Simulation in Engineering Sciences
J. Burger and Y. Jarny (eds.)
Elsevier Science Publishers B.V. (North-Holland)

SIMULATION SOFTWARE: TODAY AND TOMORROW

Francois E. Cellier

Institute for Automatic Control
The Swiss Federal Institute of Technology Zurich
ETH - Zentrum
CH-8092 Zurich
Switzerland

This paper describes briefly the current situation on the simulation software market. A list of simulation software features is presented which is then graphed in tabular form versus a couple of current simulation languages and packages. In a second part, some of the major shortcomings of current simulation systems are outlined, and some prospectives for development are given.

1. INTRODUCTION

Eight years ago I have been asked already once to survey the numerical techniques used in continuous simulation together with the major software systems which existed at that time for this purpose [8]. When I was now asked once more to repeat this task, I tried to figure out whether our knowledge about simulation techniques has sufficiently advanced over the past eight years to justify a reconsideration. I then came to the conclusion that most of the "prospectives for development" mentioned in that paper had meanwhile become everydays state-of-the-art issues, while most of the software systems considered at that time are meanwhile obsolete. Moreover, I have some new ideas about future development of simulation software which were not present yet in 1975. Therefore, I considered the time come to write another survey now.

In this paper, I shall not review the basic features (such as numerical integration) dealt with in my previous survey [8]. I shall assume that the reader of this article has already acquired a basic knowledge of the functioning of simulation software. Moreover, I shall extend my view to discrete simulation as well, as it was realized in the mean time that the techniques used in these two classes of simulation systems are very much related to each other, and as there exists now a considerable number of software systems capable of performing combined continuous and discrete simulation.

There are meanwhile so many simulation software systems on the market that it has become impossible to review even the major ones within a reasonably limited number of pages. For this reason, the mentioned software systems (which are those that I know best) are meant to be respresentative for many other software systems showing only minor differences.

This paper shall basically consist of three parts. In a first section, I shall try to present a crude clustering of simulation software systems based on a selection of classifying features. In a second part, I shall then list a few simulation systems which are either available on the market now or which are currently under development. The paper shall then be concluded with a list of proposals how some of the more important shortcomings of the available systems might be overcome in the future.

2. SIMULATION SOFTWARE FEATURES

In the following table, a rather large set of characterizing features of nowadays simulation software systems are listed. Altogether 15 software systems have been analysed with respect to the availability of these features. If a feature is not available in a system, this is indicated by (-). If the feature is available, this is marked by (x). In many cases however, the implementation of a feature being present in several software systems differs with respect to the degree of sophistication (e.g. comfort of usage). In such cases, the better implementation is indicated by (xx). Sometimes, a foot note is added to explain the difference. Please note that a (-) does not necessarily indicate that the respective feature is not programmable in that language. It just means that no particular provision is taken by the language for that purpose. To cite an example: it is of course possible to receive a histogram by use of SIMULA (as SIMULA is a flexible general purpose programming language). Still, the respective box in the table is marked by (-) as no particular provision is taken by SIMULA to relieve the user from writing his own program to collect statistics and get a histogram printed.

Features: \ Languages:	ACSL	DARE-P	SIMNON	DYMOLA	SYSMOD	FORSIM-6	SIMULA	PROSIM	SIMSCRIPT	GPSS-F-2	GPSS-F-3	SLAM-2	GASP-5	GASP-6	COSY
EXPRESSIVENESS OF THE LANGUAGE															
Continuous Systems															
Ordinary Differential Equations	xx	x	x	xx	xx	x	-	x	-	-	x	x	x	x	xx
Partial Differential Equations	x1	x2	-	-	-	xx	-	-	-	-	-	-	x3	xx	xx
Difference Equations	x	-	xx4	-	x	-	x	x	x	x	x	x	x	x	x
Discrete Systems															
Event Handling	x	-	-	-	x	-	x	x	xx	xx	xx	xx	xx	xx	xx
Process Interaction	-	-	-	-	-	-	xx	xx	-	xx	xx	xx	-	xx	xx
Activity Scanning	-	-	-	-	-	-	xx	xx	-	xx	xx	xx	-	xx	xx
Symbolic Library (at Source Level)	xx	-	-	xx	xx	-	-	-	-	-	-	-	-	-	xx
Run-Time Library															
for Continuous Systems	xx	x	x	x	xx	x	-	x	-	-	x	x	xx	xx	xx
for Discrete Systems	-	-	-	-	x	-	x	x	x	xx	xx	xx	xx	xx	xx
Data Handling (Data-Base Management)	-	x	-	-	x	-	-	-	-	-	x	xx5	x	x	x
NUMERICAL BEHAVIOR															
Integration															
Library of Integration Methodes	x	xx	x	x	x	x	-	-	-	-	x	-	x	x	x
Own Integration Method	-	xx	-	-	-	-	-	-	-	-	x	-	x	x	x
Automatic Selection of Method	-	-	-	-	-	-	-	-	-	-	-	-	x6	x6	x6
Partitioning of State-Space															
Manual Partitioning	xx	-	x	x	xx	-	-	-	-	-	x	-	-	-	-
Automatic Partitioning	-	-	-	-	-	xx7	-	-	-	-	-	-	-	-	-
Algebraic Loop Solver	xx	-	-	-	xx	xx	-	-	-	-	-	-	x	x	xx
Root Solver (State Events)	x	-	-	-	xx	x	-	-	-	-	x	x	xx	xx	xx
Steady-State Solver	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Tracking Problems	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Integral-Differential Equations	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sparse Linear System Solver	-	-	-	-	-	x	-	-	-	-	-	-	-	x	x
Stiff Systems	xx	xx	x	x	xx	xx	-	-	-	-	x	-	xx	xx	xx
Highly Oscillatory Systems	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Linear Systems (Special Method)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Noisy Systems	x	x	x	-	x	x	-	x	-	-	x	x	x	x	x
Partial Differential Equations (PDE)	x	x	-	-	-	xx	-	-	-	-	-	-	x	xx	xx
Spatial Derivatives Computation															
One-Dimensional Derivatives	-	x	-	-	-	xx	-	-	-	-	-	-	xx	xx	xx
Two-Dimensional Derivatives	-	-	-	-	-	xx	-	-	-	-	-	-	-	xx	xx

1) Formulation through vector-integration and interpretive macros.
2) By use of the REPEAT-operator.
3) As GASP-V is restricted to 100 ODE's, only PDE's in one space dimension can be formulated.
4) Special operator. All other systems treat difference equations by means of time events.
5) By use of a special simulation data language (SDL) which is available through Pritsker & Assoc.
6) Provided to date only in an experimental version which is not yet made available.
7) Not in the public version of FORSIM-VI, but well documented, tested and available from AECL.

