THE GASP-VI SIMULATION PACKAGE FOR PROCESS-ORIENTED COMBINED CONTINUOUS AND DISCRETE SYSTEM SIMULATION

by: Magnus Rimvall & Francois E. Cellier
Institute for Automatic Control
The Swiss Federal Institute of Technology Zurich
ETH - Zentrum
CH-8092 Zurich
Switzerland

GASP-V, the predecessor of GASP-VI, was presented for the first time during the 8th AICA (IMACS) Congress held at Delft in 1976. This software was finally completed in spring 1978 and since then it has been very successfully applied by many (both industrial and university) research groups. GASP-V provides for mechanisms for: (i) continuous simulation (in the state-space domain), (ii) discrete simulation (event oriented), and (iii) distributed simulation (method-of-lines) and any mixture of the three (combined simulation). In the mean time, it was decided that this (FORTRAN-based) software package should be enhanced by adding to it a process interaction mechanism for both continuous and discrete processes. This enhancement is particularly essential in the context of the new simulation language COSY for which GASP-VI is the run-time system. However, even independently of COSY, a process interaction mechanism is very useful as it supports a much more modular programming, and, by these means, helps the user in formulating his models and in coding them error free.

The here presented new system GASP-VI is upwards compatible with GASP-V and provides for the above mentioned additional process interaction mechanism. This paper concentrates on the innovative aspects of this new development as there are: (i) a new mechanism for list processing with variable-length list entries, and (ii) a new resource allocation mechanism which allows for a significantly improved monitoring of resource allocations, and which moreover allows to add individual attributes to resources in a resource pool.

1. INTRODUCTION

GASP-VI is a new (upwards compatible) extension to the well known and widely distributed simulation packages GASP-IV [8] and GASP-V [2,3].

While GASP-IV was already able to perform combined simulation, this software system was still fairly simple, using an event view on the discrete side, and offering only few features on the continuous side (e.g. only one Runge-Kutta integration algorithm which made this software system unusable for stiff system simulation).

GASP-V extended this software system by adding to it:

1) a library of integration algorithms including among others the Kahaner implementation of the GEAR algorithm for stiff system integration,

2) facilities for distributed system simulation (modeled by partial differential equations) by using the method-of-lines approach,

3) improved capabilities for state event detection by employing inverse Hermite' interpolation,

4) a library of discontinuous functions (comparable to the one offered by CSMP), including among others hysteresis, step, and pulse functions, but resolving all discontinuities by an internal event handling mechanism and, by these means, getting around the numerical problems (creeping effects) which are well known to arise when these functions are used in a CSMP (or similar) program,

5) a data base mechanism to store simulation data away during the run together with a postprocessor to retrieve the data from the data base and to display them in a variety of attractive formats, and

6) improved facilities for optimization studies.

As one can see, all these improvements concern the continuous side while the discrete side was left unchanged in GASP-V.

GASP-VI, finally, adds a proceess interaction view to GASP both for discrete and for continuous processes. GASP-VI programs resemble to a large extent GASPPI programs [10], another software development from Pritsker & Assoc. which was, however, never brought to an end. Still, the GASP-VI software is entirely recoded and uses internally quite different mechanisms from GASPPI.

One of the major reasons for the development of this new extension GASP-VI was to obtain a reasonably adequate target system for the COSY simulation language which was described in [2,4,6]. Originally, COSY was intended to be an easy to use front end for GASP-V to make the coding of GASP programs somewhat easier and more error safe. During the design of the new language, we then realized that COSY could be much more than what our original intentions were. We now view COSY as the Swiss contribution to a replacement for the (meanwhile somewhat outdated) CSSL specifications [11]. With this modified design goal in mind, we noticed that, sticking too closely to GASP-V, would be more of a hindrance than of a help. So, COSY contains now several new features which are not easily expressable in terms of GASP-V, while some of the previously available GASP-V features are no longer supported in COSY. When the design of COSY came close to its finalization, we realized that GASP-V no longer was an acceptable candidate for use as a target software for the COSY preprocessor. This led to the development of GASP-VI which, on the one hand, supports all features offered by COSY reasonably well but, on the other hand, is still up-

wards compatible with GASP-V to make this software attractive also to former (and future) GASP programmers. One problem with this approach is the difficulty that error reporting should, in all stages, be done with respect to the source code, that is, differently for COSY programmers (who need to obtain references to their original COSY programs) as compared to GASP-VI programmers who need references to their GASP programs. For this reason, the COSY preprocessor must generate, in addition to a GASP-VI program, a (machine readable) crossreference table of COSY and GASP variables and line numbers together with a switch in the GASP data "cards" which tells the GASP executive software whether the code was generated by the COSY compiler or directly user written. This flag, however, influences only the error reporting as well as the monitoring and tracing activities, thus creating some memory (space) overhead but no execution (time) overhead.

