*Proceedings of the 1986 Winter Simulation Conference*
J. Wilson, J. Henriksen, S. Roberts (eds.)

# COMBINED CONTINUOUS/DISCRETE SIMULATION APPLICATIONS, TECHNIQUES, AND TOOLS

François E. Cellier
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721, U.S.A.

## ABSTRACT

Beside from purely discrete event and/or continuous system simulations, there exists yet another simulation methodology that combines both classes of simulations into one. It is often possible to model one and the same system by use of completely different world views. Several papers have been written in which one particular application was modeled once by use of continuous system simulation, and once by use of discrete event simulation. Both techniques may eventually lead to the same answers. Sometimes however, one technique lends itself more easily to answering some particular questions about a system while the other is more convenient for answering some other questions about the same system. Thus, before the modeler can decide which methodology best to employ, he must know what purposes his model is going to be used for. In this paper, we want to describe applications that call for a combined continuous/discrete modeling methodology together with the techniques (concepts) that characterize this type of simulation approach. We shall also describe briefly what simulation systems are currently on the software market that can be used for this type of simulations.

## 1. INTRODUCTION

Special-purpose languages for both discrete event simulation and continuous system simulation exist since the fifties of this century, and the first commercially available truly combined simulation software GASP-IV was made available in 1974 (Pritsker, 1974). Therefore, although combined simulation is much more recent than either of its parent technologies, even this technology has been established for roughly a dozen years meanwhile, and thus, one might expect that combined simulation has conquered its market place.

However even after all these years, there can still be found a number of hard-headed simulation software developers who claim that the percentage of applications calling for a combined continuous/discrete modeling methodology makes up less than 5% of the total number of simulation applications, thus, although combined simulation seems to be a nice idea from an academic point of view, it is not really commercially exploitable. However, these so-called «statistics» base on observations of numbers of requests of different kinds received by the company, and are hardly justifiable as a company who has created for itself a name in say continuous system simulation will obviously receive predominantly requests for solutions of problems where this particular technology together with the software marketed by the company look promising. In contrast, a company who is dealing primarily in discrete event simulations will be approached mostly to solve problems that lend themselves naturally to this type of a solution, and thus will «observe» that the majority of simulation applications really are of this nature. It is one of the major aims of this paper to show that there do exist larger groups of real-life applications for which a combined continuous/discrete modeling methodology is the most natural and best suited solution technique available.

Gradually while introducing the different types of applications of combined simulation, we shall identify the techniques (concepts) that make this modeling (and simulation) technique particularly powerful for the application in question.

In the very end of this paper, we shall summarize some of the existing software tools for combined simulation, and discuss for which kinds of problems they are most profitably used.

One final remark with respect to our terminology: Some people call «combined simulation» also «hybrid simulation». We mention this term for further reference. However, we shall not use this term ourselves as the term «hybrid simulation» is also used in a completely different context namely to denote combined analog and digital simulation.

## 2. MODELING OF DISCONTINUOUS FUNCTIONS

Every continuous system simulation language (CSSL) offers a set of discontinuous functions such as a limiter function, a hysteresis function, a dead-space function,

etc. Moreover, most of them provide a «NOSORT» option and/or procedural sections in which ordinary FORTRAN constructs such as IF statements can be employed to model discontinuous functions. No serious modeler today would claim that these mechanisms are not really needed, and that a world view consisting of continuous and continuously differentiable functions only would make much sense.

Discontinuities can be accurately located by exploiting the fact that numerical integration algorithms as offered in todays CSSL's all operate on polynomial extrapolations. As polynomials newer exhibit any discontinuities, obviously the extrapolation around a discontinuity must be in error. However, the accuracy of the numerical integration is controlled by comparing the «result» obtained from different integration algorithms with each other. If they disagree, the step size of the integration will be reduced, and the step will be repeated by using the new smaller step size. Obviously, different polynomial approximations have no reason to agree when integrated through a discontinuity, and thus, the step size control mechanism of the integration algorithm can be used to locate the discontinuity rather accurately.

Unfortunately, this technology is always inefficient, and it may sometimes even fail entirely as the following example will demonstrate.

## 2.1 Speed Control of a Train Engine

Electrical locomotives normally are driven by AC motors. The speed of the engine is controlled by the power *(P)* flowing through the engine:

$$P = U \cdot I \cdot \cos(\phi)$$

where *U* denotes the voltage, *I* denotes the current, and $\phi$ denotes the phase angle between voltage and current in the engine.

