CONFERENCE ON CONTINUOUS SYSTEM SIMULATION LANGUAGES

Proceedings of the Conference on
Continuous System Simulation Languages

23-25 January 1986
San Diego, California

Edited by
Francois E. Cellier, PhD
University of Arizona

A Society for Computer Simulation (Simulation Councils, Inc.) publication.
San Diego, California

Enhanced run-time experiments for continuous system simulation languages

Dr. François E. Cellier
Associate Professor
Dept. of Electrical & Computer Engineering
The University of Arizona
Tucson, Arizona 85721

Abstract

In recent years, CSSL's have seen a drastic improvement of their capabilities. Most of these enhancements are concerned with:

a) **modelling**: enhanced power of model building and model structuring facilities including specification of submodels, model bases (data bases to store models and submodels), and proper discontinuity handling (combined continuous/discrete simulation)

b) **compilation**: enhanced compilation techniques including separate compilation of submodels, and direct executing languages (faster response in an interactive execution environment)

c) **execution**: enhanced integration techniques including new algorithms for stiff systems, highly oscillatory systems, linear systems, and techniques for state-space partitioning (faster execution of large-scale systems).

However, very little has happened with respect to the flexibility in experimenting with the simulation model. Basically, only two «experiments» are available in the classical CSSL:

a) simulation from time 0 to time t_f (expressed by: FINTIM, TMAX, ...)

b) simulation from time 0 until something happens (expressed by: FINISH, TERMINATE, ...).

This is not what we like to find. Ideally seen, we should be able to design our experiment entirely separate from the model itself. With the modelling stage already completed, we would like to decide on the experiment to be performed, such as the nature of the variables to be monitored during simulation, evaluation of steady-state points, derivation of linearized models and their eigenvalues, execution of complete sensitivity analyses, or replication analyses, etc..

DARE-INTERACTIVE is a new dialect within the DARE family of simulation languages which has been designed and implemented as a testbed for enhanced and more intelligent run-time experiments. With one single command, it is already possible to perform a complete range analysis (sensitivity analysis for nonlinear models), or a complete replication analysis. The graphics postprocessor allows to view families of trajectories either by use of envelope graphics or threedimensional graphics with hidden lines removed. Envelope graphics allow to view the range of a variable as a function of parameter incertainties, instead of a particular trajectory, displayed with fourteen digits accuracy out of which possibly not a single one may be significant (!) Other experiments such as automated steady-state finding, and curve fitting are currently under development.

In this paper, we wish to summarize what possibilities exist with respect to enhancing the capabilities of simulation run-time experimentation. At the same time, we shall show practical examples of implementation of some of these ideas in the DARE-INTERACTIVE software system.

Sensitivity Analysis

Sensitivity Analysis describes the behavior of a (linearized) simulation model in the neighborhood of an operating point. The sensitivity model determines the derivatives of the state variables with respect to the parameters under investigation.

Eventhough sensitivity analysis has been developed for the analysis of linear systems, it can easily be extended to large classes of nonlinear systems as well. In this case, however, the results are not global, but local to the neighborhood of an operating point. Nonlinear systems can be linearized in the neighborhood of their operating points as long as no discontinuities take place as part of the operating behavior (e.g. bang-bang control), and as long as the nonlinearities do not essentially govern the behavior of the system in its operating environment (e.g. oscillators).

So far, sensitivity analysis was always performed either by numerical approximations at run-time, or by manually adding the sensitivity model to the set of differential equations describing the system behavior. The former procedure is inefficient and inaccurate while the latter is tedious, as for each parameter, there have to be added another *n* differential equations where *n* is the model order.

