METHODOLOGY IN SYSTEMS MODELLING AND SIMULATION
B.P. Zeigler, M.S. Elzas, G.J. Klir, T.I. Ören (eds.)

COMBINED CONTINUOUS/DISCRETE SYSTEM SIMULATION LANGUAGES --- USEFULNESS, EXPERIENCES AND FUTURE DEVELOPMENT

François E. Cellier
Institute for Automatic Control
The Swiss Federal Institute of Technology Zurich
Physikstr. 3
CH-8006 Zurich
Switzerland

Combined system simulation is a relatively new technique for the simulation of a class of systems having properties suitable to both continuous system simulation and discrete event simulation, two techniques well known to the simulation community. This combined technique has first been proposed by Fahrland [13,14].

This paper surveys the techniques and methodology involved in this simulation approach. Major aspects considered are numerical behaviour and information processing. It is shown that this technique is applicable to a much larger class of problems than originally suggested by Fahrland [13,14].

I) INTRODUCTION:

The term "combined simulation" is not yet sufficiently well understood in the literature to mean one and only one specific methodology or problem class. For example, one can find references where combined simulation is used as synonym for hybrid simulation. This term, therefore, first requires some definition to clarify how it is going to be used in this paper.

If one speaks of simulation as a technique one usually thinks of a specific solution tool (digital simulation, analog simulation, hybrid simulation). On the other hand the term "system simulation" refers to a specific class of systems under investigation (continuous system simulation, discrete system simulation). However, as early as 1967 Kiviat [17, p.5] stated that it is common to find the terms "simulation" and "system simulation" used interchangeably. In this paper we do not have primarily a specific simulation methodology in mind, but rather the simulation of one specific class of problems which we call combined systems. However, restricting ourselves to fully digital solutions only; simulation of this class of problems does suggest the use of a specific simulation methodology which we are going to discuss in detail. Although the term "combined system simulation" is thus appropriate, we will use the term "combined simulation" as well for simplicity.

It remains to define what the term "combined systems" means precisely. It can be paraphrased as follows:

> *Combined systems are systems described, either during the whole period under investigation or during a part of it, by a fixed or variable set of differential equations where at least one state variable or one state derivative is not continuous over a simulation run.*

Using this definition the famous pilot ejection study (which is probably the best known test case for continuous system simulation) will also fall into this class of problems, since the acceleration of the ejector seat and the first time derivative of its angular position (both state derivatives in the system's definition) are discontinuous at the moment when the ejector seat is disengaged from the mounting rails.

The most comprehensive volume on combined simulation published to date [27] cites two examples of continuous systems -- the above mentioned pilot ejection study and an analysis of a slip clutch. Both do belong to the class of combined systems according to our new definition. This shows that the definition used here is not entirely in accordance with the "common" use of this term (as a matter of fact, a proper definition for this term has never been given!). We must redefine the term "continuous system" as well to keep it consistent with our definition for combined systems.

> *Continuous systems are systems described by a fixed set of differential equations with state variables and first state derivatives both being continuous over the whole simulation run.*

This definition restricts the term "continuous system" to a more narrow sense than is commonly used.

According to these definitions for combined and continuous systems, most of the more complex "continuous" systems belong to the class of problems under investigation. The motivation for this definition will be given in due course.

As the traffic control example presented in [8] shows, one and the same physical system may be modeled either in an entirely continuous, entirely discrete, or in combined fashion. The term "combined system" is, therefore, rather an attribute of the selected model than of the underlying physical process, and we should thus better talk of "combined models" than of "combined systems". As explained in [34], one should in any event not expect to simulate a physical system but rather a model derived from the physical system via an experimental frame (within which data can be collected representing the behaviour of the real system under specified experimental conditions). Hopefully, under novel experimental conditions the constructed simulation program will produce data representative of the data to be observed when the real system is observed under these new conditions. However, since we will disregard the problem of modeling in this paper entirely, we shall use the terms "model" and "system" interchangeably.

II) HISTORICAL DEVELOPMENT:

Surveying simulation languages and packages for combined system simulation available on the software "market" we may find that most of them are extensions of existing "pure" discrete simulation languages/packages. As examples we may mention:

GASP-II --> GASP-I .
SIMSCRIPT-II.5 --> C-SIMSCRIPT
SIMULA-67 --> CADSIM.

The reason for this is the following: Although a numerically well performing package for continuous system simulation is much more difficult to achieve than one for discrete system simulation, the language structures for the latter are much more complex than for the former. Thus, extending discrete simulation languages to encompass combined problems is a much easier task to achieve than extending a continuous simulation language for that purpose.

Extensions of discrete simulation packages have been performed in most cases either by the original designers or at least by former users of the original software. However, these people usually having a background in operations research, normally do not consider the requirements of systems' analysts for continuous systems from either the numerical or information point of view. For example, one may find that one specific integration algorithm has been coded into the control routine, or that no provision has been made for parallel structures, adequate run-time control procedures, etc..

We have already seen that most so-called "continuous" systems are really "combined" systems according to our definition. On the other hand there exist many systems, which may be conveniently described by purely discrete simulation elements. As a result there is a much greater impact of combined simulation on the treatment of continuous processes than of discrete processes. This allows for the conclusion that the state-of-the-art of combined system simulation languages is by no means satisfactory yet.