One way to influence the speed of the engine would be to prevent current from flowing through the engine for a particular time span (or during a particular angle $\alpha$) in each half period of the sine wave. A thyristor (SCR) in the input loop is fired $\alpha^0$ after each zero crossing of the voltage, and is stopped again as soon as the (lagging) current crosses through zero as shown in Fig. 1.

This technique has one considerable disadvantage. The third harmonic of the Fourier spectrum of the power signal contains a substantial amount of energy. As this system was designed for Switzerland where the trains run on $16\frac{2}{3}$ Hz, the third harmonic is exactly at 50 Hz, that is: it interfers with the electric net frequency. As an effect of this interference, when the first phase cut

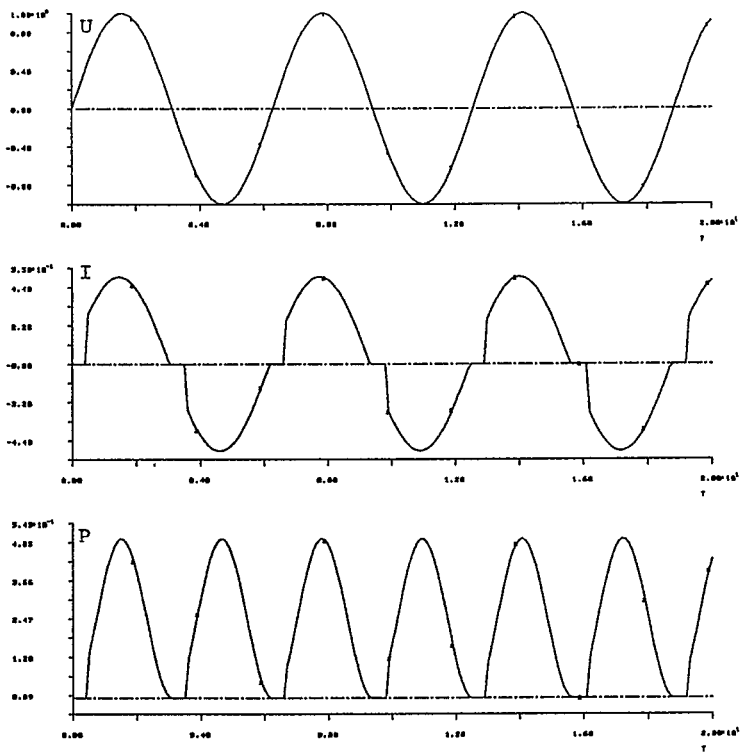


Fig. 1: AC–motor driven in phase cutting technique

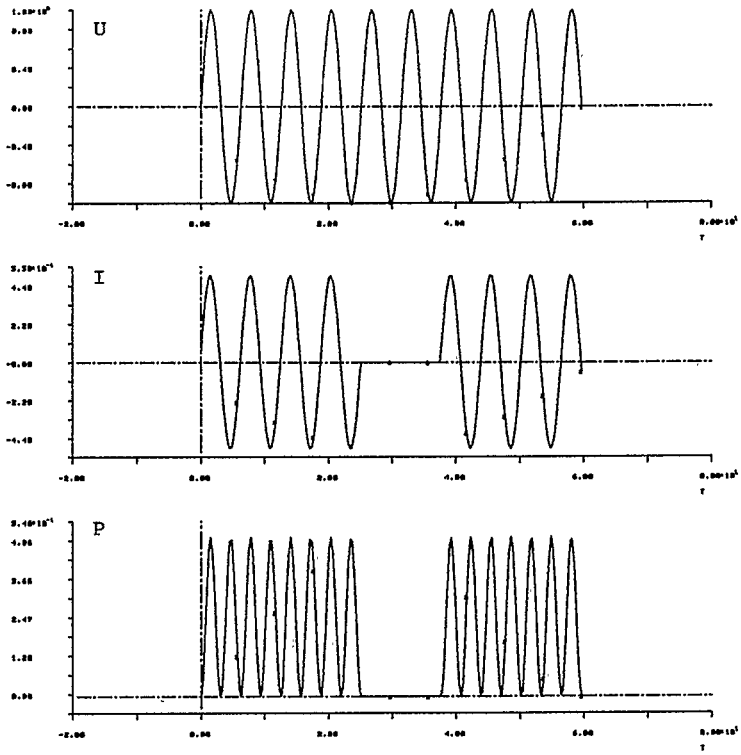


Fig. 2: AC–motor driven in burst technique

driven trains drove up the Gotthard mountain, the electric counters in some of the houses along the rails were reset to zero.

