics of Control and Simulation Group and from 1984 on, as associate professor at VUT for "Simulation and Mathematics of Control" doing education and RD in modeling and simulation. He was guest professor at University Glasgow, at Univ. Clausthal-Zellerfeld, Univ. Ljubljana,. and Univ. Linz.

He is active in various modeling and simulation societies: president and past president of EUROSIM since 1992, board member and president of the German Simulation Society ASIM, member of INFORMS, SCS, UKSIM, etc. In 2004 he has been elected into the executive board of GI, the German Gesellschaft für Informatik. In 2001 he received as first European, the Distinguished Service Award of INFORMS, the OR Society of USA.

He has organized and co-organized the European Simulation Congress Vienna (1995), ASIM conferences in Vienna, the conference series MATHMOD in Vienna, and Simulation Workshops in UK, Germany and Austria.

Felix Breitenecker covers a relatively broad research area, from mathematical modeling to simulator development, from discrete event simulation to symbolic computation, from numerical mathematics to object-oriented simulation implementation, from biomedical and mechanical simulation to workflow and process simulation.

He is involved in various national and international research projects and he is active in industry projects, e.g. with Daimler-Chrysler and EADS. In co-operation with ARCs, the Austrian Research Center Seibersdorf, he takes part on research and industry projects in biomedical engineering and in process engineering.

He has published about 210 scientific publications, and he is author of two 2 books and editor of 16 books (Proceedings and Series). Since 1992 he is editing the journal Simulation News Europe, Editor in Chief since 1995, and he is co-editor of the SCS Series "Frontiers in Simulation" and "Advances in Simulation".

Inge Troch graduated (Dr.techn.) with a thesis in mathematics of control at the Vienna University of Technology. She is University Professor since 1974 and initiated courses and scientific work in Mathematics of Control, Modeling and Simulation at TU-Vienna. Her teaching comprises courses in these

areas as well as on basic mathematics for engineering students and on differential equations at TU-Vienna, courses in robotics at the Universities of Linz and Bologna and for the Scientific Academy of Lower Austria at Krems. At present she is head of the Institute for Analysis and Scientific Computing at TU-Vienna.

She is Austrian delegate in IFAC Technical Committees (TCs) 'Optimal Control' and 'Linear Systems' and was chairperson for a four year term (and is still active member) of the VDI/VDE-GMA Committee on 'Modeling and Simulation in Automation' in Germany. She is chairperson of the IMACS-TC on 'Mathematical Modeling' and organizes successfully a series of triennial conferences on Mathematical Modeling (MATHMOD Vienna). She is senior member of IEEE.

Inge Troch is Editor-in-Chief of the *journal Mathematical and Computer Modeling of Dynamical Systems* and a member of the international editorial board of *Mathematics and Computers in Simulation* (MATCOM), *Systems Analysis, Modelling and Simulation*, *J. Intelligent and Robotic Systems* (JIRS) and *Surveys on Mathematics in Industry*). She was member of the editorial board of C-TAT (*Control -- Theory and Advanced Technology*), and was/is also a member of the IPC and/or session organizer of some 60 international symposia and congresses.

Inge Troch is a co-author of two books, co-editor of nine Proceedings, editor of several special issues of scientific journals, author or co-author of about 120 articles in scientific journals and books in the fields of mathematics of control (stability, systems theory, optimization), modeling, simulation and robotics.