GASP-VI, the youngest "child" in the GASP program family, is to be briefly presented in this article.

2) DATA STRUCTURES IN GASP-VI

Beside of the standard FORTRAN data types (scalars and arrays), GASP-IV (and with it also GASP-V) offers one additional standard data type which is a linearly forward and backward linked list (called "file" or "queue" in GASP-IV). The elements of this list structure are records (in GASP-IV called "entities" or "entries"). Each list has associated with it one (out of four available) ranking rule (e.g. FIFO, or HVF on ATRIB(5), etc.). The manipulation of records in lists is done through standard subroutines (e.g. FILEM to insert a record into a list, or RMOVE to remove a record from a list).

This data structure is used for two distinct purposes:

1) the modeling of waiting queues which are elements common at least to all transportation problems, and

2) the maintenance of the calendar of events. This latter usage is somewhat special, in that:

   a) Each event record contains two standard attributes
      - event time (time when the event is to take place): ATRIB(1), and
      - event code (type of event to take place): ATRIB(2).

   b) Event removal from the calendar of events is performed automatically at event time.

   c) There exists precisely one calendar of events: (file no 1).

   d) The ranking rule for this list is always LVF on ATRIB(1) (in GASP-V: HVF on ATRIB(1) in case of backward integration) with a possibility to specify a secondary ranking for simultaneous events.

During the implementation of SLAM-II [9], another direct descendant of GASP-IV, the inaugurators (Pritsker & Assoc.) obviously realized that the above outlined approach was not ideal as:

1) the GASP-IV user had always to remember that, in the event list, the first two attributes are system reserved -- and so they modified the scheme in that SLAM-II maintains the standard attributes as the last rather than as the first attributes in the record, and

2) the GASP-IV user had always to remember that file no 1 is reserved for the calendar of events, that is that the freely available lists for waiting queues start from index no 2 -- and so they modified the scheme to preserve the calendar of events as the last rather than as the first file number.

However, although this modification clearly indicates an improvement, it does not solve all problems, as, in process interaction, additional "special purpose" list types should be introduced, e.g. to store the attributes of transactions. Due to the above mentioned limitations, SLAM-II does not store these attributes in the user accessible list structure with the implication that attributes of "other" transactions are not easily user accessible, and that there exists only a limited possibility to modify transaction attributes from within the event portion of the program (except for the case when a transaction causes the event by passing through an EVENT-node).

It was one of the design goals of GASP-VI that both transaction records and resource records (the latter shall be described in more detail in the next section of this paper) should be transparently stored by use of the standard list processing mechanism. For this purpose, however, the list processing mechanism of GASP-IV had to be extended and, in effect, to be largely recoded.

Let us look more closely at a transaction record. For such a record, GASP-VI maintains 11 transaction specific standard attributes (called "hidden attributes" in GASP-VI):

1) pointer to time event record
2) pointer to resource request records
3) pointer to records of allocated resources
4) pointer of timeout event record
5) current status of the transaction
6) process number
7) block number
8) generation time of the transaction
9) mark time of the transaction
10) priority
11) new block number (in case of a GGOTO statement being performed)

From this list, it should become clear that:

a) Neither is it acceptable that the user starts counting his own (user) attributes from index 10 onwards (and again differently for continuous process records, resource records, etc.), nor is it acceptable that the user is forced to access e.g. the mark time by counting his own attributes plus an offset of 5. We obviously need a separate access mechanism for hidden attributes. In GASP-VI, the user must count only his own (user) attributes while all hidden (system) attributes are automatically copied from and to special common blocks (except for the before mentioned standard attributes of the calendar of events for reasons of upwards compatibility).

b) We may easily face situations in which one list (e.g. containing transaction records) requires 50 attributes while another (e.g. for a waiting queue) requires only one or two. In the case of the event queue, it shall even be the normal situation that one record contains many attributes while another contains only very few. That is: neither the GASP-IV rule that all records in all lists must be of equal size, nor the other GASP-IV rule stating that the maximum allowed size is 25 are acceptable. To circumvent this problem, we now distinguish between:

- logical records which may be of any size (no limitation) and which may be of unequal size, and

- physical records (also called "slices") which are of equal size but without restriction as concerning their tolerated size.