Another solution might be to let a certain number of periods pass through as a whole, while disabling the current during some other period of time as illustrated in Fig. 2.

Also this approach has its considerable disadvantages. If the number of periods forming one «burst» is taken too small, the train does not accelerate and decelerate smoothly enough which puts a discomfort on the travelers. On the other hand, if the burst is chosen large, the train does not react quickly enough in case of emergency.

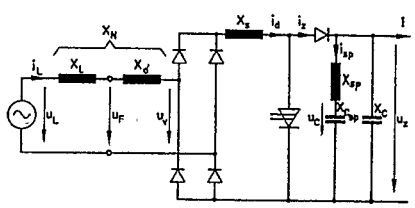

Fig. 3: AC–motor driven in chopper technique

For this reason, one of our former co–workers designed a new circuit that should overcome all these disadvantages (Schlunegger, 1977). The circuit shown in Fig. 3 places a hysteresis around a sine wave of desired amplitude.

An SCR circuit is driven such that the current follows a desired sine wave plus/minus the hysteresis band around it. The power is controlled by changing the amplitude of the desired sine wave. This circuit was expected to work much better than either of the previous alternatives as the power can be altered continuously, and yet most of the power goes into the $16\frac{2}{3}$ Hz frequency line. The third harmonic carries hardly any power. Only much higher harmonics contain a noticeable percentage of the power. The circuit should first be simulated in order to calculate a Fourier spectrum to quantify the amount of power going into the various harmonics.

Our colleague tried to simulate this circuit in one of the standard CSSL's modeling the hysteresis by use of the built in HSTRS–function. However, the simulation did not work at all. Fig. 4 shows the result of the simulation (current), and Fig. 5 shows the integration step size as a function of simulated time.

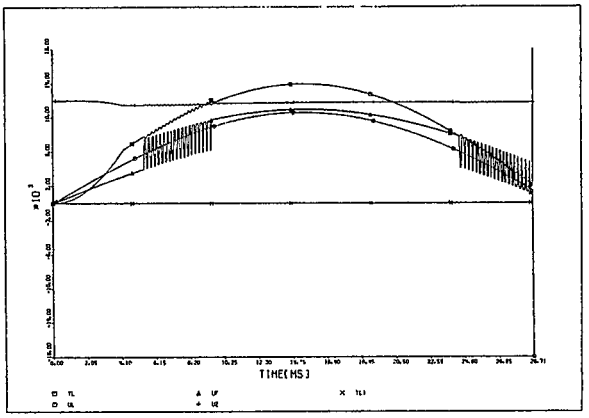


Fig. 4: Current as a function of simulated time

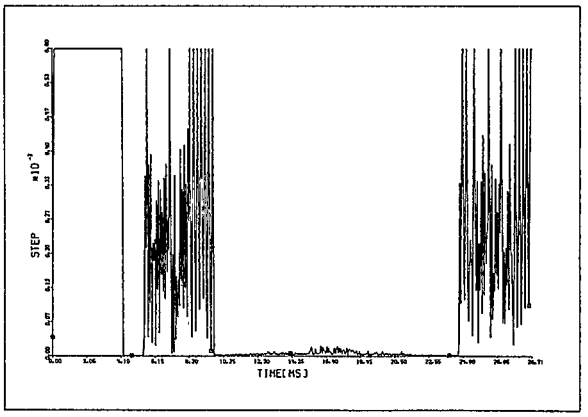


Fig. 5: Step size as a function of simulated time

As can be seen from Fig. 4, the simulation first exhibits a correct pattern, and although the step size is trying to adjust itself frantically to the frequent discontinuities taking place, the simulation works fine. Suddenly however, the oscillation dies out, and at the same time, the step size drops to a very small value making the –incorrect– simulation extremely expensive to run. What went wrong? Due to the steep gradient in the current, the model switches several times back and forth within one single integration step, thus invalidating the integration, and thus leading to a reduction in the step size. However, when the step size is sufficiently small, all higher order terms in the polynomial approximation become insignificant, and any explicit integration scheme is finally behaving like a forward Euler integration. Therefore, the two integration algorithms compared to each other shall finally «shake hands» which may happen either on the correct or on an incorrect result. In our simulation, the number of model switchings taking place in the finally accepted integration step was odd (and thus «correct») during the initial simulation phase, but it was even (and thus «incorrect») during a later stage of the simulation. Therefore, the model is on the «wrong side of the fence» after the step has finally been completed, and the simulation tries to repeat the cross–over during the following step obviously with equally little success. The integration step remains thus small until the number of model switchings again becomes odd at yet a later time instant when the simulation finally resumes «correct» operation.