Features: \ Languages:	A	D	S	D	S	F	S	P	S	G	G	S	G	G	C
	C	A	I	Y	Y	O	I	R	I	P	P	L	A	A	O
	S	R	M	M	S	R	M	O	M	S	S	A	S	S	S
	L	E	N	O	M	S	U	S	S	S	S	M	P	P	Y
		-	O	L	O	I	L	I	I	C	-	-	-	-	
		P	N	A	D	M	A	M	R	F	F	2	5	6	
						-			I	-	-				
						6			P	2	3				
									T						
Three-Dimensional Derivatives	-	-	-	-	-	xx	-	-	-	-	-	-	-	xx	xx
Library of Methods	-	-	-	-	-	xx	-	-	-	-	-	-	x	xx	xx
Spline Interpolation	-	-	-	-	-	x	-	-	-	-	-	-	x	x	x
Up-wind Interpolation	-	-	-	-	-	x	-	-	-	-	-	-	x	x	x
Own Interpolation Method	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Automatic Selection of Method	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Variable Grid Width	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Automatic Grid Width Control	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Error Estimation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Parabolic PDE's	x	x	-	-	-	xx	-	-	-	-	-	-	x	xx	xx
Hyperbolic PDE's	x	x	-	-	-	x	-	-	-	-	-	-	x	x	x
Ellyptic PDE's	x	-	-	-	-	x	-	-	-	-	-	-	x	x	x
Shock Waves	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Mixed PDE's and ODE's	x	x	-	-	-	xx	-	-	-	-	-	-	xx	xx	xx
Statistical Analysis															
Random Number Generation	x	x	-	-	x	x	-	x	x	xx	x	xx	xx	xx	xx
Distribution Function Library	-	-	-	-	-	-	-	x	-	xx	x	x	xx	xx	xx
Tabular Distribution Functions	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x
Distr. Funct. Parameter Fitting	-	-	-	-	-	-	-	-	-	-	-	-	xx1	-	-
Statistical Report Generation															
Sampling Statistics	-	-	-	-	-	-	-	x	xx	xx	xx	xx	xx	xx	xx
Time-Persistant Statistics	-	-	-	-	-	-	-	-	xx	xx	xx	xx	xx	xx	xx
Histograms	-	-	-	-	-	-	-	-	xx	-	xx	xx	xx	xx	xx
Confidence Intervals	-	-	-	-	-	-	-	-	-	x	x	xx1	-	-	-
Variance Estimation															
Variance of the Mean	-	-	-	-	-	-	-	-	-	x	x	x	x	x	x
Run-Length Determination	-	-	-	-	-	-	-	-	-	x	x	-	-	-	-
Transient Period Duration	-	-	-	-	-	-	-	-	-	x	x	-	-	-	-
Significance Tests	-	-	-	-	-	-	-	-	-	x	x	xx1	-	-	-
Variance Reduction															
Replication and Batch	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Subinterval Analysis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sensitivity Analysis															
Linear Approximation (Metamodel)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Replication	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Table Look-up															
Two-Dimensional Tables	x	x	-	-	x	x	-	-	-	-	-	x	x	x	x
Three-Dimensional Tables	x	x	-	-	-	x	-	-	-	-	-	-	-	x	x
Linear Interpolation	x	x	-	-	x	x	-	-	-	-	-	x	x	x	x
Non-linear Interpolation	-	-	-	-	x	-	-	-	-	-	-	-	-	x	x
Spline Interpolation	-	-	-	-	x	-	-	-	-	-	-	-	-	x	x
Dynamic Table Load	-	-	-	-	-	-	-	-	-	-	-	x	x	x	x
Sequential Interpolation	-	-	-	-	x	-	-	-	-	-	-	-	-	x	x
Mass-Storage Interpolation	-	-	-	-	x	-	-	-	-	-	-	-	-	x	x

1) By use of a separate program (AID) also provided by Pritsker & Assoc.

Features: \ Languages:	ACSL	DARE-P	SIMNON	DYMOLA	SYSMOD	FORSIM-6	SIMULA	PROSIM	SIMSCRIPT	GPSS-F-2	GPSS-F-3	SLAM-2	GASP-5	GASP-6	COSY
STRUCTURAL FEATURES															
Application Program Development															
Model Structuring Capabilities															
Parallel Structures															
Continuous Systems (Sorting)	x	x	x	xx	x	-	-	-	-	-	-	-	-	-	xx
Discrete Systems (Networks)	-	-	-	-	-	-	-	-	-	x	x	xx	-	-	-
Procedural Structures															
Continuous Systems (Nosort)	xx	x	-	-	xx	x	-	x	-	-	x	x	x	x	xx
Discrete Systems (Algorithmic)	x	-	-	-	x	-	xx	xx	x	x	x	x	x	x	xx
Event Handling															
Time Events	x	-	-	-	xx	-	x	x	xx	xx	xx	xx	xx	xx	xx
State Events	x	-	-	-	xx	-	-	x	-	-	xx	xx	xx	xx	xx
External Events															
Operator Intervention	-	-	xx	xx	xx	-	-	-	-	-	-	-	-	-	-
Real-Time Interrupts	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Process Interaction															
Continuous Processes	x	-	xx	xx	xx	-	-	-	-	-	x	-	-	xx	xx
Discrete Processes	-	-	-	-	-	-	xx	xx	-	xx	xx	xx	-	xx	xx
Network Description	-	-	-	-	-	-	-	-	-	x	x	xx	-	-	-
Activity Scanning	-	-	-	-	-	-	xx	xx	-	xx	xx	x	-	xx	xx
Submodel Definition															
Continuous Submodels	x	-	xx	xx	xx	-	-	-	-	-	x	-	-	x	xx
Discrete Submodels	-	-	-	-	-	-	xx	xx	-	xx	xx	-	-	xx	xx
Hierarchical Model Definition															
for Continuous Systems	-	-	-	xx	xx	-	-	-	-	-	-	-	-	-	xx
for Discrete Systems	-	-	-	-	-	-	xx	xx	-	-	-	-	-	-	xx
Initial Computations	x	x	x	-	x	x	-	-	x	x	x	x	x	x	x
Terminal Computations	x	x	x	-	x	x	-	-	x	x	x	x	x	x	x
Controlled Experiments	x	xx	xx	xx	xx	-	-	-	x	-	-	xx	xx	xx	xx
Optimization	-	-	-	-	-	-	-	-	-	-	-	-	-	-	xx[1]
Parameter Fitting (provided)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	xx[1]
Modularity															
Macro Feature	xx	-	-	-	xx	-	-	-	-	-	-	-	-	-	xx
Interpretive Macros	xx	-	-	-	xx	-	-	-	-	-	-	-	-	-	xx
Module Feature	-	-	-	xx	-	-	-	-	-	-	-	-	-	-	xx
Graphical Model Specification	-	-	-	-	-	-	-	-	-	-	-	xx[2]	-	-	-
Program Validation and Verification															
Model Comparison	-	x	x	-	xx	-	-	-	-	-	-	xx[3]	x	x	x
Sensitivity Analysis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Linearized Model Analysis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Parameter Fitting (possible)	x	xx	xx	xx	xx	-	-	-	-	-	-	xx	xx	xx	xx
Eigenvalue- Eigenvector Analysis	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Debugging Aids															
Dimensional Analysis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Declaration of Variables	-	-	x	xx	xx	-	xx	xx	x	-	-	-	-	-	xx
Steady-State Finding	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Graphical Model Representation	-	-	-	-	-	-	-	-	-	-	-	xx[2]	-	-	-

1) By use of a separate run-time library (NLP) also available from ETH Zurich.
2) In an experimental version which is currently under development by Pritsker & Assoc.
3) By use of a special simulation data language (SDL) provided by Pritsker & Assoc.

Features: \ Languages:	ACSL	DARE-P	SIMNON	DYMOLA	SYSMOD	FORSIM-6	SIMULA	PROSIM	SIMSCRIPT	GPSS-F-2	GPSS-F-3	SLAM-2	GASP-5	GASP-6	COSY
Tracing															
Event Monitoring	-	-	-	-	xx	-	-	xx	x	x	xx	xx	x	xx	xx
Range Surveillance	-	-	-	-	-	-	-	-	-	-	-	-	-	-	xx
Deadlock Detection	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Range Analysis	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Program Execution															
Batch Processing	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Interactive Operation															
Parameter Setting	x	x1	x	x	x	-	-	-	-	-	-	-	-	-	-
Parameter Tuning	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Output Selection	x	x1	x	x	x	-	-	-	-	-	-	-	-	-	-
Run-Time Display	-	x1	-	-	-	-	-	-	-	-	-	-	-	-	-
Operator Intervention	-	-	x	x	x	-	-	-	-	-	-	-	-	-	-
Execution of Monitors	-	-	-	-	x	-	-	-	-	-	-	-	-	-	-
Real-Time Synchronization	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Data Handling															
Library of Parametric Models	-	-	-	-	x	-	-	-	-	-	-	-	-	-	xx
Library of Parameter Sets	x	x	x	x	x	-	-	-	-	-	-	-	-	-	-
Library of Experimental Frames	x	-	xx	xx	xx	-	-	-	-	-	-	-	-	-	x
Library of Trajectories	-	xx	x	x	x	-	-	-	-	-	-	xx2	xx	xx	xx
Numerical Manipulations	-	-	-	-	-	-	-	-	-	-	-	xx2	-	-	-
Structural Manipulations	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Input/Output															
Model Parameters	x	x	x	x	xx	x	-	-	-	x	x	-	-	-	xx
Driving Functions															
Tabular Functions	-	-	x	-	xx	-	-	-	-	-	-	-	-	-	xx
Real-Time Input	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Graphical Model Representation	-	-	-	-	-	-	-	-	-	-	-	xx3	-	-	-
Graphical Model Specification	-	-	-	-	-	-	-	-	-	-	-	xx3	-	-	-
Storing Output Data (on Files)	-	xx	x	x	xx	-	-	-	-	x	xx	xx2	xx	xx	xx
Retrieving Input Data (from Files)	-	-	x	-	xx	-	-	-	-	-	-	-	-	xx	xx
Crossplots	x	x	x	x	x	-	-	-	-	-	-	xx4	x	x	x
Overplots	x	xx	x	x	xx	-	-	-	-	-	x	xx4	xx	xx	xx
Three-Dimensional Plots	-	x5	-	-	-	x6	-	-	-	-	-	-	x	x	x
Histograms	-	-	-	-	-	-	-	x	-	-	xx7	xx8	x	x	x
Bar Charts	-	-	-	-	-	-	-	-	-	-	-	xx4	-	-	-
Pie Graphs	-	-	-	-	-	-	-	-	-	-	-	xx4	-	-	-