Each logical record contains now (at least) three pointers, two of them to denote the logical predecessor and successor (as in GASP-IV), and a new one to point to the next slice belonging to the logical record. Three additional hidden attributes pertain to all records:

1) number of (hidden and user) attributes of the record (statistics are automatically recorded on the average number of attributes -- useful for tuning), and

2) record type (used for proper monitoring and also to prevent illegal use)

3) time the record was inserted in it's present file, an information which is used for two purposes:

   a) statistics are automatically recorded on the average time a record spent in a queue beside of the (in GASP-IV available) average number of records in the queue -- a quantity which we noticed to be frequently very useful, and

   b) to allow for an automated timeout mechanism for the detection of deadlock situations.

The physical size of a slice (NNATR) must be user determined. Whenever one or several processes take part in the model, the minimum size is automatically put to 10. It is a good rule to dimension the slices such that a little more than 50 percent of the logical records occupy one slice only.

3) RESOURCES

Let us for once model a car rental company. Tourists (transactions) arrive to get hold of a car (resource). As long as there are cars available, the tourist shall receive his car, otherwise he has to either wait until a car comes back or do something else.

This situation is modeled in most process-oriented simulation programs (e.g. in SLAM-II, GPSS-V, or GPSS-FORTRAN) by means of an integer counter which is decremented whenever a car leaves the company and incremented whenever a car is returned. Usually, a second integer denotes capacity.

By use of this mechanism, any tourist may easily deposit his rented car at the next corner and fly home (that is: the transaction leaves the system) without being punished for doing so. Even worse, as long as always at least one car is out, a tourist who decided to travel around by using the public transport system, may readily return a car (which he never possessed) before going home! (An error message shall be produced by some of the available software systems when the number of available resources increases beyond capacity.)

Most software producers are obviously happy when their product is able to handle all "correct" programs correctly. However, we feel that it should be one of the most noble duties of any software producer to support the application programmer against his own stupidity!

In GASP-VI, we therefore went along a different line. Each resource is represented by a record. All free resources are linearly linked in a resource list. Whenever a resource is allocated, this resource record is taken out from the list of free resources and linked into another list which in itself is linked to the transaction record. That is: all resources (of possibly different types) which a transaction currently holds allocated are linked into a linear list which in itself is forward and backward linked to the transaction record. If a tourist forgets to return his car, hotel key, etc. when flying home, a warning message is printed out, and automatically all allocated resources are taken from him. If one tourist rents a car which is then "stolen" by another tourist over night (a situation which is, in fact, quite difficult to program), this other tourist must be an extremely clever GASP-VI programmer if he should be able to return the stolen car to the car rental company without being caught.

One side effect of this methodology is that automatically resource utilization statistics can be obtained even within resource pools for each resource individually.

Another side effect is that each resource in a resource pool may carry its own individual set of attributes. That is: the car company may dispose of large (expensive) and of smaller (less expensive) cars (classes A to D), and each tourist may get a car out of his desired class only. The GASP-VI programmer may still decide to keep all cars in one single resource pool (e.g. to get global statistics on all cars).

As any other solution, also this approach has its inherent weaknesses. Let us model a computer system for which the central memory is to be represented by a resource pool. A new job (transaction) arrives and wants to occupy a portion of the central memory. Obviously, it is not a splendid idea to model 4 MBytes of available memory by 4 millions of resource records out of which 128'000 are allocated (that is: relinked) to the new job! By our approach, it is important to keep the number of resources in a pool within reasonable limits, e.g. by using appropriate measurement units. In the here described situation, we may for instance foresee that memory is assigned in multiples of 32k only which are then represented by one single resource.