Similar effects were observed during the simulation of the «rattling» of undercritically damped electrical discharge machines for dye–sinking work, in the simulation of short circuits on electric power lines which simply failed altogether unless the short circuit was assumed at time zero, and in several other cases of highly discontinuous system behavior. None of the above examples is «far fetched». These are simply the type of examples that a simulation specialists meets in everyday's practice once he advances beyond the typical school book examples of Van-der-Pol's equations and the pilot ejection study.

What can be done about. Obviously, it was not such a splendid idea after all to misuse the step size control mechanism of the numerical integration algorithm for discontinuity handling. Instead, we must *tell* the simulation program explicitly that a discontinuity takes place.

## 2.2 Generator Functions and Scheduled Events

One type of discontinuities that can take place in an otherwise continuous model is a discontinuous input function. It may e.g. be desirable to drive a model with a square wave generator as depicted in Fig. 6:

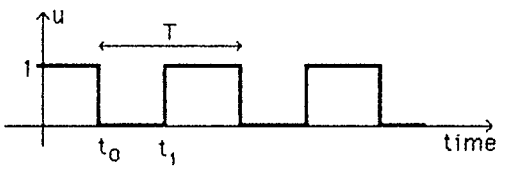


Fig. 6: Square wave generator function

This can be described by assigning an initial value to $u$, and by scheduling two initial time events to take place at times $t_0$ and $t_1$, resp.:

```
INITIAL
  u = 1.0
  SCHEDULE down AT t0
  SCHEDULE up AT t1
END INITIAL
```

Each event description schedules a new event of the same type to happen $T$ time units into the future:

```
EVENT down                EVENT up
  u = 0.0                   u = 1.0
  SCHEDULE down IN t        SCHEDULE up IN t
END down                  END up
```

By use of a mechanism well known from discrete event simulation, we were able to tell the simulation program explicitly about the forthcoming discontinuity. The last step before getting to the next discontinuity can be automatically reduced to hit the discontinuity accurately. No unnecessary repetition of integration steps is going to take place. Moreover following the discontinuity, the integration algorithm can be restarted from scratch avoiding an integration through the discontinuity altogether. This is why this approach is called combined continuous/discrete simulation.

### 2.3 State Events and State Conditions

Not all discontinuities can be resolved by scheduling events ahead of time. More frequently in fact are discontinuities that do not depend on time directly but rather on another (time-dependent) variable of the model such as the limiter function depicted in Fig. 7.

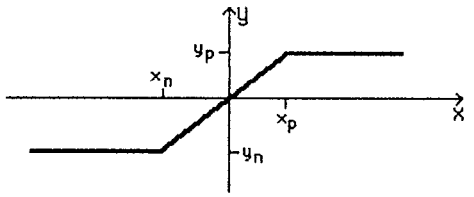


Fig. 7: Limiter Function

This function can e.g. be modeled as:

```
y = IF x < xn THEN yn
    ELSEIF x < xp THEN c*x
    ELSE yp
```

At a first glance, this looks like a more fancy version of what we would have programmed in a procedural section of one of the current CSSL's. However, the simulation preprocessor is expected to translate this convenient IF statement into code that automatically checks so-called *state conditions* that decide whether the model is currently about to switch from one branch of the discontinuous function to another, and if so, iterate to hit the discontinuity with a prescribed accuracy, then execute immediately a so-called *state event* that performs the switch-over, and finally restart the integration algorithm from scratch thereafter.

