GASP IV: A COMBINED CONTINUOUS/DISCRETE, FORTRAN BASED SIMULATION LANGUAGE

by

Nicholas R. Hurst

A. Alan B. Pritsker

Center for Large-Scale Systems

Purdue University

Abstract

GASP IV is an extension of the next event simulation language GASP II. A generalization of the definition of "event" and additions to the language structure enable GASP IV to be used for continuous or combined models while retaining the full power of GASP II for discrete models.

Continuous system state description may be in the form of a set of algebraic and/or differential equations. GASP IV handles the details of state and event control (including state variable integration when necessary), information storage and retrieval, system performance data collection and analysis, and report and plot generation.

In addition to the models which can be coded in GASP II, the following types of models have been successfully coded in GASP IV: Systems Dynamics Models (Industrial, Urban, and World Dynamics Models); Mechanical Impact Models; and Chemical Process Models.

In each case, an analyst familiar with GASP II has been able to quickly write the GASP IV code.

Introduction

GASP IV is a new simulation language with new capabilities. Although it is an extension of GASP II, it provides many of the capabilities normally associated with continuous simulation languages. These additional capabilities are integrated into the GASP II structure resulting in a conceptually and physically integrated language. Because GASP II is well documented (References 3 and 4), this paper will emphasize those features and capabilities of GASP IV not included in GASP II.

GASP IV consists of a set of FORTRAN subroutines organized to assist the analyst in preparing discrete, contiuous, or combined simulation models. GASP IV formalizes an approach to the preparation of such models by providing an appropriate world-view supported by prepared subroutines which handle the problem-independent structure of the model. The world-view provided describes the status of the subject system in terms of a set of state variables and a set of entities with their associated attributes. The GASP IV simulation philosophy is that the dynamic simulation of a system can be obtained by modeling the events of the system and advancing time from one event to the next. This philosophy presumes an expanded definition of "event" which will be stated later.

Every GASP IV simulation model consists of:

1) A set of subroutines which describe a system's operating rules. (Subroutines defining events, conditions causing events, and the trajectories of the state variables.)
2) Lists and matrices which store information.
3) An executive routine.

The set of subroutines describing the operating rules represent the technological logic of the system being studied. The lists and matrices represent the specific entities, their attributes, and associated control information. Variables common to many simulation programs are defined and provided as GASP variables requiring the user to define only problem dependent, non-GASP, variables.

The executive routine and its supporting subroutines provide the nine functions shown below:

1) State and event control including state variable integration when necessary.

   SUBROUTINE GASP

2) System initialization.

   SUBROUTINE DATIN

   SUBROUTINE CLEAR

SUBROUTINE SET

3) Information storage and retrieval.

   SUBROUTINE FILEM (IFILE)

   SUBROUTINE RMOVE (NTRY, IFILE)

   SUBROUTINE CANCL (NTRY)

   SUBROUTINE COPY (NTRY)

4) Location of specified state conditions.

   FUNCTION KROSS (IKRSG, IDRSD, CONST, LDIR, TOL)

   SUBROUTINE FIND (XVAL, MCODE, IFILE, JATT, NTRY, TOL)

5) System performance data collection.

   SUBROUTINE COLCT (XX, ICLCT)

   SUBROUTINE TMST (XX,T,ISTAT)

   SUBROUTINE HISTO (XX, A,W,IHIST)

   SUBROUTINE GPLOT (IPLOT,ITAPE, NVARP, LCODE, TIME, P)

6) Statistical computation and reporting.

   SUBROUTINE PRNTQ (IFILE)

   SUBROUTINE PRNTS

   SUBROUTINE SUMRY

7) Monitoring and error reporting.

   SUBROUTINE MONTR

   SUBROUTINE ERROR (KODE)

8) Random variate generation.

   FUNCTION DRAND (ISTRM)

   FUNCTION UNFRM (ULO,UHI,ISTRM)

   FUNCTION RNORM (IPAR,ISTRM)

   FUNCTION RLOGN (IPAR, ISTRM)

   FUNCTION ERLNG (IPAR,ISTRM)

   FUNCTION GAMA (IPAR,ISTRM)

   FUNCTION BETA (IPAR,ISTRM)

   FUNCTION NPOSN (IPAR,ISTRM)

   FUNCTION GAM (AK,ISTRM)

9) Miscellaneous support.

   FUNCTION SUMQ (JATT,IFILE)

   FUNCTION PRODQ (JATT, IFILE)

   FUNCTION GTABL (TAB,X,XLOW, XHIGH, XINCR)

   SUBROUTINE GDLAY (IFS,ILS,XIN,DEL)

Because of the functions performed by GASP IV, the analyst need only prepare subroutines defining the events and state variables in order to obtain a complete simulation model.