1) Not in the official version provided by the University of Arizona. There exist however several interactive versions of DARE-P, one VAX version available from ETH Zurich, and one portable (but less interactive) version from the University of Arizona.
2) By use of a special simulation data language (SDL) provided by Pritsker & Assoc.
3) In an experimental vesion which is currently under development by Pritsker & Assoc.
4) By use of a separate program (SIMCHART) also provided by Pritsker & Assoc.
5) Not in the official version of the DARE-P postprocessor as provided by the University of Arizona. An enhanced postprocessor is available from ETH Zurich.
6) Only provided in an internal version from the Atomic Energy of Canada, Ltd. but not in the version handed out to external users (bases upon a local graphics software package).
7) Quality plots base on the Erlanger graphics system. Not provided for external use.
8) Quality histograms only in connection with the separate program AID from Pritsker & Assoc.

Features: \ Languages:	ACSL	DARE-P	SIMNON	DYMOLA	SYSMOD	FORSIM-6	SIMULA	PROSIM	SIMSCRIPT	GPSS-F-2	GPSS-F-3	SLAM-2	GASP-5	GASP-6	COSY
Output Quality															
Lineprinter Plot	x	x	x	x	x	x	-	x	-	x	x	x	x	x	x
Plotter output monochrome	x	x	x	x	x	x1	-	-	-	-	xx2	x3	x	x	x
Color Graphics	-	-	x	x	-	-	-	-	-	-	-	-	-	-	-
STATUS OF IMPLEMENTATION															
Compiler	x	x	x	x	-	-4	x	x	x	-4	-4	x	-4	-4	-
Run-Time System	x	x	x	-5	-	x	x	x	x	x	-6	x	x	-7	-8
Environment	-	x	x	x	-	-	-	-	-	-	-	-	-	-	-
PORTABILITY	x	xx	-	x	xx	-	-	-	-	x	x	xx	xx	xx	xx
DOCUMENTATION	xx	x	x	xx	-	x	xx	x	xx	xx	-6	xx	x	-	-
Machine Readable	-	-	-	x	-	-	-	-	-	-	-	-	-	x	x
On-Line HELP Information	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-

1) Only provided in an internal version of the Atomic Energy of Canada, Ltd. but not in the version which is handed out to others (bases upon a local graphics package).
2) Quality plots base on the Erlanger graphics system. Not provided for external use.
3) By use of a separate program (SIMCHART) provided by Pritsker & Assoc.
4) These are FORTRAN packages. A compiler is therefore not required.
5) The run-time system of DYMOLA is SIMNON. A special run-time system is therefore not required.
6) The run-time system exists already at the University of Erlangen, but has not yet been released for external use. An official release is planned for late 1983 as soon as the complete documentation becomes ready.
7) A preliminary version of GASP-VI exists at ETH. This version is however not yet sufficiently debugged and consolidated to allow for external distribution.
8) The run-time system of COSY is GASP-VI. A special run-time system is therefore not required.

A first set of characteristics describes general features, e.g. whether a system is meant for purely discrete simulation (like SIMULA) or for purely continuous simulation (like DARE-P). This group is headed "expressiveness of the language".

Continuous systems may be described by several formalisms, ordinary differential equations (ODE's), partial differential equations (PDE's), or possibly difference equations. Please, note that systems described by difference equations are called continuous here, although many people (e.g. most control engineers) would call them discrete. This is due to the fact, that the simulation methodology required for the solution of difference equation systems is much closer related to the continuous methodology than to the "real" discrete methodology, a term which I reserve for event oriented systems.

Discrete systems may either provide for a (primitive) event description only which may result in rather unreadable models when the systems are large, or for a process interaction mechanism. Activity scanning means that the duration of an activity (e.g. service time) may depend on some other parts in the system, e.g. the completion of another service performed on another transaction at the same time.

Symbolic libraries are libraries at source level, e.g. macro libraries for the description of submodels. Such source libraries provide in some sense for an open ended operator set of the language.

Data handling describes in global terms the way in which the simulation data are managed by the simulation system. I shall not further emphasis on this point now, as this shall become a central aspect of my chapter on future developments.

The next section is headed "numerical behavior", and describes many aspects of how the simulation software treats a model numerically. Four major topics here concern the numerical integration of differential equations, computation of spatial derivatives (for PDE's), statistical analysis, and table look-up.

Almost any system able to cope with ODE's provides nowadays for a library of integration algorithms. This is important as there does not exist any integration algorithm which is equally well suited for all types of ODE problems. In particular, methods for the solution of stiff systems are quite common in current simulation software. SLAM-II (which is otherwise a strong piece of software) is particularly weak in this respect as it offers just one Runge-Kutta algorithm for integration. Many continuous systems can therefore not be successfully treated by use of SLAM. By far the best library is present in DARE-P. This software offers more than a dozen different integration algorithms both in single and double precision. A special conversion routine allows to convert single precision to double precision and vice-versa, such that only one version of each algorithm needs to be stored at any time.

However, many users are demanded too much by a rich selection of routines. It would be highly welcome, if the simulation software could find out automatically which routine is best to apply in each case, and load that routine from the library. First steps towards an achievement of this goal are implemented in an experimental version of GASP-V [10]. A first commercially available integration routine which is able to switch between a stiff and a non-stiff algorithm has been described by Petzold and Hindmarsh [21]. However, this algorithm has to my knowledge not yet been implemented in any ready-to-use simulation software.

For some problems, it may be beneficial to split them up into submodels which are integrated independently by either a different step size or even a different integration algorithm. This feature is called "partitioning of the state-space". Some software systems (such as ACSL) allow for a definition of submodels, whereby each submodel may be integrated by use of a separate integration routine. Communication between submodels takes place after each communication interval only, that is: the variables of each subsystem are considered constant during each communication interval by each other subsystem. This is not necessarily optimal as: (a) the so introduced discontinuities may create serious numerical stability problems; (b) the so introduced artificial sampling may introduce analytical stability problems; and (c) the model splitting which is optimal for model structuring and readability is not necessarily also optimal for numerical integration. Some other systems (such as SIMNON) try to overcome these problems by letting the user separately specify submodels (which exist only at source level but no longer during run-time) from groups of state variables which are to be integrated together (slow subsystem versus fast subsystem). The integration method offered for that purpose in SIMNON is somewhat obscure. It works neatly with some adaptive control systems where it is quite clear that the inner control loop consists of considerably faster modi than the outer adaptation loop, but in other cases the method tends to fail. Another such algorithm has been described by Eitelberg [14]. A third (and rather promising) approach is taken by Carver who developed an automated partitioning scheme which is rather simple to use and even works well on non-linear problems [7].

Algebraic loop solvers are quite common in current simulation software. However, their use tends to be costly, as an iteration has to take place during each function evaluation. For that reason, it may be advantageous to apply an integration routine which solves the algebraic equations at once during integration. Such a code has e.g. been described by Petzold [34]. This approach shall lead away from the currently used state-space description of ODE's (that is: $x'=f(x,t)$) to the more general form: $f(x,x',t)=0$.

Root solvers are required for the location of state events, and are essential algorithms for combined continuous and discrete simulation.

Steady-state solvers allow for the computation of the steady-state at less computational cost than by simply integrating there. Such algorithms are rare. ACSL offers something, but the offered algorithm fails often in non-linear cases.

