BASIC HYTRAN SIMULATION LANGUAGE—BHSL

Jon C. Strauss *

Electronic Associates Inc.
Princeton, New Jersey

INTRODUCTION

An appropriate subtitle for this paper might read: "A Fortran Compatible Dialect of the SCI Continuous System Simulation Language." Here the words "continuous system" distinguish the application area of interest from that encompassed by the event based simulation languages of the genre of GPSS, SIMSCRIPT, etc. The reference to SCI indicates that BHSL, while certainly a member of the growing family of simulation languages,1,2 was designed to meet standardization guidelines established by the Simulation Software Committee of Simulation Councils Incorporated.3

The Basic Hytran Simulation Language and the associated translator and runtime system are a programming system for the EAI 8400 digital computer. It serves as a basis for the complete Hytran Simulation System on the EAI 8900 Hybrid Computing System.

The primary aim of BHSL is to provide a problem oriented vehicle for the representation (description) of continuous dynamic systems that can be modelled by sets of ordinary differential and/or difference equations in one or more independent variables. The language includes a set of control statements which permit the simulation analyst (programmer) to exercise control over the solution of the equations representing his problem. This control can be programmed into a simulation program or introduced at execution time by use of an associated interactive command and control language system.4

Design Objectives

The Continuous System Simulation Language (CSSL) was designed to incorporate the better features of the previous dynamic system simulation languages. These features include:

1. Automatic sequencing of operations to optimum calculational order,
2. Problem oriented operators and diagnostics,
3. Mnemonic variable and operator naming, and
4. Expression and statement constructions.

The design process was guided by several broad objectives on CSSL programs. These guidelines indicated that CSSL and CSSL programs should be:

1. Easily adapted to various levels of programmer and problem sophistication,
2. Modular (include user written problem
oriented operators in the CSSL vocabulary),
3. Conversational (provide for exception only, conversational interaction with a running CSSL program), and
4. Efficient (contain a variety of user selected and controlled numerical methods).

These considerations are discussed in detail in Reference 3.

BHSL includes all the design requirements of CSSL, but the dependence on a general procedural language has been particularized to Fortran IV. Fortran statements are used directly for sophisticated procedural operations (i.e., formatted I/O, conditional logic, and algebraic processing). In addition, the BHSL processor translates the source program to a valid Fortran IV program for subsequent compilation by the EAI Fortran IV compiler. This technique not only simplifies the implementation task, but it provides for more efficient object code than would be likely in an initial compiler implementation.

Language Features

BHSL can be used as, and contains the desirable features of, the block diagram description languages (e.g. SCADS, MIDAS, etc.). At the next level of problem description sophistication, an equation based notation similar to that of Fortran can be used as in MIMIC and DSL/90. For more sophisticated problems (and programmers), a BHSL program can be expanded to include all the capabilities of Fortran, still retaining the inherent language features of integration, special simulation oriented operators, optimum sequencing of program operations, and problem oriented diagnostic checking. This feature is termed programmable structure.

As a trivial example of the flexibility of the representation aspects of the language, one might choose to describe the differential equation

\[
\frac{d^2x}{dt^2} + a \frac{dx}{dt} + bx = f(t)
\]

in a differential equation oriented form:

\[
DX = \text{INTEG} [DX0, F - B*X - A*DX]
\]
\[
X = \text{INTEG} [X0, DX]
\]

or in an analog computer equation oriented form:

\[
S = - (-F + 10*POT1 + 10*POT2)
\]
\[
DXM = - \text{INTEG} [-DX0, S]
\]
\[
X = - \text{INTEG} [X0, DXM]
\]
\[
DX = - DXM
\]
\[
POT1 = (A*DX) /10
\]
\[
POT2 = (B*X) /10
\]

or one might add new operators to BHSL and describe the problem exactly as though programming an analog computer:

\[
A1 = \text{AIN} [P3,F,,P1,P2]
\]
\[
A2 = \text{AIN} [P4,A1]
\]
\[
A3 = \text{INV} [A2]
\]
\[
P1 = \text{POT} [A,A1]
\]
\[
P2 = \text{POT} [B,A3]
\]
\[
P3 = \text{POT} [DX0, -10]
\]
\[
P4 = \text{POT} [X0, 10]
\]

The most important user feature is the macro. The macro consists of a named set of prototype statements that are inserted into the program each time the name is referenced. During insertion, certain of the variable names are altered in order to maintain the requisite uniqueness properties. The macro is inserted into the source program before the statements are sorted to optimum calculational order. This preserves the true parallel system description aspect of the language.