III) USEFULNESS OF COMBINED SYSTEM SIMULATION:

In references [14,15] it has been shown, that there exist problems which cannot be modeled in a proper way by either purely discrete or purely continuous simulation elements. (The author believes, however, that for most problems it is possible to find either an entirely discrete or an entirely continuous formulation, but the work required for converting the problem may be considerable and even may not be desirable.) Examples given in the above references include a steel soaking pit and slabbing mill and also a chemical batch process. The arguments given in these references to justify the new combined approach to these processes are certainly correct. However, we shall show that the needs for combined simulation languages are even more evident and elementary than explained in these references.

Reference [30] describes control of the motion of trains by SCR's (silicon controlled rectifiers). For this purpose a current has to follow a sine wave within a prespecified tolerance range. This was achieved by controlling the SCR's in a way that the current always switches back and forth between basically two different models where reaching the tolerance bound is the condition for switching to the other model. A simulation using CSMP-S/360 required approximately US $400 for one half period of the sine wave, whereas a simulation using GASP-V required only US $13 for two whole periods of the sine wave. (The CSMP program was coded using only standard features offered by the CSMP language.) Thus the reduction factor in computing time was more than 100 for this example. The reason for this is that CSMP utilizes the step size control mechanism of the integration algorithm for event location which is a rather inefficient method for location of state-events and an extremely inefficient method for location of time-events. (For the definition of the terms state- and time-event see [27].) Any other continuous simulation language would behave in the same way as CSMP, so this is not a shortcoming of the particular language CSMP, but a problem of the inadequate underlying solution technique applied.

Fig.1 depicts schematically what happens to the integration step size at event times.

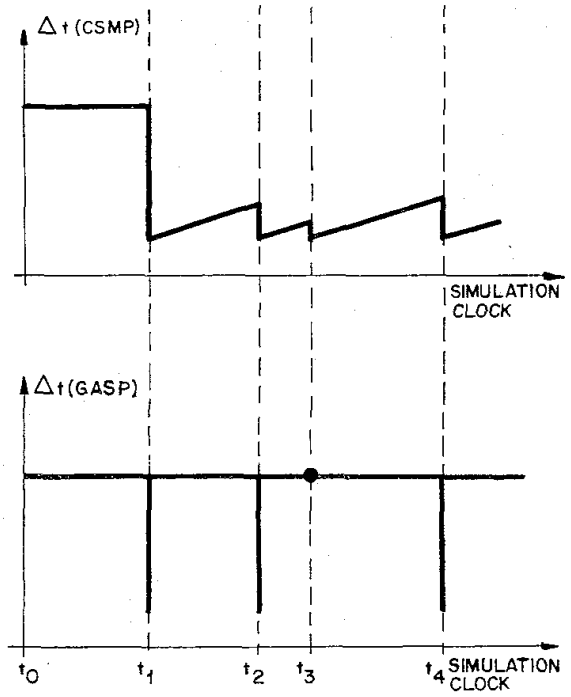

Fig.1: Integration step sizes of continuous and combined languages versus the simulation clock.

Using a continuous simulation language the step size will be reduced heavily when a discontinuity is encountered since the integration algorithm is unable to compute a step properly over discontinuities. This fact has been used in these languages to "localize" event times. However, since the program cannot know that there is a discontinuity taking place (no language element is provided to explain this to the system) the algorithm will "think" that the set of equations suddenly became extremely stiff and reduce the step size to cope with the new situation.

Having located the event, the algorithm will carefully explore the possibility to make the step size larger again, but is not allowed to enlarge the step size immediately since this would lead to instability behaviour in a real stiff case.

A combined simulation language, on the other hand, will provide for a language element to describe discontinuities. Now the step size will be reduced by an event iteration procedure inherent in the program to locate the unknown event time in case of state-events (events 1, 2 and 4 on Fig.1) taking place, whereas the step size will simply be reset to the known event time in case of a time-event (event 3 on Fig.1) taking place. After accomplishment of the event the integration algorithm will be restarted and use the internally provided (and hopefully efficient) algorithm to obtain a good guess for the new "first" step size to be used. Note, that the independent axis on the graph denotes the simulation clock and not CPU-time.

Considering the numerical aspects of the problem one should describe the above mentioned control system rather by combined simulation techniques than by purely continuous simulation. This gives the required motivation for our redefinition of combined and continuous simulation stated in the introduction. (Note that the example would definitely belong to the class of continuous systems according to the "common" use of the term.) Considering the aspect of information processing CSMP obviously offers modeling elements (such as a switch function) which it is unable to preprocess into properly executable code. Again this is not a problem of the language CSMP, but holds for all CSSL-type languages. Some of the languages (like DARE-P [18]) are somewhat more modest in the facilities they offer in this respect, for which they are blamed by many users. The author, however, believes that it is more faithful to offer few facilities than to write a beautiful manual offering many nice features, which effectively cannot be properly used. On the other hand the reaction of these users proves that the facilities are useful and needed. For this reason the author feels, that in a future revision of the CSSL specifications [36] combined simulation facilities should be taken into account, opposing herein to the opinion expressed in [1]. (A revision of the CSSL specifications will be necessary anyway, if for no other reason, the original definition contains over 40 syntactical errors as shown in [22].)

As can be seen from the previously stated: The problem of combined simulation can be subdivided into the numerical aspects (executability of the run-time system) and into the aspects of information processing (definition of the descriptive input language). These two problems are now to be considered more carefully.