Tracking problems ask for "freezing" of state variables during simulation. This feature is currently only provided for by ACSL.

Integro-differential equations are numerically harmful, and there is no system which is currently able to deal with them successfully. Typical examples include the flow through a pipe with variable delay. Some systems (like DARE-P) claim to offer a solution which, however, mostly fails when applied.

A sparse linear system solver is very much needed for efficient integration of high dimensional problems (e.g. PDE's). The economization there may be dramatic. Still, most simulation systems do not offer this feature, though it is extremely simple to implement.

A stiff system solver is nowadays very commonly found. However, it is rarely realized that not all "difficult" problems are really "stiff". A very good discussion of this point may be found from Gear [18]. One other class of difficult problems to solve is the class of "highly

oscillatory problems". Possible solutions may be by means of stroboscopic integration methods which use a low order integration scheme to integrate over a short time by use of a small step length while storing the maxima of the oscillations away, and then a superimposed higher order integration scheme which uses these maxima as supporting values to extrapolate by use of a much larger step length. One such algorithm has been described by Petzold [33]. Another possible solution to this problem may be to solve the problem x''=f(x,x',t), a representation into which many highly oscillatory systems may be transformed, by means of a Fourier rather than a Taylor series expansion. I am unaware of any such code though.

Linear systems may be more efficiently solved by special integration techniques (implicit integration) which is commonly done in network analysis programs but never in general purpose simulation software, although this feature would be very simple to implement.

Noisy systems are formulatable in all languages, but they are not necessarily also solvable, as the variable step length integration algorithms tend to fail, while fixed step algorithms produce rubbish. A (x) here means simply that there is provision for a random number generator, but nothing more than that.

PDE's are solved in simulation by the method-of-lines approach. They are converted to sets of ODE's by means of discretizing them in the space dimensions. The time dimension is kept "continuous" for integration. This methodology works well on parabolic PDE's. The integration over time should use a stiff system solver, as the discretization in the space domain creates stiff sets of ODE's. A sparse linear system solver is here very advantageous.

Hyperbolic PDE's create some headache. A step forward has been made by Carver and Hinds with their up-wind interpolation routine [5]. The idea behind this algorithm is very simple. A central interpolation scheme does not make much sense when a wave moves towards one direction only (direction of the characteristic). Such a scheme would mean to interpolate by use of the unknown future. It is then better to use a biased scheme which only requires values from one side, that is: from the past. The trick in this algorithm is to detect the direction of the characteristic automatically, a task which is much simpler to accomplish than finding the characteristics as a whole.

The Jacobian of hyperbolic PDE's tends to have its eigenvalues also wide spread. However, the resulting ODE's are not stiff, as some of the eigenvalues are complex and close to the imaginary axis. The Gear algorithm does an awful job on them. More research for adequate integration schemes is still required.

Hyperbolic PDE's moreover tend to develop "shock waves" which move through space with time. Such shock waves require an adaptive spatial grid. A first code which offers this feature has been presented recently by Schiesser [42]. It might meanwhile be implemented in either the LEANS or DSS software which are not described further in this survey, as they differ not sufficiently much from FORSIM-VI to demand for a separate discussion. An excellent survey on simulation software for PDE problems has been very recently worked out by Karplus [24].

Elliptic PDE's may be solved by means of invariant embedding techniques. This approach is mostly less efficient though than using a finite element technique. A powerful steady-state solver might make invariant embedding somewhat more attractive again.

An entirely different topic is the discussion of statistical analysis techniques. In current simulation software, these techniques are much further developed in discrete event software than in continuous software, although they may be used to a large extent also there. Some systems (such as SLAM-II) provide for about a dozen different distribution functions. Some systems allow to describe the distribution function in a tabular form to allow for even more general distributions. This feature is very useful when the distributions result from measured quantities. SLAM-II (in connection with the separate program AID) even allows to fit the statistical parameters of some distribution functions such as the BETA and GAMMA distributions from measured data.

Statistical report generation is today only available for discrete system simulation, although it may be equally important to know from a noisy continuous systems in which range the results are expected to lie rather than getting one single trajectory displayed (which may be rather at random). Statistical analysis of the results is available in few systems only. SLAM-II does it in connection with a simulation data-base system (SDL), while GPSS-FORTRAN offers a few functions for that purpose. The separation of the statistical analysis from the simulation task may on the long run be more fruitful, as I shall discuss later.

Table look-up is another critical issue. Most continuous systems provide for means of handling at least two-dimensional tables. Some others also allow to handle three-dimensional tables. Interpolation formulae are mostly linear, but some systems offer also non-linear interpolation (e.g. CSMP-III,) some others also spline interpolation. However, most systems just provide for static tables (e.g. "FUNCTION" in CSMP). This is a nice toy for school examples. Real problems (e.g. wind tunnel experiments) tend to require very huge tables which cannot be handled in this way. A minimum requirement is a dynamic table load feature (e.g. offered in CSMP-III). Even

better is mass storage interpolation which allows to keep the data in the data base (where they belong), rather than to copy them into the central memory. A useful feature is also sequential interpolation which allows to interpolate between data which are cuncurrently produced (e.g. for real-time experiments).

Structural features describe the comfort with which models may be generated by use of a particular simulation software. Subheadings are: application program development, program validation and verification, program execution, data handling and I/O.

A sorting algorithm is available in most continuous simulation systems. It enables the user to specify his set of equations in virtually any sequence. DYMOLA and MODEL [39] (which is not described here as being too similar to DYMOLA to demand for separate discussion) go both a giant step further, in that they allow for the syntax: expression = expression instead of the commonly used syntax: variable = expression. DYMOLA tries to use formula manipulation to solve for the required output variable (which may be context dependent). This procedure shall be only successful if the output variable appears linearly in the equation (though all other variables may appear as non-linear as they like). MODEL keeps the equations as they are, while utilizing an integration algorithm which is able to solve the problem: f(x,x',t)=0. Thus, DYMOLA shall require more compilation but less execution time than MODEL for the solution of the same problem.

Discrete systems may be "parallelized" by means of a PERT network type description or similar. Concurrent processes are another (algorithmical rather than graphical) approach.

All discrete simulation systems offer time events, all combined systems also state events. However, there exist some more event types to be mentioned which are not commonly found in current simulation software. These are the external events. In interactive simulation, the user may wish to suspend a simulation run (e.g. by typing CTRL_C). He may then want to modify a parameter value and resume the simulation thereafter from where it was suspended. Another typical example of an external event would be a real-time interrupt.

Modularity can be guaranteed by quite different approaches. Quite common are macros (partly even rather powerful, e.g. in ACSL). However, macros imply that it is known beforehand, what are inputs and what are outputs of a macro. This is not necessarily always the case. A statement such as: U=I*R may well reappear as: I=U/R in another context. Therefore, macros are modular only in a restricted sense. A much more powerful concept is the concept of "modules" (or "models" as they are called in DYMOLA). The functioning of this mechanism is quite simple. As there exist formula manipulation routines in DYMOLA, it is immaterial, whether the above equation is coded as: U=I*R or I=U/R or even U-I*R=0. The formula solver shall produce the required form automatically depending on the context in which the equation is used. An alternative approach to the previously presented module approach consists of a graphical model specification which is e.g. provided in MODEL. (In fact, this is just another layer of sophistication.)

How are models validated, once they are defined? Obviously, there is no firm and final answer to this question. A first step may be to be able to compare the results from a simulation run to some measured data or to some data produced by another model. Both cases would require the (simulation and measurement) data to be stored away in one and the same data base for later reuse. Sensitivity analysis provides another means for gaining confidence into simulation results. It is possible to automate this procedure even for non-linear models. We have an ALGOL program available which produces the derivative of another ALGOL program with respect to any variable or array of variables [23]. There exists also an experimental version of a PASCAL program for the computation of derivatives of FORTRAN subroutines. Linearized model analysis is very similar. These models may, however, be of lower order. A discussion of so called metamodels can be found from Kleijnen [26]. Also eigenvalue and eigenvector analysis are useful tools to find out whether the modi of the model are within a reasonable domain. This works particularly well for linear models where modal decomposition may be applied.

