TOWARDS A SPECIFICATION OF THE STRUCTURE FOR CONTINUOUS SYSTEM SIMULATION LANGUAGES

HUBER Rafael M., GUASCH Antonio

Instituto de Cibernética
Universidad Politécnica de Cataluña
Diagonal, 647
08028 BARCELONA

Present CSSLs are a valuable tool in systems engineering but some changes in the structure and a standardization of the language constructs would be useful.

A comparative review of the structure in recent simulation languages is carried out.

Afterwards, a language structure is proposed for the simulation of piecewise continuous systems whose aim is to provide a modular way to build-up the model as well as to ease the experimentation, validation and debugging.

Introduction

Recent years have seen intensive attempts to extend the art of mathematical modeling to an ever-expanding range of applications. This stress on modeling has been stimulated by the increased availibility of interactive digital computers and special-purpose simulation languages. A particularly significant impetus in the development of mathematical models, however, has been the tendency of virtually all the scientific fields to become more quantitative in their metodologies.

Nowadays the scientific community feels two main necessities

.Languages able to accurately represent continuous-discrete systems. There are two advises: Those who think that the right way is the combined system simulation languages (1) and those who advocate to define a new CSSL standard (2) for picewise continuous system simulation languages. From the confrontation of both points of view very interesting discussions have arisen (3).

.Languages supporting higher modularity in, at least, three aspects: the programming of the simulaton study itself, allowing program blocks consistent with the division of the physical system into subsystems, and sutomatic building of the model by calling in submodels.

Review of the Structure in Recent Languages

In 1981 an outline proposal for a new CSSL specification to replace the 1967 SCi CSSL specification was presented (2), (3),(4).

Four years later, in order to get an insight about the trends and results we have tried to extract the main structural features of several languages from the published information at our disposal.

The simulation software systems used for this purpose have been:

.COSY (5), SYSMOD (6) pertaining to the combined systems simulation languages.

.ISIS (7), (8), ESL (9) COSMOS (10) from the group of piecewise continuous systems simulation languages.

and the structural aspects considered:

. Simulation study organization.
. Coding of the system model.
. Coding of subsystems.
. Experiment description.
. Handling of disconinuities.
. Interactive features.

In this approach data types, details in the syntax, compiler versus interpreter, simulation environment and other features have been neglected.

Simulation study organization

All the aforesaid langauges use separate program units to describe the system and the experiment but the way in which they do it as well as the syntax change from one to another:

In some of them, the system description is explicitely called using a specific sequence while in others the call is implicit through a control statement or even without it.

. In some languages the model call is within the block describing the experiment while in others the system model is built from submodels in an ad-hoc block outside the experiment.

. The control statements which bracket the study are not the same and some languages use symbols.

Coding of the system model

All the studied languages,perhaps with one single exception, allow the users to build the models from submodels but they differ on structural aspects such as:

. Some of them divide the model description into two parts: a set of declaration blocks where data, procedures and processes may be declared and a specific block where the model itself is described while some others divide the model roughly into the classic regions and, if necessary, the submodel calls are done from the DYNAMIC region.

. The alloued regions in the model range from three (INITIAL, DYNAMIC, TERMINAL) to five by adding to the previous ones: declarations and a communication region which may contain output statements or procedural code concerning calculations which need not be performed in all the steps of the dynamic region.

. The execution of the INITIAL region which contains the procedural code describing the initial conditions setting, has been extended in one case to any time rather than just "time zero".

. The discrepancies concerning the DYNAMIC region definition are larger:

- In some languages it is a non procedural description of the differential and algebraic relationships which describes the model while in others the code has to be procedural because there is no requirement for automatic sorting of its statements.
- The concept of DERIVATIVE section within the dynamic region to separate code which is executed only during the integration from that which has to be executed only at each communication interval is advocated in the CSSL81 proposal but none of the above mentioned languages retains it.