IV) NUMERICAL ASPECTS:

IV.1) Structure of the run-time package:

Experience has shown that for the execution of combined simulation the following concept is to be used: A combined problem may be subdivided into

a) a discrete part consisting of all elements used for discrete simulation,

b) a continuous part consisting of all elements used for continuous simulation and

c) an interface part describing the conditions when to switch from (a) to (b) and vice-versa.

During the execution of a combined simulation we are, therefore, either performing entirely discrete simulation (with its well known properties) or entirely continuous simulation (with its also well known properties), whereas execution of simultaneously combined continuous and discrete simulation does not exist. Thus a combined simulation run-time package must be composed of

a) a discrete simulation run-time package,

b) a continuous simulation run-time package and

c) some algorithms describing the activities to be taken, when branching from (a) to (b) or vice-versa is required.

The numerical requirements for the subsystems (a) and (b) are both well known and discussed on many occasions and, thus, need not be considered here again. An excellent survey of the major simulation systems for problem class (a) is [19], whereas the problem class (b) is surveyed in [1,4,20]. Once this structure has been understood we can restrict ourselves merely to combining previously developed software for discrete and continuous simulation to obtain a good run-time package for combined simulation as well.

In the following we will restrict our view on subsystem (c).

A) Conditions for changing to continuous simulation when executing discrete simulation:

Let the simulation clock be advanced to event time t1. Executing discrete simulation means that the system is about to perform event-handling at time t1. We have to execute discrete simulation until all events scheduled for time t1 have been performed. We have then to switch to continuous simulation if there are differential equations currently involved in the combined simulation (for some intervals of time there may be none). Otherwise we reset the simulation clock to the next event time and continue with executing discrete simulation until again there are no events left to be performed for this new event

time. Therefore, no special algorithms need to be developed for this case. After event handling being performed the integration algorithm needs to be restarted. This is especially important in case multi-step methods are being used.

B) Conditions for changing to discrete simulation when executing continuous simulation:

Continuous simulation has to be performed either up to the next scheduled event time (for time-events) or until a state-condition is met triggering execution of a state-event, whichever comes first. In both cases the step-size control mechanism of the integration algorithm has to be disabled. In the former case (handling of a time-event) the step-size simply has to be reduced down to the scheduled event time, in the latter case (handling of a state-event) a new step-size control algorithm must be activated for iteration of the solution to the unknown event-time. Again these algorithms are not really new. Any good iteration procedure (like Newton-Raphson) can solve the problem. The author recommends a combination of the inverse Hermite interpolation (fast convergence) with Regula-falsi (unlimited convergence range). This iteration scheme has been described in detail in [7].

C) Selection of the initial subsystem:

Having discussed the conditions for branching from (a) to (b) and vice-versa it remains to determine the subsystem to be used first at initialization time t0. The rule is simply to start with the discrete subsystem. This will then check whether there are any events to be treated at time t0 and if not transfer control to the continuous simulation package in which case the taken activity would be none (this under the assumption that differential equations form part of the system's description at time (t0+dt), otherwise proceed as described above).

IV.2) Unsolved problems:

In the author's opinion the best among existing coded packages utilizing the ideas above is GASP-V [7,8]. Many problems have been tested using this software and the results were quite promising. There are still two unsolved problems:

A) Taking the definition of Pritsker [27] for event times:

> *"An event occurs at any point in time beyond which the status of a system cannot be projected with certainty"*

it is clear that an infinite density of events must not occur. This may, however, happen in at least the following two cases:

a) A system is modeled by a set of PDE's and discontinuities exist. In this case the discontinuity may "walk" through space with the time and can no longer be localized in the

way proposed in section IV.1. As an example let us consider a long electric wire where a current is imposed at one end which suddenly (at time t1) changes its value. This discontinuity will remain in the system for some time and "flow" through the wire. If there are effects of reflection assumed at the other end, it may even remain in the system forever. Thus, in this example we will find, that for any instant of time t t1 the system will be discontinuous at one particular point in space (x1) which is moving around.

b) The behaviour of the continuous subsystem is stochastic in nature. The spectrum of a random number stream has infinite frequencies which has the effect that it is nowhere differentiable. If such a random number is superposed to the input of an integrator, we face the problem mentioned above. This holds of course only for stochastic behaviour of the continuous subsystem and not for the discrete subsystem. Stochastic interarrival times of customers to a queue, for instance, will not effect the numerical behaviour of the system, since new samples for the random numbers are only computed at event times, whereas in between these variables are constant.

Zeigler [34, Chapter 9] has shown that the existence of an infinite density of events always results in an illegitimate model. In the case of (a) it is, theoretically, always possible to respecify the model so that the new equivalent model is no longer illegitimate. In this new formulation the propagation of discontinuities will follow the axes of the coordinate system. This is well known as the "method-of-characteristics". In the case of the linear wave equation we know that the characteristics are straight parallel lines and the required variable transformation is easy to achieve. For complex situations (nonlinear cases), however, to find the characteristics of the problem (which are now curves bended in time and space) is almost equivalent to solving the entire problem. Thus while we can solve the problem (a) theoretically, in practice the required computations for obtaining the variable transformations are extremely tedious and may prevent us from doing so.