Debugging of application programs seems not very in vogue today. Some systems at least ask the variables to be declared to allow for some redundancy. Most typing errors can be detected in this way. It is not necessarily true that a simulation system which can represent the Van-der-Pol equation elegantly (in terms of a short user code) is equally well suited for large scale models. Dimensional analysis (by asking the user to provide the dimensions for all of his variables) might be another way of checking correctness. Most systems provide for event monitoring (run-time check). However, other run-time checks would also make sense, like range surveillance (for large scale models) or automated deadlock detection (e.g. timeout).

Most simulation systems today are batch operated. Some systems provide for (still rather moderate) means of interactive operation. Only DARE-ELEVEN, a "dialect" of DARE-P, provides for a run-time display which allows for "on-line" surveillance of some simulation trajectories. There exist some simulation systems for real-time execution (e.g. MICRODARE, another DARE "slang",) which are however not surveyed here as they are supposed to run on very small computers and offer little to no user comfort. These systems hardly offer any of the other

features (beside from the run-time synchronization), and make a discussion therefore not very profitable. The problems here are still so special and machine dependent that it is far too early for a survey.

Data handling and I/O shall be described in due course.

3. SURVEY OF EXISTING SOFTWARE

ACSL [28]: is a powerful continuous simulation language. It is also representative for another program: CSSL-IV [30] which is therefore not further discussed. Both supersede the famous and widely spread CSMP-III software [22]. A current extension to ACSL introduces state events and time events, making ACSL also usable for combined problems [29]. Still, the discrete features offered are very limited, and it is suggested to use ACSL only for "mildly" combined problems (that is: continuous problems with some discontinuities).

DARE [27,49]: stands for an entire family of simulation languages. Described in this survey is the version DARE-P which is the most portable, most powerful, and best tested dialect within the DARE family. DARE-P is certainly less powerful than ACSL, but it has also its distinct advantages. In particular, it provides for a (very primitive) data base interface which allows to compare different simulation results with each other on one sheet of paper. By separating out the postprocessor from the simulation run, DARE-P offers much more flexibility with respect to output representations. The offered portable high quality output is a real jewel. DARE-P is also particularly strong with respect to integration techniques. This software is very simple to teach and to learn, and it is available at nominal cost. The jungest "child" in the DARE family is EARLY DESIRE. An interactive version of DARE-P, offering most of the features of the previous DARE-ELEVEN software, but running on VAX under VMS, is meanwhile made available from ETH Zurich. Another DARE dialect is PSCSP [19], a real-time simulation software from ETH Zurich. There exist also multiprocessor implementations of PSCSP and of MICRODARE.

SIMNON [15,16]: is a highly interactive continuous simulation language running on VAX under VMS. Other versions exist for UNIVAC and DEC-10. A special feature of SIMNON is its notation of subsystems described by difference equations. Such subsystems may freely be mixed with other subsystems described by ODE's. In this way, SIMNON is particularly useful for control engineers simulating continuous plants with digital controllers. For model description, ACSL is in almost any respect superior to SIMNON. However, SIMNON offers a much higher degree of interactiveness than ACSL. Controlled experiments can be executed in a much broader sense, as the "MACRO" feature of SIMNON (for repetitive execution of command sequences) is much more powerful than ACSL's "PROCED" feature. SIMNON provides also for something like a primitive data base mechanism which bridges the gap to some other interactive programs, e.g. IDPAC for parameter identification (available from the same source).

DYMOLA [17]: is really a modeling language rather than a simulation language. A PASCAL coded preprocessor (there exists also a SIMULA coded version of the preprocessor) translates DYMOLA programs into either SIMNON models or FORTRAN subsystems which may be loaded and executed together with the SIMNON system. The beauty of DYMOLA lies in its modularity. Large scale models can be coded much easier and less error prone than in other languages. DYMOLA is in so far experimental, as it offers no other simulation features. That is: the user is often forced to access the precompiled SIMNON program to add some other features at that level. DYMOLA is also representative for MODEL [28] which is therefore not further discussed.

SYSMOD [1]: is basically a workable subset of COSY (discussed further down). The main reason to include this software in this review lies in the assumption that SYSMOD shall be sooner available and better (because industrially) maintained than COSY. Moreover, SYSMOD has also some nice extensions. SYSMOD is certainly the simulation language with the nicest features for description of experimental frames. Controlled experiments can therefore be expressed particularly nicely in SYSMOD. Also the run-time system is improved. Submodels can be separately compiled but nevertheless be jointly integrated. SYSMOD is predominantly a continuous simulation software. There exist discrete events and waiting queues, but no process mechanism is foreseen. SYSMOD is meant for large scale models (with declaration of variables, good structuring capabilities), and is also designed to digest large amount of data (e.g. measurements from wind tunnel experiments or similar). SYSMOD is currently under development by a British Company (Systems Designers, Ltd.) under contract from the British Ministry of Defence. Current plans are to release this software in 1984.

FORSIM [6]: is a continuous simulation software primarily designed for the solution of PDE problems (by use of the method-of-lines approach). Parabolic PDE's can be solved efficiently in one, two or three space dimensions. A very good implementation of the Gear stiff system solver together with use of the Reid sparse linear matrix routines makes FORSIM-VI a very powerful tool for that purpose. Both the integration method as the spatial discretization method are parameterized. The user can select among a large variety of different algorithms by simply changing one single parameter. In this way, FORSIM-VI is also a very nice experimentation tool, in that a large variety of alternative algorithms can be tested out by

minor program modifications. Hyperbolic PDE's can be solved to a lesser extent. Up-wind interpolation implements a "pseudo-characteristic method", but more research is still required here to find better suited integration algorithms and adaptive spatial grids (for shock wave treatment). Elliptic PDE's may be solved to a still lower extent. Finite element methods are far more efficient here for most applications. FORSIM-VI is also representative for some other systems such as LEANS-III [41] or DSS [50] which are therefore not further discussed here.

SIMULA [2]: is a powerful programming language. Its strength lies in the fact that virtually any problem can be solved in a highly structured way. SIMULA'67 is a very good language to implement compilers. As a simulation language, SIMULA offers far too little simulation specific support though. Looking into our table, SIMULA must leave the impression of being a very poor simulation language. In its basic version, SIMULA may be used for the solution of discrete event problems only, but even for that purpose, the available support is minimal. The powerful CLASS concept of SIMULA provides for a virtually unlimited open-ended operator set. No wonder therefore, that there exist several "extensions" to SIMULA (which are basically collections of precut SIMULA classes) to make SIMULA better usable for simulation purposes. One such extension is DEMOS [3] which adds to SIMULA a transaction flow view (comparable to GPSS), tabular distribution functions, statistical report generation (including histograms), event monitoring, and automated deadlock detection (a feature which is rarely found in today's simulation software). Another extension is DISCO [20] which adds some primitives for ODE solution, making DISCO a combined simulation language (although the continuous aspects of DISCO are by far insufficient for complex continuous applications). The class concept of SIMULA is possibly not optimal for maintaining continuous attributes in a user friendly way. Another extension, COSMOS (which is currently under development by Kettenis from the Agricultural University of Wageningen, The Netherlands), goes therefore another way by implementing a (SIMULA coded) preprocessor which translates COSMOS programs down to an intermediate SIMULA code.

PROSIM [45]: is another SIMULA dialect. Its implementation is such that all SIMULA features have been reimplemented in a PL/I environment. PROSIM offers also some combined features. Sierenberg has implemented ODE's in a rather original way, in that the ODE's in PROSIM are not implemented as code, but rather as a data structure. The specification of an ODE in PROSIM is done by declaring a continuous attribute. This genuine solution looks very interesting. It is, however, rather difficult to use in more complex application, as logical connections between different continuous equations are almost unexpressible. All continuous problems which would require somewhere a NOSORT section are almost uncodeable in PROSIM (and these are, to my experience, almost all realistically large problems). PROSIM is therefore basically a discrete event simulation language which allows a few attributes of entities to change continuously in time rather than discretely.

SIMSCRIPT [25]: is another rather popular simulation language. The version described in this report is SIMSCRIPT-II. As with SIMULA, there exist several extensions to this version. SIMSCRIPT-II.5 [40] adds a process interaction to the software comparable to GPSS. Another extension is C-SIMSCRIPT which adds some means for ODE solution, making C-SIMSCRIPT another combined simulation language. The version SIMSCRIPT-II.5 is well maintained by C.A.C.I.