Event Definition

The GASP IV definition of "event" is fundamental to the world-view which supports the modeling of continuous, discrete, or combined systems within the same conceptual framework.

AN EVENT IS ANY POINT IN TIME BEYOND WHICH THE STATE OF A SYSTEM MAY NOT BE PROJECTED WITH CERTAINTY.

It should be noted that this definition does not relate an event to any change, either discrete or continuous, in the state of a system. Such a relationship often exists, but it is possible to have an event with no associated change in system state. Conversely, it is possible to have a change in system state with no associated event.

In GASP IV, it is useful to describe events in terms of the mechanism by which they are scheduled. Those events which occur at a specified time are referred to as time-events. They are the type of event commonly thought of in conjunction with "next-event" simulation. Events which occur when prescribed conditions defined in terms of the system state are met are called state-events. Unlike time-events, they are not scheduled in the future. They may, however, initiate time-events. Likewise, time-events may initiate state-events. The example presented in this paper illustrates these types of interaction.

The GASP IV Language

The execution of a typical GASP IV program begins with a user provided main program which initiates the simulation. Control is then transferred to GASP, the executive routine, which controls the simulation until completion. A general flow chart of SUBROUTINE GASP is shown in Figure 1.

GASP first calls SUBROUTINE DATIN which initializes all GASP variables either directly or from reading data cards. In addition to initialization, DATIN also provides an echo check of the input data.

Immediately following initialization, GASP prepares to advance simulated time. GASP uses a combined "next-event" and "step-evaluation-step" method of time advance. This combined method is necessary because of the potential existence of state-events whose location on the time axis are not known. GASP first checks to see if there is a time-event to process. If there is, that event is processed by calling SUBROUTINE EVNTS(IX) with the proper event code. If not, GASP checks to see if there are any active state or derivative equations. If there are none, time is advanced from time-event to time-event as each is processed. That is, it proceeds as in GASP II. If there are active state or derivative equations, a different time event mechanism is used. Time is advanced by the maximum allowable step size (user specified) or to the next time-event, whichever is less. (If there are active derivative equations, this involves intermediate steps and accuracy checks.) At that point the system state is examined to see if a state-event has occurred. If a state-event has been passed by more than the specified tolerance, time and state are reset to the beginning of the step and a smaller step size is tried. If no state-events have been passed by more than the specified tolerance all state-events which have occurred within the specified tolerance are processed. If a time-event is scheduled, it is processed. If no event is scheduled, another step is started.

Upon satisfaction of user specified conditions, the run is terminated. GASP then calls SUBROUTINE SUMRY to provide a summary report, calls SUBROUTINE OTPUT to provide user defined output, checks the number of runs remaining, and then either begins a new run or returns to the main program.

Description of Example Problem[1]

As an example of the use of GASP IV, consider the system depicted in Figure 2. A hydrogeneration reaction is conducted in four reactors, each of which may be started, stopped,

1 The authors are indebted to Professor J. M. Woods of the Purdue University School of Chemical Engineering who formulated this example problem.

discharged, or cleaned independently of the others. A compressor with constant molal flow rate provides a supply of hydrogen gas to the reactors. The hydrogen flow is as shown in Figure 2.

The operating policy for the facility is to start each of the reactors initially at 30 minute intervals. The concentration of the reactants is then monitored until it reaches 10% of its initial value at which time the reaction is complete and the reactor is turned off. Following completion of a batch, the reactor is discharged, cleaned, recharged and restarted. The time to discharge the reactor is known to be exponentially distributed with a mean of one hour. The time to clean and recharge the reactor is known to be approximately normally distributed with a mean of one hour, a standard deviation of one-half hour, a minimum of zero hours, and a maximum of two hours.

The valve connecting each reactor to the rest of the system is adjusted by controls so as to maintain an effective pressure of 100 psia in each active reactor unless the system pressure has fallen below 100 psia in which case the effective pressure is the actual system pressure.

In order to preclude the pressure from falling too low; if system pressure falls below the critical value (100 psia), the last reactor to have started is immediately shut off. In addition, no reactors are ever started if the system pressure is below the nominal value of 150 psia.

Only two events are associated with the system; a start of reaction event and a stop reaction event. The start of reaction event may be either a time-event, based upon a specified time from completion of one batch to the start of the next batch; or a state-event, based upon system pressure rising above nominal. The stop reaction event will be treated as a state-event, based either upon completion of a batch or pressure falling below critical. These events will be described more fully in the discussion of the coding which follows.