Therefore, we usually find another solution for this problem: In using the method-of-lines approach [3,6] we found that the integration over time is not much effected in most applications by these discontinuities whereas the computation of the spatial derivatives is heavily disturbed. Therefore, we first try to identify (for each step) in which discretization interval the discontinuity is situated at the moment, then we split up the region and compute the spatial derivatives independently for the two parts lying to the left and to the right of the discontinuity. This procedure can easily be expanded for several space dimensions as well.

The case of (b) is in principle more difficult to treat. Zeigler's characterization of illegitimate models was developed only for discrete event models. However, his discussion of the intrinsic limitation of the class of continuous systems which can be simulated by digital computers [34, Chapter 5] and [35]

may be applied to the present problem. According to this analysis, there must always be a non zero interval separating computer updates of the model's state. Thus the computer must guess what the behaviour of the model is in the interval separating computational instants -- the problem of "bridging the gap". Since the computer is given a description of the model components and their coupling it can guess correctly only if certain conditions enabling perfect interpolation in the gap hold. Polynomial trajectories, commonly assumed in integrating differential equation models, serve this purpose.

In the case of stochastic continuous models it is not easy to justify the assumption of polynomial trajectories. For example, if the model contains a white noise component then no means of bridging the gap exist in principle. This is because, by definition, the correlation between sample values, however closely spaced in time, is zero. Even if the noise is not white, current numerical methods are not geared to exploiting autocorrelations specified by the model for optimum choice of integration step. As a result, most step size control algorithms will produce extremely pessimistic guesses for the step sizes to be used, resulting in high computational costs.

We found the following approach useful in many applications: First we compute one run by setting the noise to zero using variable step integration (the continuous subsystem is now deterministic). In this run we collect statistics (histogram) of the utilized step sizes dt. From the cumulative frequency curve we select the 0.1 level point (10% of the step sizes fall below this point). Now we compute a new run, this time with inclusion of the noise, where we keep the step size dt fixed at this 0.1 level point. A disadvantage of this solution is, of course, that we now have no measurement for the quality of the approximation. We must thus be very careful in the interpretation of results obtained in this way. Furthermore, the proposed method can be applied only, if the signal/noise ratio is high. For a low signal/noise ratio we do not know any good numerical technique to go round this problem.

B) As explained by Elzas [12] the user of a simulation package wishes either to obtain reliable results or have a "bell" ring when an algorithm is unable to perform proper work. Under no circumstances does he want to obtain results which are wrong. So far this can be guaranteed with a high confidence in the case of ODE problems only, whereas for PDE problems numerical difficulties need not necessarily be detected by the package, resulting in inaccurate or even entirely wrong results. More research needs to be devoted to this problem. It arises from the fact that we always use a fixed grid for the spatial discretization and thus have no control on the error resulting from this discretization. Adaptive algorithms would be required (similar to the variable-step algorithms for numerical integration) to solve this problem. Some attempts have been made in several places to find a solution, but these have not been successful so far. Also in this respect, packages like GASP-V [7,8] or FORSIM-VI [3] may prove very useful, since they allow to design new experiments in a very flexible and simple manner.

V) ASPECTS OF INFORMATION PROCESSING:

So far we have discussed the numerical behaviour of a run-time system able to perform combined simulation. Now it remains to question what is the easiest and most convenient way for the user to formulate combined problems to the computer, so that the computer will be able to produce properly ececutable run-time code. For this we will have to identify the structural elements of combined simulation languages.

V.1) The elements of the language:

A combined simulation language will primarily consist of the well known elements of continuous and discrete simulation languages. There are few additional elements required to weld these two subsystems together.

A) The state-event:

The only essential new element is the state-event describing conditions of the continuous subsystem status when to branch to the discrete subsystem. A typical situation is illustrated in the following:

> *When the angular velocity of a DC-motor crosses a threshold of 1500 RPM in the positive direction, the motor has to be loaded.*

This situation could, for example, be coded using a 'CONDIT'-statement in the continuous subsystem:

```
CONTPROCS
  ...
  CONDIT EV1: OMEGA POS CROSS
         1500.0 TOL=1.0E-3 END;
```

and the reaction to this would then be coded by an event description in the discrete subsystem:

```
DISCPROCS
  EVENTS
    EV1: TL := 200.0 END;
    ...
```

(the torque load (TL) is to be reset to 200.0). The CONDIT-statement is similar to a CSMP FINISH condition, except that the time of the crossing is iterated until a prespecified tolerance is met (TOL=1.0E-3), and in that the simulation run is not terminated, but control is handed over to the discrete simulation system. After event handling as described by the discrete subsystem (DISCPROCS) control is returned to the continuous subsystem (CONTPROCS) where the new value of TL will be used somewhere on the right hand side of the equal sign.

B) Operations of the continuous subsystem on the discrete subsystem:

There are none.

C) Operations of the discrete subsystem on the continuous subsystem:

It is most commonly found that not only parameters of the continuous subsystem (as the torque load TL above) change their values at event times but that some of the equations are replaced by others. This situation can be taken care of by the following language elements:

a) The "one out of n" situation:

There are n possible "models" out of which one is always active. This situation can best be expressed by a CASE-statement:

```
CASE NMOD OF
```

where NMOD is an integer number pointing to the currently active model. This language element is used in general to describe n different functional ways of behaviour of one model component.

b) The "k out of n" situation:

Another frequently found situation is illustrated by the following example:

There are n cars in a system, out of which k are moving around and (n-k) are parked somewhere.