GPSS FORTRAN [43]: is a GPSS dialect which implements most features of GPSS-V [44] in terms of a FORTRAN program. This implementation makes the coding of small GPSS models somewhat cumbersome (longer code) and more error prone (FORTRAN and common blocks), but on the other hand makes this software much more flexible for larger applications. Logical branching is much more generally possible and easier accomplishable in GPSS_FORTRAN-II than in GPSS-V. I, therefore, have not included GPSS-V in my table, as I consider this software to be really obsolete by now. GPSS_FORTRAN-III shall add means for ODE handling, making also this software a combined simulation language.

SLAM [37]: is a very powerful combined continuous and discrete simulation language. Discrete systems are modelled in terms of PERT networks. However, to enhance the flexibility of the tool, special event nodes have been introduced. Whenever a transaction passes through such an event node, an event subroutine (to be coded in FORTRAN by the user) is executed. In this way, SLAM-II combines the comfortable modelling capabilities of its predecessor Q-GERT [36] with the flexibility of its other predecessor GASP-IV [35]. Continuous models are expressed in terms of a FORTRAN subroutine (thus no sorting capability is provided) precisely as in GASP-IV. SLAM-II is still rather weak with respect to its continuous simulation features. In particular, the integration algorithm (a Runge-Kutta code) has been coded directly into the execution control routine, making SLAM unusable for stiff systems (which most of the higher order systems are). SLAM-II is therefore highly recommended for predominantly discrete problems, whereas predominantly continuous problems are better solved by use of other software.

GASP [9,35,38]: is a library of extremely portable FORTRAN coded subroutines for combined continuous and discrete simulation. GASP-IV (which is meawhile obsolete) was designed by Pritsker & Assoc. GASP-V added to GASP-IV many continuous simulation features, making GASP-V as

versatile for continuous as for discrete simulation problems. GASP-V can therefore be considered to be the first fully combined simulation package. GASP-VI finally adds to GASP-V a process interaction view similar to those of GPSS or SIMULA (transactions may pass through work stations as in GPSS, but they may alternatively also suspend processes as in SIMULA). GASP-V is available from ETH Zurich at nominal cost, whereas GASP-VI is still experimental.

COSY [11,12]: was the first simulation language to be formally designed from the beginning with the help of a syntax analysis program [4]. It uses strictly a LL-1 grammar which makes its compiler easily maintainable and upgradable. (Only SYSMOD shares this advantage with COSY). COSY was originally designed as a front end to GASP-V, to make the use of that software somewhat more comfortable, but COSY has meanwhile advanced much further. COSY is by far the most general and versatile simulation language proposed to date. A price, which we have to pay for this versatility and universality, is the complexity of the software. Although it is possible to write users manuals for subsets of COSY, making the use of that subset extremely simple, it is quite difficult to master COSY in its entirety. Thus, COSY shares some of the disadvantages of PL/I.

We may once try to add up all (x) and (xx) over each of the columns, to get a measurement unit for the universality of these simulation software systems. Doing so, we receive the following table:

ACSL	62
DARE-P	51
SIMNON	45
DYMOLA	47
SYSMOD	84
FORSIM-VI	45
SIMULA'67	27
PROSIM	37
SIMSCRIPT-II	26
GPSS_FORTRAN-II	45
GPSS_FORTRAN-III	64
SLAM-II	92
GASP-V	71
GASP-VI	104
COSY	136

Obviously, such a quality measurement has to be taken with care. SIMULA and SIMSCRIPT-II turn out badly, because they do not offer anything for continuous systems, and because their simulation support is insufficient. This does not necessarily mean that there is no space for these languages. Very powerful simulation systems can be coded on the basis of these general purpose programming languages. A very nice continuous simulation language is ACSL, probably the best among the currently available systems. SYSMOD shall still be superior when it becomes available. SLAM-II is highly recommended for discrete simulation. Truely combined simulation should at the moment probably best be performed by using GASP-V.

4. SIMULATION SOFTWARE IN THE FUTURE

One of the problems of modern simulation software lies in its complexity. The appetite of the users has grown drastically, and with it also the number of features, the software is supposed to offer. This is in deed a problem, as it becomes more and more difficult to (a) implement such software, and (b) learn to master it as a user. Every language designer knows that a "good" and "successful" computer language should offer less than about 100 reserved keywords, otherwise the compiler gets large and clumsy. What happens to languages which do not obey that rule, we know at least since the days of PL/I. The user manual of any language should be completely expressible in less than about 100 pages, otherwise the average user shall never be able to master all features offered by the language. Here we run into a serious problem with the design of simulation software, as the number of required keywords is dictated by the complexity of the task rather than by the wishes of the purist language designer.

What can we do to overcome that problem: As I believe, the key lies in the data handling. The base of a simulation system should be a data-base management system (DBMS) adapted to the needs of simulation users. A first step into that direction has been reported recently by Standridge [46,47,48]. Several independent programs for different aspects of system analysis and/or synthesis may then be implemented independently of each other, programs which communicate through their data-base interface. Advantages of this approach are manifold. Let me start with some advantages which concern simulation alone:

1) Representation of one variable trajectory from several runs (overplot). Many current simulation languages offer this feature ("PAGE MERGE" in CSMP-III). However, while current languages require an additional concept to be mastered, implementation of this feature becomes most natural when the data are stashed away into a data-base during simulation, while data retrieval and display are accomplished by a separate postprocessor grouped around the same data-base as the simulation program. The only means of communication between the two programs takes place through the data-base.

2) Representation of one variable trajectory from both simulation and real experiment on one graph. The implementation of this feature requires yet another program for real-time data acquisition grouped around the same data-base. Otherwise, there is nothing new about. This extremely useful

feature (for model validation) does not exist in any of the current simulation languages I am aware of.

3) Dynamic table load. Tabular data need no longer be coded directly into the simulation program (e.g. by a CSMP "FUNCTION" statement), but may be stored in the data-base. These data may be user generated, generated by another (previous) simulation run, or even generated by real measurements. They may then be used as driving functions to another simulation model. One application of this technique could be the solution of the finite-time Matrix-Riccati differential equation where the Riccati equation needs to be computed first backward in time from the final time (T) to initial time (0.), while the system equations must be computed thereafter forward in time from initial time (0.) to final time (T) making use of the previously stored trajectory of the Riccati matrix.

4) Statistical analysis of noisy data. It may often be interesting to analyse stochastic data for their statistical parameters. Again, this is not really a "job" for the simulation language to accomplish. It is much more natural to store the stochastic simulation data away into the data-base, and to analyse them thereafter by an independent statistics program grouped around the data-base.

5) After many replications of a stochastic model, one may wish to display not a particular time history, but a range in which the results are expected to be found. Such a representation is much more useful for a manager, as he may see trends in that curve which are not easily expressible in figures, as not a single digit of the results may be significant. Again, the implementation of such a feature would be a command belonging to one of the postprocessors, and need not really be mixed up with the simulation language features.

6) There may exist several models for the same system, or several submodels to form an entire model. It may be very useful to store also parametric models, sets of parameter values for these models, experimental frames, and possibly some other structures in the data-base for more comfortable model manipulations. These considerations have been discussed in several articles by Oren and Zeigler [31,32].

One may easily find more examples to show that this concept is fruitful. Another advantage of this concept is that independent manuals may be written for the independent program modules. The manager may then e.g. only study the postprocessor manual, as his only direct access to the computer will be to display the data which have been gathered by other people beforehand. In this way, the concept also allows to split portions of the modeling business among several individuals, a separation which is much more natural and much easier accomplishable then to ask several people to write independently different submodels.

