A Tutorial on the SIMPLE-1 Simulation Environment

Philip Corbin
Sierra Simulations & Software
303 Esther Avenue
Campbell, California 95008

Overview:

SIMPLE-1 is an integrated modeling environment for interactive simulation using the IBM PC, XT, AT and true compatibles. The system is composed of a full screen editor, file management routines, compiling and run time systems for processing models written in SIMPLE-1. This tutorial will overview the modeling environment and provide an introduction to the SIMPLE-1 modeling language with an emphasis on the animation and advanced statistics features for simulation of manufacturing systems.

The SIMPLE-1 programming language supports modeling discrete and continuous systems world views using a network modeling orientation. Features of the language include the ability of the user to declar variables and statistics requirements, perform I/O operations on files and to animate simulation results in real time easily utilizing built in features of the language. SIMPLE-1 utilizes a repetitive approach to run control to facilitate goal seeking modeling. The language has features particularly suited for modeling assembly operations in manufacturing due to SIMPLE-1's unique approach to managing groups of entities in models of discrete systems.

At the time of presentation this tutorial will be augmented with slides to illustrate features of the SIMPLE-1 modeling environment.

SIMPLE-1 is an integrated modeling environment: Simulation projects inherently involve the integration of many activities and analysis skills to accomplish study objectives. In addition to the obvious requirement to construct, execute and analyze simulation results, data collection and analysis, model validation, and convincing decision makers as to the merits of simulation are important issues in simulation that emerging simulation software must address.

SIMPLE-1 has been implemented as an integrated modeling environment to facilitate simulation related activities by organizing the software into a set of integrated modules for performing the tasks of:

1) Editing models via a full screen text editor coupled to the compiler and run time system's error detection routines.

2) Managing disk files with a function key driven operating system to manage disk directories, active path names and drives, etc.

3) Collection and analysis of data via "toolbox" programs written in SIMPLE-1 to accomplish histogram generation, runs testing for correlation of data sets, Mann-Whitney U test, etc.

4) Interactive compilation of models with errors reported back to the full screen text editor.

5) Interactive execution of models with disk or keyboard input of program variables.

6) Animation of simulation results using SIMPLE-1 language elements to animate results as they occur during the simulation.

7) Interrupt model execution to allow an analyst to review and alter program variables during the simulation. This feature is particularly useful for verifying model execution.

8) On-line tutorials to facilitate learning the system and accessing key documentation quickly. Syntax and theory of operation data on language elements is available on-line.

Design requirements for the software included the avoidance of special hardware. Accordingly, SIMPLE-1 has no special hardware requirements for graphics adapters or special monitor requirements. The software runs on the IBM PC, XT, AT and like compatibles such as the AT&T PC 6300 equipped with either a monochrome or a color monitor.

SIMPLE-1 is an integrated modeling environment: hence it is more than a compiler and run time system for models written in SIMPLE-1. Basically, how SIMPLE-1 works is as follows: Upon execution of SIMPLE-1 the software displays an initial banner then a screen for displaying file related information and executing file commands. Disk directories can be reviewed and the default disk drive and DOS path name for model files can be changed. Files are loaded into the system for editing or compiling by pressing the F1 Function key and inputting the file name for the model. Editing, compiling, and execution of the model are controlled using the key board's built in function keys. Figure 1 is a reproduction of a typical SIMPLE-1 environment display.

All of the various elements of the SIMPLE-1 modeling environment include banners at the top of the screen to show the user how to use the function keys. Much of the software and language documentation is available on-line via tutorial screens. At the top of Figure 1 the function keys: F1, F10, X, C, A, and B keys are listed along with the function associated with each key. To obtain the directory listing of files on the disk displayed in the figure, the "DIR" key was depressed. To load a file called "TV 301.6DL" into memory the F1 was depressed and the file name entered. Once loaded into memory the file could be edited, or compiled & executed by depressing the F7, F9 and F10 keys.
A Tutorial on the SIMPLE_1 Simulation Environment

Figure 3 illustrates the interrupt menu and a listing of the global variables defined for a model.