This situation can be represented by the following syntactical construct:

```
FOR I:=1 TO N DO
  IF CAR[I] THEN
```

where CAR is a boolean array with the values "true" for cars moving around and "false" for parked cars.

For n = 1 this case degenerates to a simple IF clause.

c) Example:

Let us consider a mechanical system with a dry friction force (TFR) modeled somewhere in the system. This can be shown by the following graph:

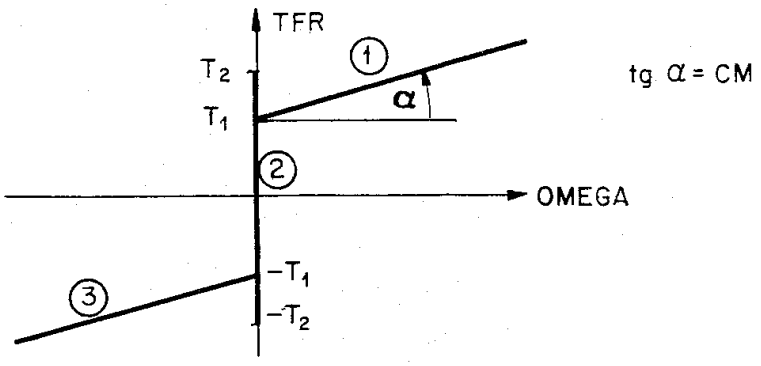

Fig.2: Dry friction force graphed versus velocity

In this example we face the typical "one out of n" situation, where n = 3 are the three continuous branches of the discontinuous TFR-function. Each of them is represented by a different equation and by a different set of state conditions.

This situation could thus be coded as shown in Fig.3. Using this formalism for describing a combined system, the resulting description is not much more complicated than for normal CSSL-type languages, but allows the preprocessor to produce properly executable run-time code.

V.2) Requirements of the language:

When developing a new language for combined system simulation the following points should be remembered:

a) The language should provide for flexible structures.

b) It should be extendable (open-ended operator set).

c) Both syntax and semantics of the language should be easy to learn and to remember.

d) The language should contain as few elements as possible but as many as are required.

e) Models should be codable by as few elements as possible.

f) The preprocessor should contain provisions for faithfully detecting coding errors.

```
SYSTEM
  CONTPROCS
    ...
    ...
    MODEL DRYFRICTION (TFR <- T, OMEGA)
    (* COMMENT: <- SYMBOLIZES A LEFT ARROW AND IS USED TO
                 SEPARATE INPUT FROM OUTPUT VARIABLE LISTS *)
      CASE NL OF
        1:  TFR = T1 + CM*OMEGA;
            CONDIT MOD2: OMEGA NEG CROSS + 0.0 TOL=1.0E-3 END
            END;
        2:  TFR = T;
            CONDIT MOD1: T POS CROSS + T2 TOL=1.0E-3 END;
            CONDIT MOD3: T NEG CROSS - T2 TOL=1.0E-3 END
            END;
        3:  TFR = -T1 + CM*OMEGA;
            CONDIT MOD2: OMEGA POS CROSS + 0.0 TOL=1.0E-3 END
            END
    END (* DRY FRICTION *)
    ...
  END (* CONTINUOUS SUBSYSTEM *)
  DISCPROCS
    EVENTS
      MOD1:  NL := 1 END;
      MOD2:  NL := 2; OMEGA := 0.0 END;
      MOD3:  NL := 3 END
    END (* TIME EVENTS DESCRIPTION *)
  END (* DISCRETE SUBSYSTEM *)
END (* SYSTEM DESCRIPTION *)
```

Fig.3: Combined description of a dry friction force

g) The language must contain all elements required to enable the preprocessor to produce numerically well-conditioned run-time code.

Some of these requirements are contradictory. For example: If we want to enable the preprocessor to detect as many errors as possible, made by the user when formulating his model, the language must contain some redundancy. This certainly competes with the wish to have as short user's programs as possible.

Flexible structures: We want to obtain a universality of the program's applicability. This problem has been considered carefully when designing the CSSL-type languages which are in existence and also in designing the SIMULA-67 language for discrete simulation [10]. So we need not discuss this here again.

Extendability: Two different points must be considered: On one hand, we want to enable the user of such a language to extend it for his personal needs. This requires a superposed macros-structure [4,9]. This, however, need not necessarily form an integral part of the language definition, as it is interpreted by the preprocessor. The macros-handler can as well be realized by a stand-alone program preceeding the normal preprocessing [5]. This would then allow for more flexible macros-structures (interpretative macros-language) without

calling for too high core memory requirements for its realization. On the other hand the system's specialist should also be given the possibility to extend the basic language definition itself. For this purpose the preprocessor should be constructed in such a way that it can be easily augmented to accommodate new ideas. For this purpose the most recent compiler building techniques employing structured programming and structured data representation should be applied, as described for instance by Wirth [33]. The author suggests, therefore, to design the language as much as possible as a deterministic, one pass, left to rigth language (DLR-1 language), for which the syntax is to be described formally either by use of a BNF-notation or by use of syntax diagrams. Compilers for such languages can be written in a straight forward manner and are, thus, easily readable.