The simulation oriented BHSL operators, with the exception of the integrator, are mechanized as system macros. The user may also define macros for individual problems to represent commonly used sets of descriptive information. These macros are referenced in his program exactly as though they were operators of the language. In addition, the user has the capability of creating his own macro library which will appear to the system as an extension and/or replacement of the system library. Thus, each user may completely alter the semantics of the language while preserving the syntax and the structure of the operational environment.

The macros defining the problem oriented operators used in the preceding example could be defined as follows:

\[
A1 = \text{AIN} [P3,F,,P1,P2]
\]
\[
A2 = \text{AIN} [P4,A1]
\]
\[
A3 = \text{INV} [A2]
\]
\[
P1 = \text{POT} [A,A1]
\]
\[
P2 = \text{POT} [B,A3]
\]
\[
P3 = \text{POT} [DX0, -10]
\]
\[
P4 = \text{POT} [X0, 10]
\]
System Organization

The language is designed to augment the capabilities of Fortran through the addition of certain problem oriented simulation operators (e.g., integrator), problem oriented syntax (e.g., user defined macros, free format input, etc.), and implicit organizational features (e.g., sorting of statements into optimum calculational order). The translator processes a simulation program which consists of a collection of BHSL statements and Fortran statements by translating to a valid Fortran intermediate program (target program). This program has the requisite structure for interfacing with the runtime system. The monitor calls in Fortran to compile the target program and, under user directive, initiates the execution of the compiled simulation program.

FUNCTIONAL DESCRIPTION OF BHSL ENVIRONMENT

The presentation of a problem oriented language is facilitated by a general discussion of the application area for which it is designed.

Figure 1 presents a block diagram of the calculational flow of a general digital simulation program that involves integration of differential equations in a single independent variable. The various main regions of the program have been denoted as the Initial Region, the Dynamic Region, and the Terminal Region. It is convenient for descriptive purposes to refer to a single run of a simulation program (i.e., solution of the differential equations over the desired independent variable range) as a case. A set of consecutive runs of the same program (with altered parameters or I/O) is referred to as a job.

FUNCTIONAL DESCRIPTION OF BHSL ENVIRONMENT

The presentation of a problem oriented language is facilitated by a general discussion of the application area for which it is designed.

Figure 1 presents a block diagram of the calculational flow of a general digital simulation program that involves integration of ordinary differential equations in a single independent variable. The various main regions of the program have been denoted as the Initial Region, the Dynamic Region, and the Terminal Region. It is convenient for descriptive purposes to refer to a single run of a simulation program (i.e., solution of the differential equations over the desired independent variable range) as a case. A set of consecutive runs of the same program (with altered parameters or I/O) is referred to as a job.

FUNCTIONAL DESCRIPTION OF BHSL ENVIRONMENT

The presentation of a problem oriented language is facilitated by a general discussion of the application area for which it is designed.

Figure 1 presents a block diagram of the calculational flow of a general digital simulation program that involves integration of ordinary differential equations in a single independent variable. The various main regions of the program have been denoted as the Initial Region, the Dynamic Region, and the Terminal Region. It is convenient for descriptive purposes to refer to a single run of a simulation program (i.e., solution of the differential equations over the desired independent variable range) as a case. A set of consecutive runs of the same program (with altered parameters or I/O) is referred to as a job.
interval, initial and final values of the independent variable, and error control.