Reviewing program variables during execution is particularly useful for program debugging and validation. In addition the interrupt feature can be used to allow the analyst to interrupt the simulation and change the values of variables to introduce problems into the model and to see the reaction of the system.

Figure 2 - Full screen text editor display.

Figure 3 - SIMPLE_1 interrupt menu and listing of variables for a sample program.

SIMPLE_1 "Toolbox" programs:

The SIMPLE_1 language and environment support development and use of a "Toolbox" approach to systems analysis. Programs can be written in SIMPLE_1 to collect and analyze data; real or synthetic data. Examples of programs written in SIMPLE_1 to provide a basic "tool kit" include programs to:

1) Collect timing data using the keyboard.
2) Construct Histograms of data sets (with or without a runs test to check for correlation in the data set).
3) Perform the Mann-Whitney test on two data sets.

The histogram program reads a data set and automatically sets histogram cell parameters. Using SIMPLE_1's character based graphics scheme the program interactively displays a histogram of the
data set and allows the user to alter histogram parameters. This program illustrates the capabilities of the software for modeling and analyzing systems. The ability of the user to build "tool box" programs in SIMPLE 1 provides an open ended means of expanding the capabilities of the system. The open ended nature of SIMPLE 1 is a direct consequence of merging simulation language concepts with general purpose programming language concepts common in BASIC, Pascal, C, or FORTRAN.

SIMPLE 1: The Language

SIMPLE 1 employs a number of unique approaches to simulation from a language design point of view. The code is structured into five segments, one of which is a declaration phase. The other four phases of SIMPLE 1 describe the discrete and continuous nature of the model, and run control aspects of model execution.

SIMPLE 1 uses a repetitive approach to run control employing a PRRUN and POSTRUN code sections to set initial conditions and analyze run results. Figure 4 illustrates SIMPLE 1's approach to running the user's model. The PRRUN section of the model is executed first to establish model parameters and run control limits such as the stopping time for the simulation. After execution of the PRRUN code the DISCRETE and/or CONTINUOUS sections of the model are processed. Using SIMPLE 1's repetitive approach to run control one can look at the results of a simulation to base decisions for parameter values of the next run.

Discrete event aspects of the model are defined using an activity on node network structure. The Continuous aspects of the system model are described using algebraic state equations which define variables over time via first order differential equations. The Continuous aspects of the model are simulated using a Runge-Kutta fourth order fixed step procedure with the step size assignable by the modeller. The discrete aspects of the model are processed via an event scheduling mechanism to sequence the flow of entities through blocks in the network model.

START

OF SIMULATION

PRRRUN

CONTINUOUS

RETURN TO

MODELING

ENVIRONMENT

RE-RUN SIMULATION IF STOP

BLOCK NOT ENCOUNTERED IN

POSTRUN

Figure 4 - Schematic of run control in SIMPLE 1

SIMPLE 1 is a declarative language in that the user can define variables. SIMPLE 1 variable identifiers can have up to 20 significant characters including the underscore to facilitate the self documentation of the model. The language supports the declaration of the following classes of data structures:

1) Globally scoped reals: scalars and arrays with single or double subscript.

2) Entities: Entities are declared by name with each type having their own unique number of attributes.

3) Screens: Windows and an associated character schematic to define a background for model animation.

4) Files: File variables to control reading and writing to files and logical devices.

Statistics on globally scoped variables of an observation or time persistent nature are collected automatically by appending key words to the variable declaration. When statistics are declared for arrays the statistics are collected for each element in the array; accordingly SIMPLE 1 models can collect extensive statistics on model variables.

Screens can be declared in SIMPLE 1 which define a character schematic to be used as a background over which animation of the model state is to be performed. SIMPLE 1's approach to formatting of screen images emphasizes a "quick and dirty" approach to minimize modeling effort and overhead.

Entities are created by name in SIMPLE 1 and have their own unique attributes. Entities with Identifiers 1ka: CPU_BOARD and CHIP_SET can be declared in SIMPLE 1 each with differing attribute requirements. CPU_BOARD can be declared to have one attribute and CHIP_SET entities can have five attributes associated with them. Entities are created by name in SIMPLE 1 models and can be brought together into groups. Entities formed into groups do not lose any of their attributes in SIMPLE 1. Manipulation of entity attributes by name simplifies referencing entities traveling in groups and tends to improve the self documentation aspects of SIMPLE 1 models.