It is, however, perfectly possible to automate the generation of the sensitivity model at compile time. For that purpose, we require an **algebraic differentiation processor** that decomposes complicated expressions into sets of primitive expressions, and adds the corresponding derivatives to them. As an example, the expression:

$$y = x*\sin(x^2)$$

is decomposed into the set of primitives:

$$t_1 = x^2$$
$$t_2 = \sin(t_1)$$
$$y = x*t_2$$

The sensitivity model is then added as:

$$dt_1 = 2*x$$
$$dt_2 = \cos(t_1)*dt_1$$
$$dy = t_2 + x*dt_2$$

assuming that *x* is the parameter with respect to which we want to evaluate the sensitivity. *dy* is then the sensitivity equation. As this example shows, each primitive expression can be treated separately by a simple table look-up.

We currently have available an ALGOL-coded program that can compute the derivatives of any ALGOL procedure with respect to any variable or array of variables. As most simulation languages use FORTRAN as their target code, we currently develop a new version of that software which is PASCAL-coded, and allows to compute derivatives of any FORTRAN subroutine or set of FORTRAN subroutines with respect to any variable or array of variables. This program shall be integrated into DARE-INTERACTIVE at a later stage.

Range Analysis

Range Analysis describes the behavior of a (nonlinear) simulation model over an operating range of a set of model parameters. Range Analysis is Sensitivity Analysis in the large.

Sensitivity analysis describes nonlinear models only in the neighborhood of an operating point. How large the neighborhood may be without invalidating the results is a question that is generally not easy to answer. Range analysis solves that problem to some extent. Here, the simulation is repeated many times with parameters varying over their operating ranges. For practical purposes, only the extreme values are tested, that is: for *k* parameters having tolerances associated with them, we perform 2^k simulation runs for all worst case combinations. Of course, in a nonlinear model, we cannot know for sure that the worst case takes place at the borders of the parameter ranges, but this assumption makes the problem treatable in an orderly fashion, and, if the tolerances are sufficiently small, it is probably justified.

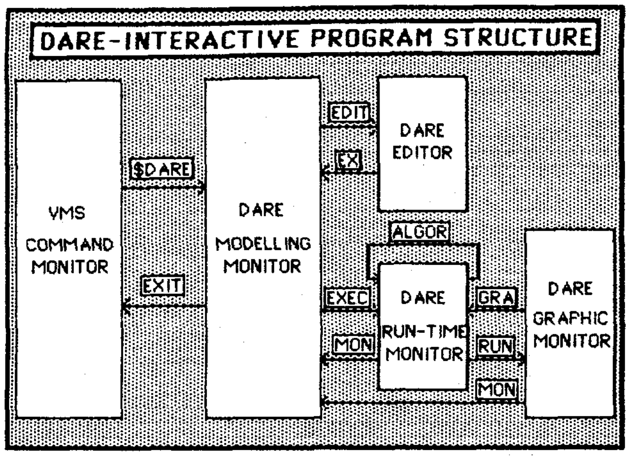
For demonstration purposes, let us look at Forrester's world model. We assume that four parameters:

NRUN:	**Natural resources «normal» consumption**
POLN:	**«Normal» pollution**
FRN:	**«Normal» food ratio**
CIAFN:	**«Normal» percentage of resources allocated to the agricultural sector**

have tolerances of *25%* associated with each of them. In DARE-INTERACTIVE, parameters are stored in a separate parameter file which is interactively generated by the run-time monitor prior to the first execution of any simulation run. Any parameter may carry tolerances which do not influence the simulation during normal runs, but which are interpreted by the system when a range analysis is performed. The range analysis is executed either by specifying **CALL RANGE** instead of **CALL RUN** in the logic block of the DARE program, or by using the interactive run-time command **RANGE** in place of **SIMULATE**. *Fig. 1* shows the program flow of DARE-INTERACTIVE.