Ease of learning syntax and semantics: Two main goals are to be achieved: It should be easy to write programs by one's self and it should as well be easy to read programs coded by somebody else. These two goals tend to compete with each other. To meet the former goal we want the different elements of the language to use the same syntactical constructs as much as possible. To meet the latter goal we want to be as flexible as possible in choosing appropriate mnemonics and close to conversational english constructs.

To give an example for the competing nature of the two goals let us consider once more the dry friction example given above (Fig.3). To meet the second goal we introduce the '=' symbol in the notation of equations of the parallel section and the ':=' symbol in the notation of statements of the procedural section. By these means the inherent difference between parallel and procedural sections is clarified, in that, for instance,

```
I := I + 1;
```

is a meaningful statement, whereas

```
I = I + 1;
```

is a meaningless equation. This rule will thus help to improve the readability of programs, it will at the same time, however, complicate the writing of programs since it simply introduces an additional not necessarily required syntactical construct to remember.

Few language elements: The language should be constituted of as few elements as possible to make it easily learnable. On the other hand we require many language elements to obtain short user's programs. If there are not enough primitives offered by the language, the coding of complex problems becomes very difficult and the resulting source program will be long. This problem is best solved by providing a hierarchical structure of both language and documentation. By means of this the user can first read an introductory manual which teaches him how to utilize the basic features required for modeling simple problems. This can be learned in a short time. Later on, when he realizes that his problem is more intricate than he thought in the beginning he may study another manual which enables him to use advanced features of the language. The user must be able to code simple situations in a simple manner, but should be assisted when

coding more complex situations as well.

Short users' programs: The user should not be bothered by being asked to provide unnecessary information (like typing FORTRAN COMMON-blocks). This point must be balanced against:

Provisions for error detecting: Some redundancy should be left to the program for error detecting purposes. The author feels that a modern simulation language should for example require from the user that all variables he is going to utilize are declared in the beginning. This enables the preprocessor to detect many typing errors. This statement has been mentioned on many occasions (e.g. in the development of PASCAL [16]). It is, therefore, amazing, that none of the CSSL-type simulation languages to our knowledge has adopted this idea, and that this fact is even praised by many developers of brand new simulation software.

Well-conditioned run-time code: This point has been discussed in section V.1 already.

VI) DISCUSSION OF EXISTING SOFTWARE:

The existing software for combined simulation has recently been reviewed in an excellent survey by Oren [23,24]. He has collected information on about 30 different simulation systems for combined system simulation. Due to the fact that combined simulation never before had been properly defined, some of the surveyed languages/packages lie on a somewhat different line from what has been presented here. Furthermore, some of the programs described have never been released. Considering only those programs being implemented at several different installations and being widely used by different people, just a very few of them remain. Among these, GASP-IV [27] which is an ANSI-FORTRAN-IV coded subroutine package, has by far the largest distribution. Together with its descendent, GASP-V this program follows the ideas mentioned in section IV. The numerical behaviour of the GASP software is, in our experience, the best of all existing run-time packages for combined simulation. Unfortunately there is no provision in GASP for a user-oriented input definition (no preprocessor is involved). Therefore, none of the ideas presented in section V is realized in GASP.

From the information processing point of view, pioneer work has been done in the definition of the language GSL [15] following the original ideas of Fahrland [13,14]. GSL has the best language structure of all the languages so far published. Unfortunately GSL has never been released and must therefore be considered to be a collection of new ideas rather than a simulation language. However, GSL also has severe shortcomings both in the underlying run-time structure and in the language definition itself (the recent developments in the field of information processing and especially compiler building, as presented in section V, have not sufficiently been taken into account). A new simulation language COSY [2] (standing for COmbined SYstems) is under development by the author. It will involve a PASCAL-coded preprocessor which translates a new input definition language (following the ideas developed in section V of this paper) into GASP-V

executable code. GASP-V thus will be used as target language for COSY. This language, however, will not be released before the end of 1979. Another new language under development which has some resemblance to COSY is GEST [21]. The language definition will soon be replaced by the newer version GEST'78 [25,26].

An entirely different approach has been taken in the definition of SMOOTH [32], which uses a network approach combining GERT networks for discrete simulation [28] with STATE networks for the continuous subsystem. A new program of this class will be released in 1979 by Pritsker combining Q-GERT [28] with GASP-IV/E (extended version of GASP-IV, released in 1978 by Pritsker). This approach certainly results in extremely short application programs, at least for such applications for which there are elements provided in the language. However, a network approach cannot be as general as a structured language. Benyon [1] states: "Such a diagrammatic approach to modelling can be very useful in some instances, but experience with the continuous languages has been that the diagrams soon grow too complicated to be enlightening, once one advances beyond quite simple models".

Moreover, it is often stated that network languages are easier to learn compared to equation oriented structured languages. The author would deny this statement for two reasons:

a) The number of language elements of such block oriented languages required to obtain at least a certain degree of flexibility is much larger than for equation oriented languages. GPSS-V [31], for example, consists of 41 building blocks and Q-GERT [28] offers 24 of them (Q-GERT requires a smaller number of blocks for an even higher degree of flexibility, because the single building blocks are more decomposible and recombinable). All of these building blocks must be understood before a truly complex program can be written.

b) Since the single building block describes a rather complex entity compared to a simple event description the semantics required to describe such an element are much more complex. (A complex situation can either be expressed by a complex syntax consisting of many "small" building blocks with primitive semantics, or by a simple syntactical construct consisting of few "large" building blocks with complex semantics, but never by both simple syntax and semantics.) This can be illustrated with a simple example. Considering the GENERATE-block of GPSS, it seems first, that the meaning of this block is very easily explained.