Let us denote state conditions by a WHEN construct of the following form:

```
WHEN x > xn THEN SCHEDULE mod2
```

which reads as: «when $x$ becomes larger than $x_n$, schedule immediately the state event called *mod2*». Using this notation, we can recode the limiter function as:

```
CASE modtype OF
  neg:  y = yn
        WHEN x > xn SCHEDULE mod2
  cent: y = c*x
        WHEN x < xn SCHEDULE mod1
        WHEN x > xp SCHEDULE mod3
  pos:  y = yp
        WHEN x < xp SCHEDULE mod2
END CASE
```

together with three event descriptions of the form:

```
EVENT mod2
  modtype = cent
END mod2
```

This is obviously a much more clumsy way to model the limiter function than the more compact IF notation proposed previously. However, this is exactly the kind of notation that the preprocessor is supposed to generate out of the IF statement contained in the user program.

However, it is important that the user has direct access to the WHEN clause as well. Not all discontinuous functions can be expressed through IF statements. Let us consider the dry hysteresis function as shown in Fig. 8.

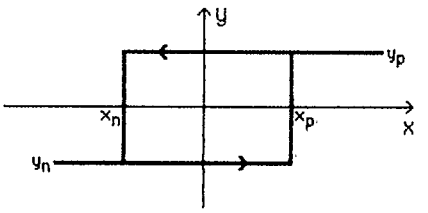


Fig. 8: Dry hysteresis Function

Obviously, the IF statement won't work, as this function is multivalued. However, the WHEN clause will work fine as the following program segment shows:

```
WHEN x < xn THEN y = yn
WHEN x > xp THEN y = yp
```

which must be accompanied by an appropriate initial condition:

```
INITIAL
  y = yp
END INITIAL
```

Again, this is a convenient abbreviation for explicitly scheduling state events, but the preprocessor can generate appropriate code out of this convenient user description.

The same mechanism can also be used to more appropriately terminate simulation runs, e.g. by writing:

```
WHEN xy < 0.0 THEN TERMINATE
```

This will ensure that the simulation not simply stops after $xy$ has become smaller than $0.0$, but that an iteration takes place to locate the zero crossing more accurately. Looking into some descriptions of the famous pilot ejection study, e.g. (Korn, 1979), one may notice that the step size of the simulation was restricted on purpose to avoid wrong decisions to be taken. Again, this is a terrible waste of computational time.

This concludes the description of the mechanisms that are required for combined continuous/discrete simulation. In fact, a combined simulation program can be viewed as a discrete event simulation program in which a continuous simulation takes place between any two consecutive event times.

## 3. MODELING OF SAMPLED DATA SYSTEMS

Some simulation languages provide special means for the simulation of sampled data systems, e.g. by introducing another operator to denote sampling delay. Unfortunately, this method is somewhat restrictive as it usually does not allow to simulate multi-rate sampled data systems in which the delay time of separate controllers can be chosen separately.

We believe that a more appropriate way to represent sample data systems is by way of combined continuous/discrete simulation. A discrete controller is simply represented as a self-generating time event:

```
EVENT sampling
  SCHEDULE sampling IN ts
  z = z + dz
END sampling
```

where $dz$ represents the rate by which the discrete state variable $z$ is to be changed at each sampling point. Obviously, $dz$ will usually be computed within the event. $T_s$ denotes the sampling interval. Obviously, this event calls for an initial condition:

```
INITIAL
  z = 1.0
  SCHEDULE sampling AT ts
END INITIAL
```

Multi-rate sampling does not pose any problem here. Simply group all variables that use the same sampling interval and sampling times together into one state event, while all others are specified in other state events. Obviously, this approach also guarantees appropriate handling by the integration algorithm, as integration will automatically be restarted after any sampling has taken place.

## 4. MODELING VARIABLE STRUCTURE SYSTEMS

Sometimes, one is interested to model a system that changes its structure entirely at event times, e.g. the number of differential equations may change. Traditionally, this is the type of problem that was considered a «combined continuous/discrete» problem. We do agree that such problems are more complicated to handle, and that they probably do not occur so frequently in practice. Often quoted examples are simulations of chemical batch processors (Pritsker, 1974), harbor operations (Pritsker, 1979), and heating of steel ingots (Fahrland, 1970).

We proposed another interesting benchmark problem to test the capabilities of simulation systems. Domino stones are placed at a distance $d$ from each other. The first stone is pushed, and after some time, all domino stones have fallen flat. Which is the distance $d$ between two consecutive stones that maximizes the chain velocity (Cellier, 1979)?