DARE-INTERACTIVE has three different modes in which the user can interact with the system:

a) **the modelling monitor**: in which the user specifies his model to the system



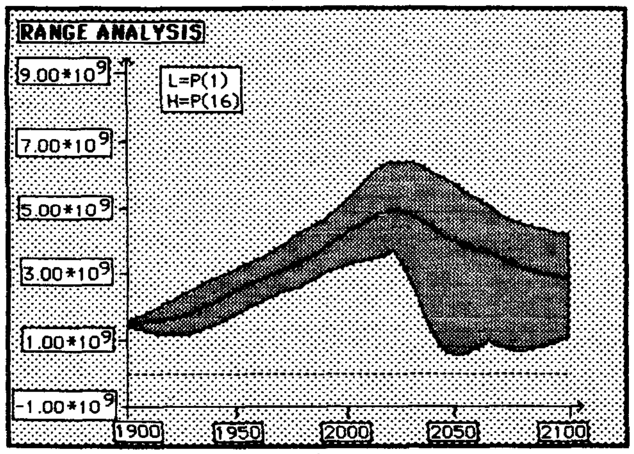
b) **the run-time monitor**: in which the user describes his experiment (e.g. parameter values, run-time display, integration algorithm, type of experiment (normal simulation, range analysis, replication analysis)

c) **the graphics monitor**: in which the user interacts with the simulation data-base to view results of one or several previously performed simulations.

In our example, we performed a range analysis in the run-time mode. Automatically, *17* different simulation runs were executed (the *16* worst cases plus one «control» run with the standard parameter settings), which were stored in *17* different SAVE records. We then went to the graphics processor to view the envelope of the population trajectories of the 16 worst case runs with the *17th* curve as a reference curve by specifying:

envelope(:blue:yellow),p(1:red),p(16),p(17:black)

which generated *Fig.2*:



As none of the parameters was an initial condition, all runs obviously start at the same point. However, they vary largely in their subsequent behavior. For the project manager, envelope graphics are much more meaningful that the regular trajectory behavior, as he can judge much better how his system is really going to perform. A large range indicates large sensitivities, that is: the project manager must be very careful in drawing conclusions. A small range indicates robust behavior of the system under study.

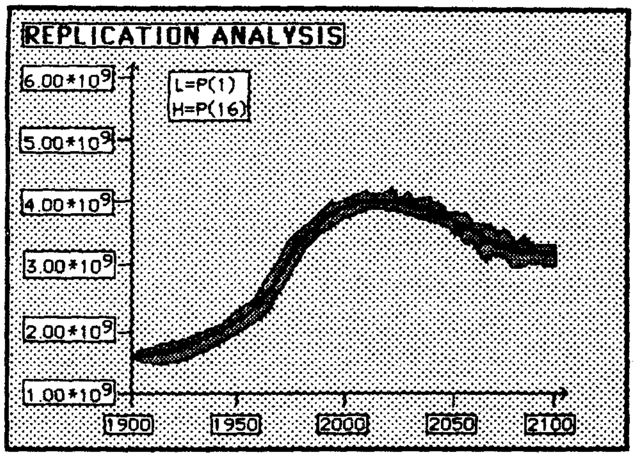
Replication Analysis

Replication Analysis describes the behavior of a noisy system over the possible ranges of noise generator parameters. Replication Analysis is sensitivity to noise.

Replication analysis is used to study the influence of noise generators on a noisy system. In general, it is difficult to judge from a single trajectory whether a particular observed behavior stems from the dynamics of the process or from the properties of the noise generator. In replication analysis, the simulation is repeated many times with different seed values for the random number generators. In this way, we again can obtain an envelope of system performance.

In DARE-INTERACTIVE, we can either specify **CALL REPLIC(100)** in the logic block, or **REPLICATE,100** as a run-time command to obtain 100 replication runs which again are stored automatically in 100 different SAVE records.

We applied this technique to Forrester's world model by making the former four «constants» now variables that are uniformally distributed over a range of *±25%* (DARE-INTERACTIVE contains a random number generator with several independent streams each of wich may take a different specifiable seed value. Random variables can be drawn from *11* different distribution functions.) We then selected a fixed step integration algorithm (as variable step integration performs poorly in case of noisy systems). Then, we performed a replication analysis over *16* runs, and displayed the resulting envelope together with the result from the *17th* run of the sensitivity analysis which we had previously STASHed away into the permanent data base for later reuse. The result is depicted in *Fig.3*:



Obviously, *16* runs were not sufficient to get a smooth envelope, but they were certainly sufficient to verify that no «catastrophic» behavior is going to take place if the parameters vary stochastically within the specified range. In fact, the model seems to be quite insensitive to this type of parameter variations. This may partly be due to the lowpass characteristics of the model which filters out all higher frequencies of the noise generators which represent a large percentage of the energy content of the noise signals. We may have to repeat the study by postulating a slower varying colored noise.