- Isolated storage and translation of models is not possible in some of the analysed languages, therfore, the creation of preprocessed libraries of models and submodels as recommended in the CSSL8i proposal cannot be envisaged.
- The TERMINAL region seems to be always devoted to the procedural description of the end-of-run calculations and not to perform functions concerning the simulation control flow. In some cases,however. output statements are allowed.

  Other langages use it just to code the invalidity conditions of the model; the post-run calculations beeing lumped in a specific block inside the experiment.

Coding of subsystems

Different blocks have been proposed and a first remark is that the bracket structure statements are misleading because some of them may have different meanings according to the different languages

The main differences we have noticed are:

. If a model is defined as the description,using representation statements (parallel code), of a part of the system as an autonomous submodel which communicates with its environement through a list of parameters, among others the following different structure statements are used:

MODEL with invocation name in the definition and SUBMODEL with the same name in the calling sequence.

SUBMODEL with invocation name in the definition and just the name in the calling sequence.

The possibility of models to be preprocessed and called, seems not to be a general rule.

. There is no general agreemant about the inclusion in the language of the MACRO, PROCED and NOSORT pseudoblocks.

. In some languages the MODULE block has been defined as modular program unit and looking for a solution recognasing that separate translation of MODELs can only be achieved for a particular class of description.

It can contain the same regions as the model and these are automatically transferred by the MODULE handler to their correct locations whenever the MODULE is called. It must be stored in source form. In its definition, formal parameters need no longer be separated into input and outputs as is the case in the MODEl or SUBMODEL block; upon usage the mapping of the actual to formal parameters is specified by explicit assignments; besides replacement of parameters it uses formulae manipulation and there is no requirement for the user to resolve the issue of the unique naming of variables.

Two of the considered languages have MODULE blocks defined but, from the information we have, we cannot ensure that both are exactly the same or have all the above features.

. SEGMENT, as opposed to MODULE, is defined in the CSSL81 proposal as a block describing a parallel process which constitutes one part of a partitioned system model, which communicates data with other segments only at a regular communication points.

None of the languages have implemented that block.

Experiment Description

The experiment description ranges from poor structures to very nice features.

. The simplest one found is a procedural control section in which it becomes possible to embed the model within a main program.

. In the mid-range, two blocks are defined: a MONITOR describing the control signals (input signals) to the model and OUTPUT describing the output variables and modes.

. At the top level of the offered features, the experiment description consists of a declaration part and one or more experimental frames.

In the declaration part the initial state of the model is given, parameters may be initialized, the integration methods and output requeriments are described.

Simulation FRAMES in which several simulation runs may be specified, can be defined as well as GLOBAL FRAME, PREFRAME and POSTFRAME.

A RUN block enables the user to initialize each run and a POSTRUN assembles the statements to be executed when a finish condition is met.

Moreover a CONTINUOUS MONITOR can be set to observe the model continuously and even to integrate a set of first order differentialequations. It is to be noticed that MONITOR had in the preceding paragraph an absolutely different semantics and it is again a misleading control statement

Handling of discontinuities

Nearly all the possibilities have been found:

. Precoded discontinuous functions offered in a library by the language. These are generally the well-known simulation operators of the CSSL67 but generating numerically well-conditioned run-time code.

. Coding subroutines,MODELs or MOUDLEs

. Using IF clauses with different syntax.

. Using WHEN statements.

. Other statements.

Interactive features

Little is said about interactivity but in those languages with the most advanced features, it is restricted to two phases:

. Firstly, the interruption of the program execution so as to pass control to the user, which can be achieved by inserting an INTERACT control statement or pressing a key in the keyboard.

. Secondly, the use of interactive commands to monitor or change variables, to change the outputs or to post-mortem graph plotting.

A system structure

At present, the group working on systems simulation is carrying out the feasability study of a structure and related language constructs converging with the trends on picewise-continuous system simulation languages (11), (2),(5).