Obviously, one does not want to consider the differential equations of hundreds of domino stones out of which most have already fallen or are still standing upright. Instead, the simulation software should provide a mechanism to describe the differential equations of one «model» stone. State events are used to «create» a new active stone whenever the next stone is pushed, and «delete» an old stone from the active list of stones whenever one stone has fallen flat. This calls for a mechanism equivalent to a SIMULA «class» to model individual stones described by their own differential equations.

One may find that variable structure simulations often consume a considerable amount of computational time. Thus, one may be inclined to aggregate the model

further into an entirely discrete simulation. This is perfectly legitimate whenever it is feasible. Even in most of the examples quoted in (Pritsker, 1974 and 1979), such an aggregation is feasible and does make sense. Unfortunately, this approach does not always work. The domino example is one which is somewhat tricky, and at least we have not been able to find a legitimate aggregation due to the strong interaction between neighboring stones.

Practical examples include the tactical simulation of fighter airplanes. Due to time considerations, problems of this kind were mostly aggregated into discrete event models in the past, but eventually, some valuable information about the dynamics of the particular aircraft to be simulated will be lost in the aggregation process. Other applications that actually were modeled in a combined continuous/discrete manner in the past include various missile simulations. For obvious reasons, these simulations are not widely publicized though.

## 5. RULE-BASED CONTROL SYSTEM DESIGN

How does a human operator control a process, e.g. how does a pilot fly an aircraft? Obviously, he doesn't solve any Riccati equation in his head. Instead, he uses a number of meters (variables) that he observes. Whenever one of the variables is outside its expected range which in return is evaluated by means of a «qualitative simulation» of a «mental model», the pilot reacts in one way or the other. The observed variables together with the mental model of the aircraft represent the «knowledge» of the pilot. Control is done by applying this knowledge to a «rule base» which is a set of rules of the type: «WHEN this and this happens, THEN do that and that».

Automatic controllers as they are in use today work very well for the local control of subsystems. However, they do a poor job with respect to global assessment of complex processes, and with respect to learning when confronted with unforeseen situations.

Unfortunately, the human operator becomes unreliable if the amount of information to be monitored grows beyond an amazingly small level. Psychological studies have shown that even a trained human operator cannot observe reliably more than about 10 different pieces of information at any one time. Several drammatic aircraft accidents have already happened for exactly this reason.

Currently, we build ever more complex processes. For such applications, e.g. the forthcoming space station, nuclear power plants, a surgical operation room, etc., we must overcome this problem by being able to «simulate» human behavior in an automatic device. We are convinced that these types of simulations will play an ever increasing role in coming years. We further propose that combined continuous/discrete simulation is exactly the tool that is needed for this type of simulation.

The «plant» is represented by a conventional model, e.g. a continuous model described by differential equations. From this model, a number of variables are «observed» (they are output variables). The «mental model» can be represented in several ways, e.g. by means of simple difference equations, some discrete events, or an input/output model as e.g. described in (Klir, 1985). The «rule base» really is nothing but a set of state conditions and state events in the language of combined simulation:

```
WHEN condition1 THEN action1
WHEN condition2 THEN action2
.....
```

Each of the conditions is one state condition to be tested, and each of the actions is one state event to be performed if the condition comes true.

## 6. COMBINED SIMULATION SOFTWARE

In this brief (and incomplete) survey, we want to mention some of the software systems that are currently on the software market, and that found a certain degree of recognition. In this context, it is more important to us to classify these -representative- systems appropriately into different categories, rather than to give a complete enumeration of available simulation software systems. A more complete survey of simulation software can be found in (Cellier, 1983).

### 6.1 Continuous System Simulation Software with Features for Discontinuity Handling

Some software systems with a limited capability for «combined» simulation were developed out of purely continuous system simulation software. Among those, we would like to mention just two:

**ACSL:** (Mitchell & Gauthier, 1981) This is one of the more frequently used continuous simulation software systems. For purely continuous simulation, this CSSL has created for itself a very good reputation. Recently, time events and state events were added to the software allowing for appropriate treatment of discontinuities, and also enabling correct simulation of sampled data

systems. However, no other attributes of discrete event simulation were added. There are no waiting queues except for the event queue itself. There are no appropriate random number generators for stochastic event scheduling, etc. Moreover, the discontinuous functions within ACSL (such as a hysteresis function) were not adjusted to the new mechanisms, therefore, discontinuous systems must be user decomposed into proper event descriptions. There is currently no concept in ACSL for «hidden events», that is: system maintained events. IF statements can be used within procedural sections, but they are not decomposed into an event handling mechanism as discussed earlier in this paper.