4. Perform simple arithmetic calculations at the console for computing changes in problem parameters on the basis of past results.

5. Provide initiation control of individual simulation runs (cases) and termination control of a set of runs (job).

Dynamic Region

The dynamic region is that portion of the simulation which takes an active part in the interaction between the digital computer and the external world. It represents all the calculations and I/O operations performed at each user defined discrete value of the independent variable.

The basic interval in the independent variable represented by each pass through the dynamic region is termed the communication interval. This interval is determined solely by the accuracy requirements on the communication with the external world. The interval at which various portions of the calculation (integration) are being updated is generally smaller than the communication interval. This calculation interval is determined strictly by the accuracy requirements on the digital calculation (especially the integration).

Figure 3 illustrates that the dynamic region can be described functionally in terms of input/output and integration.

The input/output subregion constitutes those actions performed in the basic independent variable loop other than the integration calculations (if any).

It mechanizes any time dependent algebraic calculations that are not an integral part of the derivative calculation and also includes all digital input necessary in the dynamic loop and testing of program conditions. These tests might determine whether to: 1) terminate the job or case by transferring control to the terminal region; 2) terminate the case by transferring control to the initial region; or 3) calculate new history information and restart the integration. All output of system variables at the communication rate is mechanized from this subregion.

The Integration subregion includes all integration being performed with respect to the independent variable of the dynamic region. To provide for different integration rates between sets of state variables of a simulation, this may be structured by the programmer to allow an arbitrary number of sections. These sections are generally calls to a system integration routine which integrates a portion of the state vector over a specified interval in the independent variable. Certain of the sections, however, might not involve integration at all. These could be programs simulating portions of parallel synchronous logic which need to be clocked at a different rate than that of the integration.

The specified interval associated with a section is the section communication interval. It would be set larger than the system communication interval for an integration section being updated at a slower rate than the system communication frequency and smaller in those cases where it was necessary to have communication between the variables of differ-
ent integration sections at a higher rate than that of communication. (This would likely be the case when simulating parallel logic.) The various sections are separated by procedural (Fortran) coding for both determining the frequency at which the integration sections are entered and possibly performing simple interpolation on those state variables being updated at a slower rate.

Each section involving integration has associated with it a subprogram for calculating the derivatives of the state variables being integrated. Since there is one such subprogram for each integration section, the term derivative section is used in the description that follows to indicate a section involving integration and its associated derivative subroutine. The system integration package and the various derivative sections share a common symbol table so that there can be direct communication between the various variables and derivatives.

Depending on the integration algorithms employed, there are various combinations of step size, interval alteration and corrector iteration algorithms, etc. that must be specified for each derivative section to assure accurate numerical integration.

Terminal Region

Figure 1 shows that the terminal region receives control from the dynamic region and returns control to the IC (Case) entry. The terminal region contains calculations and I/O necessary to properly terminate a single case. In addition, some system bookkeeping operations such as plot output are performed at this point in the simulation.

DESCRIPTION OF BHSL

Not surprisingly, the gross structure of a BHSL Program and its operating environment bear a marked similarity to the general structure just outlined. The following discussion delineates the specifications for a source program, the resulting object program, and the runtime system.

BHSL Statements

A program is written as a sequence of statements structured (either explicitly or implicitly) into functional groupings termed blocks. The action statements of a block are either representation statements or procedural statements. The statements that delineate the range of a block are structure statements and the statements that indicate how the block is to be processed (both for translation and execution) are control statements.

Although the syntax of each statement type is basically different, the format of all statements on the physical records of the input media is identical. Statements may be started at any position of the physical record and are continued across physical records with an explicit continuator character (\#). Either an end of record or an explicit terminator character (;) serves to terminate a statement.

Representation Statements describe the physical (mathematical) system to be simulated (solved). These statements are the heart of the simulation language; they are similar to the assignment statements of Fortran.

Each statement defines (determines the values of) one or more unique output variables as the result of one or more operators operating on a set of input variables. The variables may be either of type real or logical. In addition to the conventional arithmetic, logical, and relational operators of Fortran, a user expandable set of simulation oriented operators is included in the language system.