One of the beauties of good old closed-shop-batch-processing lay in the fact that the user needed not to learn anything beside the simulation language, where to deliver his cards and from where to get his listings. In the "ideal" case, there existed a special "box" for CSMP input such that the user even was released from those magic control cards he otherwise had to add to the job (and never understood). The introduction of interactive operation made the task somewhat more difficult to the novice user, as suddenly he had to learn something about file manipulation programs (copy, delete, concatenate, etc.) and data manipulation programs (full screen editor). The introduction of our highly recommended new DBMS concept makes his life by no means easier. It is no longer sufficient to specify what results he wants to see, he has to say what data are to be stored, where they are to be stored, where they are to be refound, how they are to be retrieved, what has to happen with these data after the session is over, and so forth. We suddenly realize that simulation no longer consists of a single well defined and well confined task, but that there exists now a SIMULATION ENVIRONMENT similar to the "environment" definition in ADA. A simulation software no longer consists of a simulation run-time system alone possibly preceded by a compilation step (simulation compiler), but -- equally important -- of a simulation environment definition describing the way in which the simulation software is embedded in the operating system of the implementation machine.

Again, there seems to be a problem we should try to do something about. One of the nicest features of modern operating systems is the fact that they allow to describe their own features in their own terms. To state an example: it is possible to implement the operating system UNIX in terms of the operating system VMS running on a VAX computer (which has been done several times). If this program is then automatically executed from within the LOGIN file, the user gets the impression that his VAX machine is running UNIX and not VMS. Obviously, this way of running UNIX is less efficient than running UNIX in native mode, and this technique is therefore not necessarily recommended for a general purpose operating system such as UNIX. However, the nice thing about this technique is that it may very profitably be used for the implementation of a special purpose "SIMULATION OPERATING SYSTEM", a new term which I want to introduce as an alternative to the classical "simulation languages" and "simulation packages". Let me cite a very simple example: to

run ACSL on a VAX 11/780 installation, I have implemented a command procedure (roughly 200 lines of code) to compile, link, and execute ACSL. This command procedure may be used in three different modes:

@ACSL pilot LIST FORT GIGI

would effect the pilot ejection study (on file PILOT.CSL) to be compiled with production of a listing of both the ACSL program itself (option LIST) and its FORTRAN precompilation (option FORT). Thereafter, the program is linked together with the graphics driver for the GIGI terminal (from which the program is operated). Finally, the program is executed. Upon termination, the user is asked whether he wished to receive a hardcopy of both print- and plot-files on a Versatek printer-plotter, and whether he wants to clean up his intermediary files thereafter. Specification of:

@ACSL HELP

would explain what parameters are at the users disposal. A third possibility then would be to specify:

@ACSL

alone in which case the command procedure would enter an interrogative mode and ask for all parameters needed.

This command procedure alone is already quite useful. However, it does not solve all problems for use of ACSL in a class environment. Still, the students would have to learn how to call the editor, copy programs, etc.. For that purpose, another command procedure SIM.COM has been coded which resides on my own accounting number (with read permission only) while being strapped to act as LOGIN file on the students account. This second command procedure (roughly 400 lines of code) enables a menu-type interaction with the user. Each user logging in to the students account is asked first for a second password (his name) after which he enters his own directory. Thereafter, he gets a menu of possible commands displayed which reads as follows:

```
Current ACSL Problem :  NONE

Code: (A) Run ACSL problem
      (C) Clean up files
      (D) Delete ACSL problem
      (E) Edit ACSL problem file
      (F) Edit ACSL data file
      (G) Display general HELP information
      (H) Start/stop HELP menu.
      (L) List of existing ACSL problems
      (M) Display the message of the day
      (N) Print Non-ACSL files (after error)
      (O) Make old version current again
      (P) Purge old versions
      (Q) Show disk quota
      (R) Read file from other problem
      (S) Select ACSL problem
      (T) Display status of queues
      (V) Display VAX-specific information
      (W) Write file to other problem
      (Z) Exit from ACSL account

--------------->
```

which we felt to be about the minimum number of commands, a user must have at his disposal. The user may now e.g. press "L <CR>" to obtain a directory of all of his currently defined ACSL problems. Thereafter, the HELP menu is repeated. Now, the user may press "S <CR>" to be prompted for the name of the problem to be executed (e.g. PILOT). "A <CR>" would then result in a call to the previously described command procedure ACSL.COM for compilation, linkage, and execution of the currently selected ACSL problem. "E <CR>" would call the editor with the currently selected ACSL problem file (depending on the terminal type, this automatically results in either a call to EDT or in a call to TECO). The students obviously must get the impression that their computer runs a special purpose ACSL operating system, even though in reality it runs under VMS (just hidden from the students). The students remain within the LOGIN file throughout their entire session. Our experiences with this mode of operation were extremely positive. The students were able to master this simple simulation operating system without any difficulties after a short demonstration of 20 minutes length.

Currently I am implementing a similar simulation operating system for the second course on discrete event simulation. For that purpose, I base on the software by Pritsker and Associates, that is: the SDL data-base management system together with the SLAM-II simulation software. Lateron, we shall add also the other programs AID (for statistical analysis and interactive fitting of distribution function parameters) and SIMCHART (graphical postprocessor). The full beauties of such a combination would be rather difficult to feel if these programs were used independently of each other. It is really the introduction of the simulation operating system which makes such a system easily manageable and useful.

The idea of such a mode of operation is in deed not really new. Already 10 years ago people were talking about Management Information Systems (MIS) as a cure-all to any disease. Unfortunately, these systems (as far as they ever got) turned out to be diseases in themselves, in that:

1) the systems were huge, clumsy, slow, and unflexible,

2) they never were able to fulfil the task they were build for, as managers were unable to understand what was going on, and therefore (and for good reason) had little trust in the results produced.

Now, 10 years later, the idea of using a computer to take decisions has been burried, still the concept survived in a new outlook, nowadays called Decision Support System (DSS). The idea here is that the manager should no longer be replaced by a computer. Instead, the computer should provide the manager with all available data to take a correct decision. Obviously, the heart of a DSS is again a DBMS, possibly enhanced by some additional modules for statistical analysis, econometric modeling, data display, and similar -- as one can see, precisely the concept which I advertised in this chapter. However, I prefer the term "simulation operating system" over DSS, as this new term does not imply that the tool is to be used by managers only which (partly and at times) well may be the case, but certainly need not. There are many technical applications for which this concept (as shown above) is useful. Moreover, modern operating systems shall now allow much more efficient implementations than what was possible 10 years ago, as this concept lends itself readily to programming at the operating system level (by means of a command language) rather than at the language level (which to a good extent would require assembly programming, as most computer systems would not allow to call a system program (e.g. the FORTRAN compiler) from within a (e.g. in FORTRAN coded) user program). Such comfortable command languages were unavailable 10 years ago.

To close up this short discussion, I strongly advertise a solution in which the many demands for system analysis (and synthesis) features are properly separated into independent modules (with independent documentation) which communicate with each other through a data-base interface. By these means, the flexibility of the resulting tool is drastically enhanced (as has been shown at some examples) while keeping each of the program modules sufficiently simple to let them be user manageable on one side, and efficiently implementable on the other hand.

REFERENCES:

[1] Baker, N. J. C. and Smart, P. J., The SYSMOD Language and Run Time Facilities Definition, Techn Note 6.82, Royal Aircraft Establishment, Farnborough, Hampshire, United Kingdom. (March 1982).

[2] Birtwistle, G. M., Dahl, O.-J., Myhrhaug, B. and Nygaard, K., SIMULA BEGIN. (Studentlitteratur Sweden and van Nostrand Reinhold, New York, 1973).

[3] Birtwistle, G.M., DEMOS: Discrete Event Modelling on SIMULA. (Macmillan, London and Basingstoke, 1979).

[4] Bongulielmi, A. P. and Cellier, F. E., On the Usefulness of Deterministic Grammars for Simulation Languages, Proc. of the SWISSL Workshop, St. Agata, Italy. Shall appear also in Simuletter. (September 1979).

[5] Carver, M. B. and Hinds, H. W., The Method of Lines and the Advective Equation, in Proc. of the ACM SIGNUM Meeting, Albuquerque, New Mexico. (November 1977).

[6] Carver, M. B., Stewart, D. G., Blair, J. M. and Selander, W. N., The FORSIM VI Simulation Package for the Automated Solution of Arbitrarily Defined Partial and/or Ordinary Differential Equation Systems, Report AECL-5821, Atomic Energy of Canada Ltd., Chalk River, Ontario. (February 1978).