Coding for Example Problem

A liberally annotated listing of the source program for the cited example is given in Figure 3. In large part, the coding is identical to that used in GASP II models, although there is not complete upward compatability. There is however, virtually complete conceptual compatability. Because of this fact, only selected features will be described in this paper. In order to facilitate understanding of the coding, some of the important GASP IV variables are defined below.

ATRIB (I)	Buffer storage for entries being stored in or removed from NSET.
D(I)	The derivative of the Ith state variable.
DTMAX	The maximum step size used to advance time if any state equations requiring integration are active.
DTVG	The difference between TNOW and TLAST.

ID	Maximum number of entries allowed in NSET.
IEVNT	Event code for state-events.
IM	Number of attributes per entry in NSET.
INN(I)	A code establishing the ranking for file I.
IS(I)	A flag indicating the occurrence of a state-event.
JEVNT	Event code for time-events.
KRANK(I)	The attribute number on which file I is ranked.
LSEV	A code indicating whether state-events may cause discrete changes in the system state.
MFA	The relative address of the first space in NSET available for storing a new entry.
MFE(I)	The relative address of the first entry in file I.
MLE(I)	The relative address of the last entry in file I.
NEQD	The number of derivative equations.
NEQDS	The total number of state and derivative equations. (NEQDS=NEQD+NEQS)
NEQS	The number of state equations.
NOQ	The number of separate files in NSET.
NQ(I)	The current number of entries in file I.
NSET(I) QSET(I)	The filing array for storing all entities and their associated pointers.
S(I)	The Ith state variable.
SL(I)	The value of S(I) at TLAST. SL(I) equals S(I) except during periods when GASP is in the process of advancing time.
SEED(I)	The seed for the Ith stream of the random number generator.
TLAST	The latest time at which all of the state variables were completely updated.
TNEXT	The scheduled time of occurrence of the most imminent time-event.
TNOW	Current simulation time.

SUBROUTINE STATE is a required GASP IV subroutine whose purpose is to define the state variables or their derivatives. GASP IV allows substantial flexibility with respect to the definition of state equations. One method of coding subroutine STATE for this example is given in Figure 3. The statements shown below indicate three possible alternative formulations for this problem.

```
1)  S(I)=SL(I)*(1.-DTVG*RK(I)*PEFF*RON(I))

2)  TRL(I)=accumulated running time for
           present batch in reactor I at
           TLAST.

    TR(I)=TRL(I)+RON(I)*DTVG

    XPNTI=TR(I)*RK(I)*PEFF

    S(I)=SO(I)*EXP(XPNTI)

3)  XPNTI=-RON(I)*DTVG*RK(I)*PEFF

    S(I)=SL(I)*EXP(XPNTI)
```

The coding of subroutine STATE and the above alternatives show three general approaches which may be used: 1) Use of the GASP IV provided Runge-Kutta integrator (as in SUBROUTINE STATE); 2) Construction of an Euler integrator (as in

Alternative 1), or; 3) Use of the closed form solution of the problem (as in Alternatives 2 and 3).

SUBROUTINE SCOND performs the dual functions of setting flags to indicate state-event occurrences as well as causing SUBROUTINE GASP to locate any state-events within a prescribed tolerance. The prescribed tolerance may be on the appropriate state variable, on time, or a combination of both.

SUBROUTINE EVNTS(IX) performs the same functions in GASP IV as in GASP II. The only difference is that in addition to being called for each time-event, it is also called for each state-event. For time-events, the argument passed is the event code (JEVNT=ATRIB(2)) of the event to be processed. For state-events, the argument passed is the user specified event code (IEVNT=3 in this example) for state-events. Substantial flexibility exists with respect to making IEVNT a constant or variable and coding the event logic directly into EVNTS or into event subroutines.

SUBROUTINE SEVNT is the state-event subroutine. In this example, it could easily be coded directly in EVNTS, but is separate in order to clarify its function. SEVENT checks those flags set by SCOND and causes the appropriate events to be processed. An alternative method of processing the events, rather than calling the appropriate event routine directly, would be to schedule the event as a time-event to occur at TNOW. That is, to replace CALL START and CALL STOPP by CALL FILEM(1). This approach allows the user to control the sequencing of events which occur at the same instant of time. Thus, simultaneous events may be processed in any user-defined sequence.

SUBROUTINE START describes the performance of the system at the instant in time that the event occurs. If system pressure is below nominal, it causes the entity representing the associated reactor to be filed in the file awaiting conditions enabling the reactor to be started. If system pressure is above nominal it sets the appropriate counters, flags, and attributes and files the entity in the file awaiting conditions causing it to be stopped.