Range Surveillance

Range Surveillance allows to monitor the behavior of state variables and auxillary variables otherwise never looked at by specifying operating conditions on individual variables or sets of variables usually in form of inequality constraints.

In a large-scale model, there may be hundreds of variables out of which only a small percentage will ever be looked at by the user. It is quite frequent that the model is validated on the basis of individual variables which seem to take on reasonable values, and show a dynamic behavior for which a plausible explanation can be found. However, some other variables may pass unnoticed while exhibiting physically impossible behavior (e.g. a population growing negative). For that purpose, we should be able to specify operating conditions either by means of ranges for individual state variables and/or auxillary variables, or by means of inequality constraints involving several variables. Currently, there is no such feature built into DARE-INTERACTIVE yet.

Continous Steady-State Finding

The Continuous Steady-State is defined as the state in which all state variables have come to a rest, that is: All state derivatives take simultaneously a value close to zero.

When we discuss techniques for continuous steady-state finding, we mean techniques that execute faster than straight forward simulation until the steady-state is reached. Moreover, nonlinear models may exhibit several continuous steady-states. Which of them is approached, may depend on the chosen initial condition. Continuous steady-state finding tries to determine all possible steady-state points of a (nonlinear) model.

One way of solving this problem is to apply nonlinear programming techniques to minimize the weighted sum of the squares of all state derivatives. Such a technique has been built into one of the DARE dialects already, and is described in a separate paper in this conference. Another way (as currently applied in ACSL) is to look at the Jacobian of the model, and solve for the linearized equations.

We plan to integrate similar algorithms also into our version of DARE-INTERACTIVE.

Periodic Steady-State Finding

The Periodic Steady-State of a Model is defined as a limit cycle describing the behavior of the model over all future times.

Many nonlinear models do not approach a continuous steady-state. Instead, they enter into a limit cycle, that is: they show oscillatory behavior after the transient period has elapsed. For low frequency limit cycles, there is probably no better technique available than straight forward simulation. However, we face frequently limit cycles exhibiting fairly high frequency behavior. In such cases, the integration algorithm of ordinary simulation runs has to assume very small step sizes to follow the oscillations properly.

Periodic steady-state finding means to trade the phase information of the limit cycle for more efficient computation of its frequency and shape. Again, there have been proposed several different techniques to tackle this problem. One technique suggests to simulate the system over one «period» of the limit cycle, and use the average between initial value and final value for the next simulation over another period, until the final value converges to the initial value. This technique requires at least a very educated guess about the length of the period of the final limit cycle. Another technique tracks either maxima or minima of the simulation over some periods of the transient oscillation, and uses these points for extrapolating the solution over a longer time span, effectively superimposing a low frequency envelope integration over the high frequency oscillatory integration. This technique has been referred to as the **pseudostroboscopic integration** algorithm. It is pretty effective as long as only one pair of complex eigenvalues of the Jacobian lies close to the imaginary axis. Several such pairs in the vicinity of each other interfere in a way that is known to communication engineers as «fading». The pseudostroboscopic technique mostly fails in such cases. **Singular perturbations** could then still be used, but this technique is very hard to automate.

Currently, no techniques for periodic steady-state finding have been incorporated into DARE–INTERACTIVE, but we plan to implement those techniques described above.

Stochastic Steady–State Finding

The Stochastic Steady–State of a model is defined as the time after which the short term mean value of a stochastic variable does no longer change over time.