[7] Carver, M. B. and MacEwen, S. R., Automatic Partitioning in Ordinary Differential Equation Integration, in Cellier, F. E. (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[8] Cellier, F. E., Continuous-System Simulation by Use of Digital Computers: A State-of-the-Art Survey and Prospectives for Development, in Hamza, M. H. (ed.), Proc. of the International Symposium and Course SIMULATION'75. (Acta Press, Calgary and Zurich, 1975).

[9] Cellier, F. E. and Blitz, A. E., GASP-V: A Universal Simulation Package, in Dekker, L. (ed.), Simulation of Systems, Proc. of the 8th AICA Congress. (North-Holland, Amsterdam, 1976).

[10] Cellier, F. E. and Moebius, P. J., Towards Robust General Purpose Simulation Software, in Skeel, R. D. (ed.), Proc. of the ACM SIGNUM Meeting on Numerical Ordinary Differential Equations, Dept. of Computer Science, University of Illinois at Urbana-Champaign. (March 1979).

[11] Cellier, F. E. and Bongulielmi, A. P., The COSY Simulation Language, in Dekker, L., Savastano, G. and VanSteenkiste G. C. (eds.), Simulation of Systems, Proc. of the 9th IMACS Congress. (North-Holland, Amsterdam, 1979).

[12] Cellier, F. E., Rimvall, M. C. and Bongulielmi, A. P., Discrete Processes in COSY, in Maceri, F. (ed.), Proc. of the European Simulation Meeting held in Cosenza, Italy. (April 1981). Also in Crosbie R. E. and Cellier, F. E. (eds.), TC3-IMACS, Simulation Software, Committee Newsletter, No 11. (July 1982).

[13] Cellier, F. E., (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[14] Eitelberg, E., Modular Simulation of Large Stiff Systems, in Cellier, F. E. (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[15] Elmqvist, H., SIMNON - An Interactive Simulation Program for Nonlinear Systems - User's Manual, Report TFRT-3091, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden. (April 1975).

[16] Elmqvist, H., SIMNON - An Interactive Simulation Program for Nonlinear Systems, in Hamza, M. H. (ed.), Proc. of the International Symposium SIMULATION'77. (Acta Press, Anaheim, Calgary and Zurich, 1977).

[17] Elmqvist, H., DYMOLA - A Structured Model Language for Large Continuous Systems - User's Manual, in Crosbie, R. E. and Cellier, F. E. (eds.), TC3-IMACS, Simulation Software, Committee Newsletter, No 10. (September 1981).

[18] Gear, C. W., Stiff Software: What Do We Have and What Do We Need?, in Aiken R. C. (ed.), Proc. of the International Conference on Stiff Computation, Dept. of Chemical Engineering, University of Utah, Salt Lake City. (April 1982).

[19] Halin, H. J., et alia, The ETH Multiprocessor Project: Parallel Simulation of Continuous Systems, Simulation, Vol 35, No 4. (October 1980).

[20] Helsgaun, K., DISCO - A SIMULA-based Language for Continuous Combined and Discrete Simulation, Simulation, Vol 35, No 1. (July 1980).

[21] Hindmarsh, A. C., Stiff System Problems and Solutions at LLNL, in Aiken R. C. (ed.), Proc. of the International Conference on Stiff Computation, Dept. of Chemical Engineering, University of Utah, Salt Lake City. (April 1982).

[22] IBM, Continuous System Modeling Program III (CSMP III) Program Reference Manual, Program Number 5734-XS9, Form SH19-7001-2, IBM Canada Ltd., Program Product Centre, 1150 Eglington Ave. East, Don Mills 402, Ontario. (September 1972).

[23] Joss J., Algorithmisches Differentieren, Ph.D. Thesis, ETH Zurich, Diss. ETH 5757. (1976).

[24] Karplus W. J., Software for Distributed System Simulation, in Cellier, F. E. (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[25] Kiviat P. J., Villanueva, R. and Markowitz, H. M., The SIMSCRIPT II Programming Language. (Prentice-Hall, 1968).

[26] Kleijnen, J. P. C., Experimentation with Models: Statistical Design and Analysis Techniques, in Cellier, F. E. (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[27] Korn, G. A. and Wait, J. V., Digital Continuous-System Simulation. (Prentice-Hall, 1978).

[28] Mitchell and Gauthier, Assoc., ACSL: Advanced Continuous Simulation Language - User Guide / Reference Manual, P.O.Box 685, Concord, Mass. (1981).

[29] Mitchell, E. E. L., Advanced Continuous Simulation Language (ACSL): An Update, in Ames, W. F. (ed.), System Simulation and Scientific Computation, Proc. of the 10th IMACS Congress, Dept. of Computer Science, Rutgers University, New Brunswick, New Jersey. (August 1982).

[30] Nilsen, R. N., The CSSL-IV Simulation Language, User Manual. Simulation Services, 20926 Germain Street, Chatsworth, California.

[31] Oren, T. I. and Zeigler, B. P., Concepts for Advanced Simulation Methodologies, Simulation, Vol 32, No 3. (March 1979).

[32] Oren, T. I., Computer-Aided Modelling Systems, in Cellier, F. E. (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[33] Petzold, L. R., An Efficient Numerical Method for Highly Oscillatory Ordinary Differential Equations, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Form UIUCDCS-R-78-933. (August 1978).

[34] Petzold, L. R., A Description of DASSL: A Differential/Algebraic System Solver, in Ames, W. F. (ed.), System Simulation and Scientific Computation, Proc. of the 10th IMACS Congress, Dept. of Computer Science, Rutgers University, New Brunswick, New Jersey. (August 1982).

[35] Pritsker, A. A. B., The GASP IV Simulation Language. (Wiley, New York, 1974).

[36] Pritsker, A. A. B., Modeling and Analysis Using Q-GERT Networks. (Halsted Press, New York, 1977).

[37] Pritsker, A. A. B. and Pegden, C. D., Introduction to Simulation and SLAM. (Halsted Press, New York and Systems Publishing Corp., West Lafayette, 1979).

[38] Rimvall M. C. and Cellier, F. E., The GASP-VI Simulation Package for Process-Oriented Combined Continuous and Discrete System Simulation, in Ames, W. F. (ed.), System Simulation and Scientific Computation, Proc. of the 10th IMACS Congress, Dept. of Computer Science, Rutgers University, New Brunswick, New Jersey. (August 1982).

[39] Roth M. G. and Runge T. F., Simulation of Continuous Networks with MODEL, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Form UIUCDCS-R-78-921. (December 1978).

[40] Russell, E. C., Building Simulation Models with SIMSCRIPT II.5. C.A.C.I., 12011 San Vicente Boulevard, Los Angeles, California.

[41] Schiesser, W. E., LEANS - III Introductory Programming Manual, Computing Center, Lehigh University, Bethlehem, Penna. (September 1971).

[42] Schiesser, W. E., Some Characteristics of ODE Problems Generated by the Numerical Method of Lines, in Aiken R. C. (ed.), Proc. of the International Conference on Stiff Computation, Dept. of Chemical Engineering, University of Utah, Salt Lake City. (April 1982).

[43] Schmidt B., GPSS-FORTRAN Version II Einfuehrung in die Simulation diskreter Systeme mit Hilfe eines FORTRAN-Programmpaketes. (Springer, Berlin, Heidelberg and New York, 1977).

[44] Schriber T. J., Simulation Using GPSS. (Wiley, New York, 1974).

[45] Sierenberg R. and de Gans, O., PROSIM Textbook, Dept. of Applied Mathematics, Delft Technical University, Delft, The Netherlands. (1982).

[46] Standridge, C. R., Using the Simulation Data Language (SDL), Simulation, Vol 37, No 3. (September 1981).

[47] Standridge, C. R., The Simulation Data Language (SDL): Applications and Examples, Simulation, Vol 37, No 4. (October 1981).

[48] Standridge, C. R. and Pritsker, A. A. B., Using Data Base Capabilities in Simulation, in Cellier, F. E. (ed.), Progress in Modelling and Simulation. (Academic Press, London and New York, 1982).

[49] Wait, J. V. and Clarke III, D., DARE P User's Manual, Dept. of Electrical Engineering, University of Arizona, Tucson. (December 1976).

[50] Zellner, M. G., DSS - Distributed System Simulator, Computing Center, Lehigh University, Bethlehem, Penna. (May 1970).