**SIMNON:** (Elmqvist, 1975) This software system was conceptualized for the simulation of sampled data systems. Subsystems can be either «continuous» or «discrete», and there exists a «connecting system» to connect the different subsystems together. However, this mechanism is not hierarchical. SIMNON was designed for the simulation of sampled data control systems described in a block diagram manner. Although sampled data systems are treated correctly, the mechanism employed in SIMNON is even less general than the one used in ACSL. There does not exist a user accessible event calendar. However, SIMNON was the first more widely used «combined» simulation software within this class of systems.

## 6.2 Discrete Event Simulation Software with Continuous Attributes

Most of today's combined simulation systems originally grew out of discrete event simulation systems. This is understandable as it is much easier to embed a continuous simulation software into a discrete event simulation software, than it is to incorporate event handling capabilities into an existing continuous simulation software. Let me mention just a few systems of this type:

**GASP–IV:** (Pritsker, 1974) GASP–IV was the first commercially available (and successful) combined simulation software on the software market. It grew out of GASP–II, a very simple discrete event simulation system. All GASP versions actually are nothing but a set of FORTRAN subroutines that the user can call to perform simulations. However, we would hesitate to call GASP–IV a truly combined simulation software. Its parent software (GASP–II) is very obvious. GASP–IV has definitely strong attributes towards discrete event simulation, but is weak in handling continuous systems. In particular, one particular Runge–Kutta integration algorithm was built directly into the software which makes the software useless for simulations of systems where Runge–Kutta integration is not appropriate, such as stiff systems, and highly oscillatory systems. Also, continuous system simulationists like to make use of system functions such as a hysteresis, a dead–space, etc. None of these functions is available in GASP–IV. Finally, the graphics capabilities of GASP–IV are very poor.

**SLAM–II:** (Pritsker, 1979) SLAM–II grew out of GASP–IV. In fact, it is a true superset of GASP–IV, making the previous system obsolete. However, SLAM–II enhanced further the discrete side of the software (by adding a network modelling capability), whereas the continuous side is as rudimentary as in GASP–IV. Therefore, we suggest to use SLAM–II for the simulation of predominantly discrete systems with some (few) continuous attributes. It is definitely not convenient to use for full–fledged continuous system simulations. We suspect that the major reason for the impressive success that both GASP–IV and SLAM–II had stems from their excellent documentation. Both books (Pritsker, 1974, 1979) are not just software manuals, but can be (and are) used as textbooks for discrete event simulation.

**SIMAN:** (Pegden, 1982) SIMAN is mainly a reimplementation of SLAM–II. Differences between the two software systems are minor, and in particular, the continuous portion is still simply a copy of the old GASP–IV concepts. As SIMAN was developed later, its developers were able to avoid some of the «mistakes» that went into the design of SLAM, and SIMAN is sometimes a little ahead of the game with respect to new features offered. (Both SLAM and SIMAN are continuously being further developed.) Unfortunately, the SIMAN documentation is much less convincing than the SLAM documentation.

## 6.3 Truly Combined Simulation Software

We want to mention only two systems under this heading:

**GASP–V:** (Cellier, 1976) GASP–V is a «nephew» of SLAM–II. It grew also out of GASP–IV. However contrary to SLAM–II, GASP–V did not develop the discrete portion of the code any further. As a matter of fact, the discrete modelling capabilities of GASP–V are identical with those offered in GASP–IV. Instead, the continuous portion was developed further. In GASP–V, a representative library of integration subroutines has been made available. A set of discontinuous functions (comparable to the one offered in ACSL) was implemented, resolving all discontinuities by means of internal hidden events. In GASP–V, we also added software for the simulation of distributed parameter systems by use of the method–of–lines approach, and as

far as we know, GASP–V is still the only software available designed to model systems of PDE's coupled with ODE's, and allowing for discontinuities. In GASP–V, we also added a graphical postprocessor. (This feature is meanwhile also available in SLAM–II when operated through TESS, and in SIMAN when used together with CINEMA.)