GENERATE A, B

means that a new transaction is to be generated with a uniform distribution in the interval [A-B,A+B]. The novice user of GPSS will take this definition and let this block be followed by a SEIZE-block to have the transaction occupying a facility. In practice, if this facility is already occupied when the transaction is born to the system, this transaction will stay in the GENERATE-block and inhibit the generation of new transactions. This shows, that the semantics required to describe this simple

situation properly, are much more complex than might be thought. Very commonly, an error occurs due to the fact that semantics are involved which have not been reported to the user in the introductory manual.

This last example unveils another weak point of network languages: The program as specified above will "work", which means that there will be output produced, although this output will be wrong. With a high probability the user will thus never detect that his program is erroneous. The reason for the inability of GPSS to detect the error lies in the fact, that hardly any redundancy has been left in the code which could enable the system to detect errors in the source program.

Much more promising, it seems to us, is a new network approach proposed by Elmqvist [11] and by Runge [29]. Intended for continuous systems, it could also be extended to encompass discrete systems as well. This new approach has its background in the equation oriented languages. The single network element consists of a set of equations programmed by the user. Different modules are connected by special elements, called cut- and path- elements in DYMOLA [11]. This new approach can be thought of as an extension of the earlier macros-constructs of CSSL. It is even more general than the classical CSSL-type language, since this latter forms a subset of the new network language. In this approach the user does not need to describe in advance which connecting variables of his submodel (variables which are visible from outside) are input and which are output variables of the module. Formulae manipulation algorithms are used in DYMOLA to obtain a computational set of statements, whereas MODEL [29] uses implicit integration techniques to go round the difficulty. Some problems may arise from the fact that these languages are no longer context-free. They are on the contrary extremely context-dependent. This sometimes may result in ambiguities which have to be resolved. The possible set of executable statements is not necessarily unique. The author believes, however, that it will be worthwhile devoting more research to this approach.

VII) ACKNOWLEDGMENTS:

The author would like to express his deep indeptedness towards Prof. A. Alan B. Pritsker. Many of the ideas expressed in this article originated from the pioneer work in combined simulation done by Pritsker and his group, and also resulted from personal discussions with them. He would furthermore like to thank Prof. Tuncer Oren for the many good ideas he suggested in several long discussions of the subject. He is also very grateful to Prof. Bernard P. Zeigler for the stimulating comments obtained as a reaction to the initial abstract sent.

VIII) REFERENCES:

[1] P.R.Benyon: (1976) "Improving and Standardizing Continuous Simulation Languages". Proc. of the SIMSIG Simulation Conference, Melbourne, Australia, May 17-19, 1976; pp. 130 - 140.

[2] A.Bongulielmi: (1978) "Definition der allgemeinen Simulationssprache COSY". Semesterwork, Institute for Automatic Control, The Swiss Federal Institute of Technology Zurich. To be obtained on microfiches from: The main library, ETH - Zentrum, CH-8092 Zurich, Switzerland. (Mikr. S637).

[3] M.B.Carver: (1978) "The FORSIM-VI Simulation Package for the Automated Solution of Arbitrarily Defined Partial and/or Ordinary Differential Equation Systems". Form: AECL-5821. Atomic Energy of Canada, Ltd.; Chalk River Nuclear Laboratories, Mathematics & Computation Branch, Chalk River, Ontario, Canada K0J 1J0.

[4] F.E.Cellier: (1975) "Continuous-System Simulation by Use of Digital Computers: A State-of-the-Art Survey and Prospectives for Development". Proc. of the SIMULATION'75 Symposium, Zurich. To be obtained from: ACTA Press, P.O.Box 354, CH-8053 Zurich, Switzerland; pp. 18 - 25.

[5] F.E.Cellier: (1976) "Macro-Handler for Simulation Packages Using ML/I". Proc. of the 8th AICA Congress on Simulation of Systems, Delft, The Netherlands. Published by North-Holland Publishing Company (Editor: L.Dekker); pp. 515 - 521.

[6] F.E.Cellier: (1977) "On the Solution of Parabolic and Hyperbolic PDE's by the Method-of-Lines Approach". Proc. of the SIMULATION'77 Symposium, Montreux, Switzerland. To be obtained from: ACTA Press, P.O.Box 354, CH-8053 Zurich, Switzerland; pp. 144 - 148.

[7] F.E.Cellier: (1978) "The GASP-V Users' Manual". To be obtained from: Institute for Automatic Control, The Swiss Federal Institute of Technology Zurich, ETH - Zentrum, CH-8092 Zurich, Switzerland.

[8] F.E.Cellier, Blitz A.E.: (1976) "GASP-V: A Universal Simulation Package". Proc. of the 8th AICA Congress on Simulation of Systems, Delft, The Netherlands. Published by North-Holland Publishing Company (Editor: L.Dekker); pp. 391 - 402.

[9] F.E.Cellier, Ferroni B.A.: (1974) "Modular, Digital Simulation of Electro/Hydraulic Drives Using CSMP". Proc. of the 1974, Summer Computer Simulation Conference, Houston, Texas, U.S.A.; pp. 510 - 514.