For example, in the representation statement:

\[ Y = \text{INTEG} \{X0, A \ast X1 + \cos (X2)\} \]

X is the output variable, X0, A, X1, and X2 are the input variables, and INTEG is a simulation operator.

Procedural Statements are standard Fortran statements that have been couched in the format of BHSL. They are separated physically from the representation statements by block groupings and the translator only performs text editing functions.

For example, the procedural block acts externally as a representation statement, but its actions are defined by standard Fortran statements. The following block represents a limiter with the defining equations:

\[
\begin{align*}
  y &= a \quad \text{for} \quad x \leq a \\
  y &= x \quad \text{for} \quad a < x < b \\
  y &= b \quad \text{for} \quad x \geq b
\end{align*}
\]

\[ \text{PROCEDURAL} \{ Y = A,X,B \} \]

\[
\begin{align*}
  &\text{IF} (X < A) \quad Y = A \\
  &\text{IF} ((A \leq X) \AND (X \leq B)) \quad Y = X \\
  &\text{IF} (X > B) \quad Y = B
\end{align*}
\]

END

The above collection of statements is sorted collectively as a single representation statement with output variable Y and input variables A, X, and B.
Structure Statements delineate explicit structural groupings of the other statement types. These groupings are treated in a different fashion by the translator according to type. The first statement of the preceding example is a structure statement.

Control Statements are utilized to instruct the translator and/or the execution monitor as to how the program should be processed. Had it been desired to use conditional transfers to symbolic labels in the above procedural block, the programmer would declare the labels be used with a LABEL control statement as follows:

```
PROCEDURAL [Y = A,X,B]
  LABEL [L1,L2,L3]
  IF ((X .LT.A) .OR. (X .GT.B)) GO TO L1
  Y = X; GO TO L3; L1: IF (X .LT.A) GO TO L2;
  Y = B; GO TO L3; L2: Y = A; L3: CONTINUE;
END
```

Blocks

Blocks are initiated by an appropriate structure statement and terminated with the END statement.

Regions are the highest level BHSL blocks. The initial region is a set of procedural and control statements that act as the functional equivalent of the initial subregion of Fig. 2. The other functions indicated in Fig. 2 are provided automatically by the translator.

The terminal region also contains only procedural and control statements; it is the functional equivalent of the terminal region in Fig. 1.

The dynamic region is the functional equivalent of Fig. 3 and may contain any of the statement types of BHSL. In general, the statements of this region are structured into derivative and parallel sections separated by procedural and control statements.

Sections contain representation statements and pseudo sections (which act like representation statements, e.g., the procedural block presented above). Certain control statements which specify the error control procedures for the execution monitor are also permitted. Prior to the specification of CSSL,² digital simulation languages, in general, constituted little more than the derivative section of BHSL.

The statements of a section are sorted by the translator to optimal calculational order prior to inclusion in the target program. In general, the sort algorithm requires all the input variables of a statement to have been defined before the statement can be processed (and so define the output variable). Sort loops are broken implicitly by memory type operators such as integrators and time delays and explicitly by certain algebraic iteration operators.

All integration operators are mechanized by the centralized integration system which is contained in the runtime system. The derivative section is translated to a Fortran subroutine which calculates the state variable derivatives. The execution of the subroutine is controlled by the integration routines. The parallel sections are also translated to subroutines, but the control flow is inserted directly into the program. Parallel sections contain no integrate operators; they are generally used to represent a collection of parallel logic elements which must be clocked at a different rate than the integration.

Source Program

A program includes all the statements and/or blocks necessary for both the representation of the simulated system and the control of the simulation itself. A BHSL program has associated with it a set of unique output variables and a unique independent variable. The body of a program may be structured into three types of regions. If no explicit structuring is indicated, the translator assumes that the whole program represents a single derivative section and it inserts a standard central program.