4) STATUS OF IMPLEMENTATION

The current status of implementation is such that:

a) The COSY language design has been completed (except for a few minor details) by use of a general purpose parser program described in [1].

b) Many quite large application problems have been coded in COSY and have been tested by use of the parser program to verify the ability of COSY to conveniently deal with large-scale problems.

c) The new data structures for GASP-VI as well as the new monitoring and debugging facilities are operational.

d) A basic set of process interaction routines has been coded and is currently in the test phase. Additional process interaction routines have been designed but not yet implemented.

e) It is planned to complete the development of GASP-VI (including a new manual for both GASP-V and GASP-VI features) still in 1982, while the COSY implementation must wait until GASP-VI is fully operational.

5) COMPARISON OF GASP-VI WITH SLAM-II AND WITH COSY

Was the development of the new software system GASP-VI really justified? An alternative approach could have been to merge the enhanced features of GASP-V (enhanced on the continuous side) with those of its "brother" software SLAM-II (enhanced on the discrete side), a task which would have been fairly easy to accomplish. SLAM-II, an application of which is described in a companion paper [7], enhanced the discrete modeling capabilities of GASP-IV by adding a network description mechanism to it. For simple applications (like the single-server-single-queue problem: Joe's Barbershop), it is fairly easy to show that this approach is equivalent and even almost identical to a process interaction mechanism. Unfortunately, this equivalence holds only for very simple processes which basically consist of sequential calls to process interaction routines (like: generation of a new transaction, allocation of resources, time-advance, etc.). However, more complex processes may involve a large amount of sequential logic. For such cases (like the elevator problem for which a COSY solution was presented in [5]), a network representation would have to consist almost exclusively of EVENT-nodes which makes this approach neither readable nor writeable, neither attractive nor efficient. That is: for simple processes, we would consider SLAM-II and COSY approximately equivalent, while SLAM-II is without question by far superior to GASP-VI both as concerning the length of the user code and as concerning the probability for writing error-free programs. On the contrary, as soon as a simulation model contains extensive logic, GASP-VI has some distinct advantages over SLAM-II, while certainly COSY must be considered the most attractive of the three.

REFERENCES

[1] Bongulielmi A. P. and F. E. Cellier: (1979) "On the Usefulness of Deterministic Grammars for Simulation Languages". Proc. of the Sorrento Workshop on International Standardization of Simulation Languages (SWISSL), Sorrento, Italy, Sept. 19 + 20, 1979.

[2] Cellier F. E.: (1979) "Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools". PhD Thesis, Diss ETH No 6483, The Swiss Federal Institute of Technology Zurich.

[3] Cellier F. E. and A. E. Blitz: (1976) "GASP-V: A Universal Simulation Package". Proc. of the 8th AICA Congress on Simulation of Systems, Delft, Netherlands, August 23 - 28, 1976. Published by North-Holland Publishing Company (L. Dekker, ed.); pp. 391 - 402.

[4] Cellier F. E. and A. P. Bongulielmi: (1979) "The COSY Simulation Language". Proc. of the 9th IMACS Congress on Simulation of Systems, Sorrento, Italy, Sept. 23 - 27, 1979. Published by North-Holland Publishing Company (L. Dekker, G. Savastano and G. C. Vansteenkiste, eds.); pp. 271 - 281.

[5] Cellier F. E., A. P. Bongulielmi and M. Rimvall: (1981) "Comments of the Swiss TC3 Group on the 'Outline Proposal for a New Standard for Continuous-System Simulation Languages (CSSL 81)'". In: TC3 of IMACS, Committee on Simulation Software, Committee Newsletter, no 10, Sept. 1981, Appendix 3 (R. E. Crosbie and F. E. Cellier, eds.).

[6] Cellier F. E., M. Rimvall and A. P. Bongulielmi: (1981) "Discrete Processes in COSY". Proc. of the European Simulation Meeting on Simulation Methodology held at Cosenza, Italy, April 9 - 11, 1981. (F. Maceri, ed.).

[7] Graber A. and F. E. Cellier: (1982) "On the Use of SLAM-II for Modeling and Simulation of Large Transport Problems". (this volume)

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

[9] Pritsker A. A. B. and C. D. Pegden: (1979) "Introduction to Simulation and SLAM". Halsted Press (John Wiley) and Systems Publishing Corporation.

[10] W. B. Washam and A. A. B. Pritsker: (1976) "Introduction to GASPPI". Unpublished Document, Pritsker & Assoc., Inc.

[11] (1967) "The SCi Continuous System Simulation Language (CSSL)". Simulation, vol 9, no 6, Dec. 1967; pp. 281 - 303.