SUBROUTINE STOPP describes the performance of the system at the instant in time that the event occurs. It first sets appropriate flags and counters to indicate the reactor is turned off. Next it checks to see if the STOPP event is caused by batch completion or low pressure. If it is caused by low pressure, attributes are set and the entity is filed awaiting sufficient pressure to start. If it is caused by batch completion, concentration is initialized for the next batch and the start of the next batch is scheduled (the only time-event in this example) for the appropriate time.

SUBROUTINE SSAVE normally does no more than provide a documentation point. It is called at least once at each event time during periods when there are active state equations. If a discrete change in system state may occur at the event time, SSAVE is called both before and after the potential change. Otherwise, SSAVE is called only once at an event time.

Selected Output from Example Problem

The selected output shown in Figures 4, 5 and 6 gives an example of standard GASP output. Several other forms such as error output,event tracing output and state variable tabular output are not shown.

The initial output, Figure 4, consists of an echo check of input data. Definition of those parameters not given previously may be found in Reference 3.

The output shown in Figure 5 is automatically generated by SUBROUTINE SUMRY. Included are tables of parameter values, statistics collected by subroutine COLCT (time each reactor is down after completion of a batch), statistics collected by SUBROUTINE TMST (number of reactors on), a histogram collected by SUBROUTINE HISTO (time each reactor is down after completion of a batch), and a final dump of both the file and state storage areas.

The output shown in Figure 6, is generated by SUBROUTINE GPLOT. In this particular case it provides a plot of each of the state variables as a function of time. The heading lists the user-specified plot symbol and associated identifier as well as the scale for each variable to be plotted. Thus, the plot symbol "P" represents the system pressure on a scale ranging from 0 to 1000 psia. The symbol " · " is used, in this case, to represent critical pressure (100 psia) and nominal pressure (150 psia). Because the plot interval does not equal the communication interval, multiple plot points associated with the same time frequently occur; specifically, where the time step has been refined to locate a state-event or to obtain more accuracy in integration. The dynamic behavior of the system can be seen clearly in Figure 6. Initially, only reactor 1 was on and pressure rose rapidly until reactor 2 was turned on at time 0.5. Reactor 1 was turned off because of batch completion (a state-event) at about 0.7 hours. (More precise accuracy on event times is readily available through either a table giving every event point, a plot with a non-linear time axis which gives every event separately, or a plot with a linear time axis and reduced plot interval.) Beginning with the start of reactor 3 (a time-event) at time 1.0, pressure fell rapidly. There is an obvious discontinuity in the pressure curve at time 1.5, when reactor 4 was started. Pressure first went critical at about 1.8 hours, causing reactor 4 to be stopped (a state-event) since it was the last one started. From 1.8 until 2.8 hours pressure oscillated several times between critical and nominal. (Critical and nominal pressure are shown by the cursors on the plot.) Because pressure falling below critical and pressure rising above nominal are state-events, there is a plot point for each occurrence which greatly aids analysis. It may be noted that the oscillations in pressure caused reactor 4 to be stopped at 1.8, 2.2, 2.5, and 2.8 hours. (Note, the plot point for pressure being critical at time 2.2 coincides with a plot point for reactor 3; thus it is indicated in the duplicates column.)

Applications

Thus far, GASP IV has been used by the authors and by graduate students in a simulation course to code previously published and locally generated models. In each case, the coding has been accomplished without undue difficulty. The previously published models which have been coded in GASP IV include: 1) An Industrial Dynamics formulation of a production-distribution system (Reference 1, pp. 383-386); 2) World Dynamics (Reference 2, pp. 132-134); and, 3) A mechanical impact, Slip Clutch, problem (Reference 4, pp. 74-76). In each case, the GASP IV model replicated the dynamic behavior of the subject model.

References

1. Fahrland, David Arthur. "Combined Discrete Event Continuous Systems Simulation." Simulation, Volume 14, Number 2, February 1970, pp. 61-72.

2. Forrester, Jay W. Industrial Dynamics. New York: John Wiley and Sons, Inc., 1961.

3. Forrester, Jay W. World Dynamics. Cambridge: Wright-Allen Press, Inc., 1971.

4. Pritsker, A. Alan B. "The Basics of GASP II: A Tutorial," 1971 Winter Simulation Conference, December 8-10, 1971, Waldorf-Astoria, New York, pp. 474-482.

5. Pritsker, A. A. B. and P. J. Kiviat. Simulation with GASP II: A FORTRAN-Based Simulation Language. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1969.

6. Tramposch, H. and H. A. Jones, Jr. "Impact Problems Efficiently Solved With 1130 CSMP." Simulation, Volume 14, Number 2, February 1970, pp. 73-79.