In all the simulation studies met so far, we were interested in analyzing the dynamic behavior of a model. However, there exist cases where we are actually interested in the steady-state of a stochastic model, and start gathering (statistical) information about the system only after the transient period has already elapsed. Such cases are called **non-terminating runs**. They are frequently discussed in the discrete event simulation literature, but were so far almost entirely ignored by the continuous system simulation community. However, the problem is of quite some interest, and several of the techniques described for the discrete event simulation case can easily be adapted to the continuous case as well. Currently, we have no results yet on this topic, but we are working on it, and plan to report about our findings at a later stage.

Curve Fitting

Curve Fitting allows to determine a set of unknown parameters on the basis of an optimal match between some postulated (e.g. measured) trajectories, and the outcome of a simulation.

Parameter identification is very closely related to simulation. It is rare that deductive modelling can be taken up to the determination of exact model parameter values. Thus, curve fitting is one of the foremost tasks of simulationists. Nevertheless does the conventional CSSL of today not offer any help in doing so. Most CSSL languages allow to initiate an optimization study, but the optimization algorithm is not provided, and traditional CSSL structures (e.g. in CSMP or ACSL) even prevent the user from applying a library function for that purpose. We believe that a nonlinear programming package should be an intrinsic part of any simulation system.

In DARE–INTERACTIVE, we are currently working on the following solution. We want to introduce e.g. the run-time command **FIT,ST-07** to fit some of the variables of our model to those variables specified in the 7th STASH record. Parameters to be fitted are declared in the parameter file as **UNKNOWN**, eventually together with a range in which the parameter values must be located (inequality constraints involving one parameter only). The format is the same as in the case of range analysis. Additional equality constraints and inequality constraints involving several parameters must be specified in a different way, e.g. by introducing the new keyword **CONSTRAINT** followed by a logical condition in the derivative block. The same syntax could then be applied to both the curve fitting and the range surveillance problem, where in the former case the constraints involve parameters only, while in the latter case they involve both variables and parameters. The algorithm to be used is specified by a separate keyword (similar to the integration algorithm). The nonlinear programming package has already been coded and tested. Currently, we are working on the interface to DARE–INTERACTIVE.

A separate program **GRAPHICS**, which has already been coded, transforms data stored on any ASCII file or series of ASCII files in free format to the format of a (DARE-internal) TIME record, then calls the graphics processor from where the data can be STASHed away. Yet another program for real-time data acquisition shall be added at a later stage.

Simulation Data Base

The Simulation Data Base serves to hold simulation data (and eventually also other time histories) for later reuse. The data may stem from different simulation runs, different simulation models, possibly even different simulation systems.

The STASH file is the DARE-INTERACTIVE data base which can hold any amount of data records stemming from various sources. The graphics processor can interact with the STASH file. In this way, results from different simulation models, eventually even coded in different languages, can be displayed together in one graph. In the same way, it is possible to graph simulated and measured data together in one plot.

DARE-P was the first simulation language on the market offering such a capability. Eventhough the current STASH file structure is most primitive, it already offers tremendous possibilities.

We are currently reimplementing the DARE graphics processor (a report on that project will be made available at a later stage). Among others, also the STASH file structure is currently being revised to further enhance the capabilities of the system.

We are also working on an interactive program for manipulation of technical data in a technical *relational data base*. The language shall be similar to **MATLAB**, Cleve Moler's famous «matrix manipulation laboratory». Relations in a relational data base are nothing but matrices. Just the way how particular elements are referenced, and the operations to be performed on those data are different from conventional matrix manipulation techniques. Recent research on *spreadsheet analysis* shall also be considered in the project. A report about this project shall be made later.

Conclusions

In this paper, we have summarized various means of enhancing the flexibility and versatility of simulation systems by improving their run-time experiment description capabilities. We have also introduced DARE-INTERACTIVE, a work bench for trying out various of these ideas in practice. Some of the tools are already operational such as the range analysis and the replication analysis. Other tools are currently under development. We project to have all the tools described in this paper ready in one year from now with the exception of the data base language which shall keep us busy for roughly two years. We hope that this presentation will stimulate more research in experiment description methodology.