The BHSL translator operates on the source program to produce Fortran IV then compiled by the Fortran IV compiler to yield an object program which interfaces properly with the runtime system.

Object Program

Certain elements of the CSSL do not appear explicitly in BHSL. These features, which include segmentation, multiple independent variables, etc., can be achieved by making Fortran patches in the target program.

Runtime System

In addition to the standard monitor functions (i.e., subroutine loading, priority interrupt scheduling, etc.), the simulation system requires two spe-
clial execution time routines. These are the integra-

tion system and the I/O control system.

The integration system has two entries, one for
initialization and one for integration. In addition to
setting up the initial conditions on the state variables
of the integration, the initialization entry also allo-
cates memory for the history information required
by the particular integration algorithms. The integra-
tion entry transfers control to the appropriate algo-
rithm to integrate the specified derivative section
over its communication interval.

The system includes a variety of algorithms and
error control options which may all be altered at
execution time under interpretive control. The in-
tegration algorithms include Euler, Runge-Kutta,
Third, Fifth and Seventh Order Adams Predictors
and Adams Predictor Correctors. Where applicable,
error control options include adaptive quadrature
error control on individual state variables through
iteration and interval alteration.

The I/O control system is a collection of input/
output routines which includes the interpreter, print
plot, and general output formatting routines.

This interpreter is an upwards compatible version
of the Hytran Operations Interpreter. In its most
simple form, it contains algebraic capabilities and is
able to respond to combinations of appropriate
BHSL control statements. The interpreter provides
readout and data alteration by exception only.

Interpretive Command and Control Features

The system is designed such that a program may
be executed under control of a set of interpretive
instructions. In any single program, this set might
be only a simple initiate execution command or it
might include a string of control statements specifying
alterations to the program and its data. The pro-
grammer specifies the point, if any, in his program
at which he wishes to accept interpretive command
and control information with the INTERPRETER
control statement.

EXAMPLES

The examples of this section are concerned with
the nonlinear two point boundary value problem:

\[ \frac{dy}{dt} = -(1 + e^y) \]
\[ y(0) = 0, \quad y(t_f) = 1.0 \]

This problem was solved as an example in a recent
article describing the MIDAS III simulation lan-
guage. The reader will no doubt find it interesting
to compare the means used to represent the problem
and control the solution in the two languages.

The object of the problem is to determine the un-
known initial condition on \( y \) such that the terminal
condition is met. This constitutes a solution; given
two initial conditions, it is a trivial matter to inte-
grate the equation over the independent variable
range to determine the trajectory \( y(t) \). Questions
of existence and uniqueness of solutions of boundary
value problems, while certainly important, are ig-
nored in this discussion.

Implicit Structure Program

The program presented below is designed to be
used in an interactive fashion for the solution of the
boundary value problem. In the sample interactive
dialog, an analyst exercises this program from a
digital I/O station in much the same fashion that he
would interact with an analog computer.

```
IMPL
  IMPLICIT STRUCTURE EXAMPLE
  HAND OPTIMIZED SOLUTION
  INTAL [RK4]; COMDEL [0.1]; CALF [10];
  TERMVAL [1.0]
  TITLE [TWO POINT BOUNDARY VALUE—J. C. STRAUSS]
  DATA [Y0, DY0/0, 1]
  Y = -INTEG [Y0, INTEG [DY0, 1 + EXP [Y]]]
  SAVE [Y, T]
END
```

The first action taken in an implicit structure pro-
gram is to transfer control to the interpreter. Since
an initial guess for \( y(0) \) is supplied to the program
(via a DATA statement), the analyst responds to
interpreter's request for input with:

```
GO;
```

This command causes control to be returned to the
program which computes a complete solution for \( y \)
on the interval \( 0 \leq t \leq 1 \).