The formal design is carried out using syntax analysis tools (12), (13) and some of the objectives to be attained are:

a) The language should manage a modularity consistent whith the division of the physical system into subsystems through a minimal but sufficient number of different blocks.

b) The separation between the model description and the experiment should be done in such a way that the model remains unchanged along the experiments and ready to be used from another model (submodel) at the end of the validation and verification experiences.

c) The language should be designed in order to be easily extendable to real time applications.

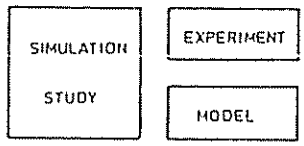
d) It should be possible to build a system model in a bottom-up way relating two or more submodels from a model or submodel of higher hierarchical level.

e) Semiphysical and hardware in-the-loop simulation studies should be available.

f) Isolated preprocessing of the models as well as run time symbolic acces to the variables should be able.

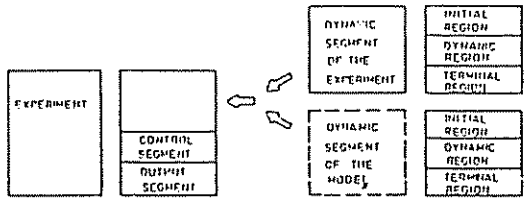
Simulation study

It consists of two distinct blocks EXPERIMENT and MODEL whose structure guarantees the proposed objective.



Besides, sets of experiments and sets of models can be created in order to be used in later studies.

Experiment



In the most general case an experiment may have three blocks:

. A DYNAMIC SEGMENT with the classic three regions: INITIAL, DYNAMIC and TERMINAL.

There is no difference with the dynamic segments defined in the submodels. Its aim is to allow the inclusion in the experiment of the dynamics external to the represented system but in most cases necessary for its evolution or continuous cost functions computation. In simple studies the segment can be the representation of the system but that is not at all its design objective.

In most frequent situations,one or more models will be called from the dynamic, the latter option opening the possibility of simultaneously performing the same experiment on different independent models.

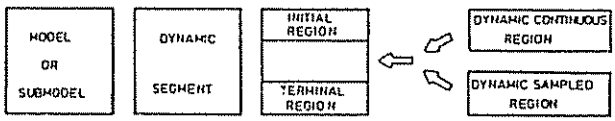
. A CONTROL SEGMENT describing the operations to be performed between runs by means of procedural statements.

In this segment it is possible to specify experiments such as parameter sweeping, optimization or boundary condition problems whose codes are independent with respect to the dynamics and output.

. An OUTPUT SEGMENT clustering the output control statements. Its objective is to make explicit the variables whose evolution has to be recorded.

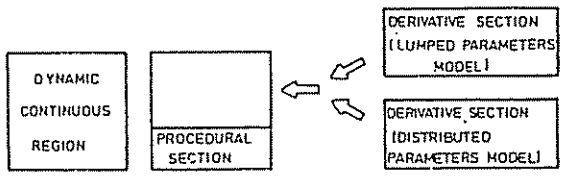
Models and submodels

They can be represented using the above described DYNAMIC segments.



The nature of the equations coded in the DYNAMIC region charaterizes the nature of the submodel as continuous (lumped or distributed parameters) or sampled.

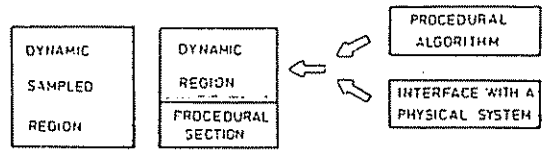
The invocation of the contributory submodels is done from the dynamic region.



. In the CONTINUOUS DYNAMIC REGION two sections are possible: a DERIVATIVE section containing representation statements (parallel code) and pseudoblocks and a PROCEDURAL SECTION.

If the syntax of some representation statement corresponds to a partial differential equation, the whole model block will be considered as representing a continuous distributed parameter system.