[10] O.J.Dahl, Nygaard K.: (1966) "Simula; A Language for Programming and Description of Discrete Event Systems". Oslo, Norwegian Computing Center.

[11] H.Elmqvist: (1978) "A Structured Model Language for Large Continuous Systems". Form: CODEN LUTFD2/(TFRT-1015)/1-226/(1978). Ph.D Thesis. Lund Institute of Technology, Dept. of Automatic Control, Lund, Sweden.

[12] M.S.Elzas: (1978) "What is Needed for Robust Simulation?" Article in this volume.

[13] D.A.Fahrland: (1968) "Combined Discrete Event / Continuous System Simulation". MS Thesis, Systems Research Center Report SRC-68-16, Case Western Reserve University, Cleveland, Ohio.

[14] D.A.Fahrland: (1970) "Combined Discrete-Event Continuous System Simulation". Simulation vol. 14 no. 2 : February 1970; pp. 61 - 72.

[15] D.G.Golden, Schoeffler J.D.: (1973) "GSL - A Combined Continuous and Discrete Simulation Language". Simulation vol. 20 no. 1 : January 1973; pp. 1 - 8.

[16] K.Jensen, Wirth N.: (1974) "PASCAL User Manual and Report". Lecture Notes in Computer Science, Springer Verlag.

[17] P.J.Kiviat: (1967) "Digital Computer Simulation : Modeling Concepts". Form: RM-5378-PR, The Rand Corp., Santa Monica, CA, U.S.A..

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

[19] W.Kreutzer: (1976) "Comparison and Evaluation of Discrete Event Simulation Programming Languages for Management Decision Making". Proc. of the 8th AICA Congress on Simulation of Systems, Delft, The Netherlands. Published by: North-Holland Publishing Company (Editor: L.Dekker); pp. 429 - 438.

[20] R.N.Nilsen, Karplus W.J.: (1974) "Continuous-System Simulation Languages - A State-of-the-Art Survey". Annales de l'Association Internationale pour le Calcul Analogique (AICA), No. 1, january 1974; pp. 17 - 25.

[21] T.I.Oren: (1971) "GEST: A Combined Digital Simulation Language for Large Scale Systems". Proc. of the AICA Symposium on Simulation of Complex Systems, Tokyo, Japan, September 3-7, 1971; pp. B-1/1 - B-1/4.

[22] T.I.Oren: (1975) "Syntactic Errors of the Original Formal Definition of CSSL 1967". Technical Report TR75-01 (IEEE Computer Society Repository No. R75-78), Computer Science Dept., University of Ottawa, Ottawa, Canada.

[23] T.I.Oren: (1977) "Software for Simulation of Combined Continuous and Discrete Systems: A State-of-the-Art Review". Simulation, vol. 28 no. 2 : February 1977, pp. 33 - 45.

[24] T.I.Oren: (1977) "Software Additions". Simulation, vol. 29 no. 4 : October 1977, pp. 125 - 126.

[25] T.I.Oren: (1978) "Reference Manual of GEST'78 - Level 1 (A Modeling and Simulation Language for Combined Systems)". Technical Report 78-02, Computer Science Dept., University of Ottawa, Ottawa, Canada.

[26] T.I.Oren, den Dulk J.A.: (1978) "Ecological Models Expressed in GEST'78", Technical Report Prepared for the Dept. of Theoretical Plant Ecology, Dutch Agricultural University Wageningen, The Netherlands.

[27] A.A.B.Pritsker: (1974) "The GASP-IV Simulation Language". John Wiley.

[28] A.A.B.Pritsker: (1977) "Modeling and Analysis Using Q-GERT Networks". John Wiley.

[29] T.F.Runge: (1977) "A Universal Language for Continuous Network Simulation". Form: UIUCDCS-R-77-866. Ph.D Thesis. University of Illinois at Urbana-Champaign, Dept. of Computer Science, Urbana, Ill., U.S.A..

[30] H.Schlunegger: (1977) "Untersuchung eines netzrueckwirkungsarmen, zwangskommutierten Triebfahrzeug-Stromrichters zur Einspeisung eines Gleichspannungszwischenkreises aus dem Einphasennetz". Ph.D Thesis, no. DISS.ETH.5867: The Swiss Federal Institute of Technology Zurich, Switzerland.

[31] T.J.Schriber: (1974) "Simulation Using GPSS". John Wiley.

[32] C.E.Sigal, Pritsker A.A.B.: (1973) "SMOOTH: A Combined Continuous/Discrete Network Simulation Language". Proc. of the 4th Annual Pittsburgh Conference on Modeling and Simulation. Pittsburgh, Penn., U.S.A., April 23-24, 1973, pp. 324 - 329.

[33] N.Wirth: (1976) "Algorithms + Data Structures = Programs". Prentice Hall, Series in Automatic Computation (Chapter 5).
or:
N.Wirth: (1977) "Compilerbau". Teubner Studienbuecher, Informatik.

[34] B.P.Zeigler: (1976) "Theory of Modelling and Simulation". John Wiley.

[35] B.P.Zeigler: (1977) "Systems Simulateable by the Digital Computer". Logic of Computers Group Report, University of Michigan, Ann Arbor, U.S.A..

[36] (1967) "The SCi Continuous System Simulation Language (CSSL)". Simulation, vol. 9 no. 6, December 1967; pp. 281 - 303.