The TERMVAL [1.0] control statement causes
control to be transferred to the interpreter when the
independent variable (T) reaches 1.0. In response to
the interpreter's request, the analyst requests the
terminal value of \( y \) with the statement:

```
Y:
```
To which, in this case, the interpreter responds:

\[ Y = -0.08524 \]

Recognizing that this is too low (the desired \( y(t_f) \) is 1.0) and hoping for a monotonic increasing relationship between \( y(t_i) \) and \( y(0) \), the analyst commands the interpreter:

\[ \text{DYO} = 2.0, \text{GO}; \]

This causes another solution to be run. The process of readout, alteration and run is iterated following a standard binary search algorithm; it is found that a \( y(0) \) of 2.472 yields a \( y(t_f) \) of 1.0001 which is deemed satisfactory.

The SAVE control statement in the program has been storing the results \((Y,T)\) of each iteration. Once the analyst is satisfied with the solution, he types in:

\[ \text{GRAPH} \ [T,Y],\text{CONTROL};\text{STOP}; \]

which produces a plot of \( y \) versus \( t \) (from the last iteration) on the I/O station; the STOP command terminates the job.

**Explicit Structure Program**

The preceding program was designed to assist a human analyst to solve a complicated search problem. The immediate reaction is to program the complete solution using the same algorithm (i.e. binary search). If derivative information were available however (i.e. the dependence of \( y(t_f) \) on \( y(0) \)), a Newton Raphson search algorithm with its associated quadratic convergence could be employed.

As illustrated in Ref. 5, the desired derivative information \( \frac{\partial y}{\partial y(0)} \) is easily obtained by solving the auxiliary differential equation:

\[ \dot{u} = - (e^y)u \]
\[ u(0) = 0, \dot{u}(0) = 1 \]

where \( u(t) = \frac{\partial y(t)}{\partial y(0)} \)

The iterative algorithm employed in the following program involves repetitively solving the differential equations in \( y \) and \( u \) adjusting the initial condition \( \dot{y}^k(0) \) at each iteration \( k \) according to the equation:

\[ \dot{y}^k(0) = \dot{y}^{k-1}(0) + \frac{\dot{y}^{k-1}(t_f) - \dot{y}^{k-2}(t_f)}{\dot{u}^{k-2}(t_f)} \]
BASIC HYTRAN SIMULATION LANGUAGE

14 FORMAT (54HO K .GE. KMAX — KMAX MAY BE INCREASED FROM INTER-
PRETER)
   GO TO 5
4  CALL PLOT (T,X)
5  CONTINUE
END
END

This program is designed to be completely automatic in operation with the additional feature that parameters of the problem such as \( t_f \), \( y(t_f) \), \( y(0) \) etc. can be easily varied at runtime. The interpretive command sequence to run the standard problem (using the data values supplied by the DATA control statement) is:

```plaintext
GO;
{the problem runs and produces satisfactory output}
STOP;
```

The program contains a number of organizational features worthy of more detailed discussion. In particular:

1. In an explicit structure program, the INTERPRETER control statement indicates the position in the program from which control is transferred to the interpreter. The variables specified in the argument list are made available for communication at runtime. In the case of an implicit structure program, the whole symbol table of the source program is passed to the interpreter.
2. The translator automatically closes the control loop around the dynamic region. Hence it is absolutely necessary that the programmer provide explicit termination logic via a procedural statement(s).
3. All output is performed with procedural (Fortran) statements although BHSL control statements could have been used.
4. The iteration logic is programmed such that the solution trajectory is printed out on an extra iteration following satisfaction of the convergence test.
5. Should the maximum number of iterations (KMAX) be exceeded, an error message is printed and control is returned to the interpreter where the analyst can take a variety of actions. Barring just terminating the problem, the simplest action would be to increase KMAX and continue. He could however change the initial guess on \( y(0) \) (DYK) and restart the iterative process.
6. The DEFINE statement in the derivative section names those variables whose values are defined external to the derivative section. There are two reasons for this: 1) The derivative sections are processed first since they determine the storage allocation for the object program. The define statement must specify those variables which are considered to be defined for sort purposes. 2) The translator does not scan the procedural statements and is thus not aware of any variables whose values are determined by procedural coding.

REFERENCES