The procedural section is intended to include the code which need not be executed in every integration step but only at each communication interval.



. The SAMPLED DYNAMIC REGION is characterized by the fact that it communicates with the other dynamic regions only at fixed intervals.

Its dynamic section can be of two kinds:

- A procedural algorithm
- An interface with a hardware subsystem. This kind of dynamic section differs from all the others in two main aspects: it depends on the hardware configuration of the host computer and it requires real time computation (14).

Postprocessor.

Although not involved in the language structure but in the system it can be mentioned that a postprocessor is envisaged in order to individually or cross analyze the results obtained in different studies.

Conclusions

CSSL81 proposals do not in themselves constitute a complete specification and, evidently, the scenario four years later shows a misleading family of continuous system simulation language structures as shown in the review, the proposal in this paper beeing one more member thereof.

The authors feeling is that, if simulation of continuous systems is to become a popular tool, it is the right moment for the scientific community to make an effort and define a complete state-of-the-art standard for CSSLs which may ensure the user program portability among its different implementations.

Bibliografia

(1) Cellier F.E. "Combined continuous/discrete system simulation languages... usefulness, experiences and future development" in Methodlogy in systems modelling and simulation pp. 201-220, ed. Zeigler et al. North Holland 1979

(2) Hay J.L. "A new CSSL standard -an implementation view" pp. 35-43 in Proc. of the UKSC Conference on computer simulation, Harrogate. May 1981

(3) Crosbie R.E., and Cellier F.E. "Progress in simulation language standards = An activity report for Technical Committee TC3 on Simulation Software 1979-82" Vol. I pp. 411-412 in Proc. 10th IMACS World Congress on Syst. Simul. and Scient. Comp., Montreal, Aug. 1982.

(4) Hay J.L. and Crosbie R.E. "Outline proposal for a New Standard for Continuous System Simulation Language (CSSL 81)", Computer Simulation Centre, Department of Electrical Engineering, University of Salford, Salford, March 1981.

(5) Cellier F.E. and L.P. Bongulelmi "The COSY simulation language" pp. 271-281 in Simulation of systems: proc. of the 9th IMACS Conf. on Simulation of Systems, ED. North Holland pub. co., Sorrento, Italy. 1979.

(6) Baker, N.J. and Smart, P.J. "The SYSMOD simulation language" in The European Simulation Congress ESC 83 pp. 281-286, Aachen, (Sept. 1983).

(7) Hay, J.L. "Iteractive simulation on minicomputers: Part 1 - ISIS, a CSSL simulation language" in Simulation, pp. 1-7 (July 1978).

(8) Pearce J.G. "Iteractive simulation on minicomputers: Part 2- Implementation of the ISIS language" in Simulation pp. 43-52 (August 1978).

(9) Hay, J.L., Crosbie R.E. and Pearce, J G. "ESL: A Simulation Language for the Sapce Industry" in ESA Journal Vol. 8 (1984).

(10) Kettenis,D.L. "The COSMOS modelling and simulation language" pp. 249-260 in First european simulation congress ESC 83, Aachen, Sept. 1983.

(11) Crosbie R.E. and Hay J.L. "Towards new standards for continuous-system simulation languages" in Proceedings of the 1982 summer Computer Simulation Conference pp. 186-190, Denver, July 1982.

(12) Lesk M.E. "Lex-A lexical Analyzer Generator" in Computing Science Technical Report N. 39, Bell Laboratories, Murray Hill, N.J., March 1975.

(13) Johnson S.C. "Yacc: Yet Another Compiler Cpmpiler" in Computing Science Technical Reprot N. 32, Bell Laboratories, Murray Hill, NJ, 1975.

(14) Crosbie, R.E. "Interactive and Real-Time Simulation" pp. 393-406 in Progress in Modelling and Simulation ed. Cellier F.E., Academic Press 1982.