**SYSMOD:** (Baker, 1986) SYSMOD is the first commercially available simulation system that was designed from scratch with a focus on combined simulation. SYSMOD can be viewed as a superset of PASCAL. It is a strongly–typed highly–structured language. A PASCAL–coded preprocessor translates SYSMOD models into FORTRAN. SYSMOD is the first simulation system on the market that truly supports separate compilation of even tightly coupled continuous subsystems. Also the experiment can be compiled separately from the model. SYSMOD is currently a little biased towards the continuous side, but it does offer a full set of discrete event handling capabilities (no process description mechanism though). Due to strong typing, SYSMOD is less easy to use for the modeling of simple continuous systems than ACSL. Therefore, SYSMOD is particularly powerful for the simulation of *large scale systems*, such as power plant simulations, and missile simulations. The complexity of the SYSMOD grammar is similar to that of ADA, but SYSMOD was more systematically designed by use of a rigid general–purpose LL(1) recursive–descent parser. The IF, WHEN, and CASE examples given earlier in this paper are all (somewhat stripped down) versions of SYSMOD code. For predominantly discrete simulations with few continuous attributes, we suggest to use either TESS (with SLAM–II) or CINEMA (with SIMAN). However, for predominantly continuous simulations with heavy discontinuities and/or complex decision making, SYSMOD may currently well be your best bid.

## REFERENCES


Baker, N. J. C. (1986). *SYSMOD, User Manual.* Systems Designers plc, Ferneberga House, Alexandra Road, Farnborough, Hants GU14 6DQ, United Kingdom.

Cellier, F. E. (1979). *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools.* PhD Thesis, Diss ETH No 6483 Swiss Federal Institute of Technology Zürich, Switzerland, 266p.

Cellier, F. E. (1983). Simulation Software: Today and Tomorrow. In: *Proc. Simulation in Engineering Sciences* (J. Burger and Y. Varny, eds.). North–Holland, pp. 3–19.

Cellier, F. E. and Blitz A. E. (1976). GASP–V: A Universal Simulation Package. In: *Proc. 8th AICA Congress on Simulation of Systems* (L. Dekker, ed.). North–Holland, pp. 391–402.

Elmqvist, H. (1975) *SIMNON – An Interactive Simulation Program for Non–linear Systems – User's Manual.* Report CODEN: LUTFD2/(TFRT–7502), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

Fahrland, D. A. (1970) Combined Discrete Event / Continuous System Simulation. *Simulation,* **14**(2), pp. 61–72.

Klir, G. J. (1985) *Architecture of Systems Solving.* Plenum Press, 539p.

Korn, G. A. and Wait, J. V. (1978). *Digital Continuous–System Simulation.* Prentice Hall, 212p.

Mitchell E. E. L. and Gauthier, J. S. (1981) *ACSL. Advanced Continuous Simulation Language – User/Guide Reference Manual.* Mitchell & Gauthier, Assoc., 1337, Old Marlboro Road, Concord, MA 01742, U.S.A.

Pegden, C. D. (1982) *Introduction to SIMAN.* Systems Modeling Corporation, 258p.

Pritsker, A. A. B. (1974). *The GASP–IV Simulation Language.* John Wiley, Interscience, 451p.

Pritsker, A. A. B. (1979). *Introduction to Simulation and SLAM–II, Third Edition 1986.* Halsted Press and Systems Publishing Corp., 839p.

Schlunegger, H. (1977). *Untersuchung eines netzrückwirkungsarmen, zwangskommutierter Triebfahrzeug–Stromrichters zur Einspeisung eines Gleichspannungszwischenkreises aus dem Einphasennetz.* PhD Thesis, Diss ETH No 5867, Swiss Federal Institute of Technology Zürich, Switzerland.

## BIOGRAPHY

FRANÇOIS E. CELLIER is an associate professor in the Department of Electrical and Computer Engineering of the University of Arizona. He received his Diploma in 1972, and his Ph.D. degree in 1979, both from the Swiss Federal Institute of Technology, ETH–Zürich, Switzerland. His main scientific interests concern modeling and simulation methodology and the design of advanced software systems for simulation, computer–assisted modeling, and computer–aided control system design. He has designed and implemented the GASP–V simulation software, and designed the COSY simulation language which served as the «model» language for SYSMOD. He has published more than 30 papers on the subject of modeling and simulation, and he has edited several books on the same topic. He is a member of IMACS and SCS.

François E. Cellier
Dept. of Electr. & Comp. Engr.
University of Arizona
Tucson, AZ 85721, U.S.A.
(602) 621–6192