The body of a SIMPLE 1 model is composed of five sections: DECLARE, PRRUN, DISCRETE, CONTINUOUS, and POSTRUN. Figure 5 illustrates the organization of SIMPLE 1 model code, the sequence of the segments describes the data structures first, followed by the code segments in their relative order of execution. The DECLARE section is used to define key model variables such as entities, screens, and so forth. The PRRUN and POSTRUN sections execute in a basic subroutine like manner much like BASIC or FORTRAN.

SIMPLE 1 employs seven (7) basic block types to define discrete and continuous models. The brevity of language concepts for discrete system modeling is due to the flexibility of the SIMPLE 1 CONDITIONS block. The network representation, syntax, and brief description of the CONDITIONS and the other SIMPLE 1 basic block types are summarized in TABLE 1.

Discrete system models involve construction of networks defining the flow in time of entities. Conceptually, entities are distinct individual objects that flow through blocks in the network model. Typically, entities are used in models to represent real objects: tools, parts, people, and so forth. The network model is used to define the interrelationship between entities and other elements of the system. In the most basic form, network models describe the processes to:

1) CREATE entities in the model

2) QUEUE entities (in waiting lines) until specified CONDITIONS are met.

170
A Tutorial on the SIMPLE_1 Simulation Environment

<table>
<thead>
<tr>
<th>BLOCK TYPE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>@LABEL ACCEPT @X Y VAR, @L</td>
</tr>
<tr>
<td>ACTIVITY</td>
<td>@LABEL ACTIVITY @DURATION</td>
</tr>
<tr>
<td>BRANCH</td>
<td>@LABEL BRANCH @COND, @LBL1, @COND, @LBL2, \ldots @COND, @LBLn</td>
</tr>
<tr>
<td>CHART</td>
<td>@LABEL CHART @X, @Y, @DIR, @SYM, @CNT, @LIM, @FORE_COLOR, @BACK_COLOR</td>
</tr>
<tr>
<td>CLEAR</td>
<td>@LABEL CLEAR</td>
</tr>
<tr>
<td>CLONE</td>
<td>@LABEL CLONE @QTY, @L</td>
</tr>
<tr>
<td>CLOSE</td>
<td>@LABEL CLOSE</td>
</tr>
<tr>
<td>CONDITIONS</td>
<td>@QNAME @LOCAL @L</td>
</tr>
<tr>
<td>CREATE</td>
<td>@CREATE @Q, @NAME, @TUG, @TF, @CLIN</td>
</tr>
<tr>
<td>REPORT</td>
<td>@LABEL REPORT</td>
</tr>
<tr>
<td>RESET</td>
<td>@LABEL RESET</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYNTAX/FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT @X Y VAR, @L</td>
</tr>
<tr>
<td>ACTIVITY @DURATION</td>
</tr>
<tr>
<td>BRANCH @COND, @LBL1, @COND, @LBL2, \ldots @COND, @LBLn</td>
</tr>
<tr>
<td>CHART @X, @Y, @DIR, @SYM, @CNT, @LIM, @FORE_COLOR, @BACK_COLOR</td>
</tr>
<tr>
<td>CLEAR</td>
</tr>
<tr>
<td>CLONE @L</td>
</tr>
<tr>
<td>CLOSE</td>
</tr>
<tr>
<td>CONDITIONS</td>
</tr>
<tr>
<td>CREATE @Q, @NAME, @TUG, @TF, @CLIN</td>
</tr>
<tr>
<td>REPORT</td>
</tr>
<tr>
<td>RESET</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BLOCK TYPE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF-THEN-ELSE</td>
<td>@LABEL IF CONDITION @SIMPLE 1 @BLKS @IF TRUE @SIMPLE 1 @BLKS @IF FALSE @LABEL END WHILE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYNTAX/FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF-THEN-ELSE</td>
</tr>
<tr>
<td>IF CONDITION @SIMPLE 1 @BLKS @IF TRUE</td>
</tr>
<tr>
<td>IF CONDITION @SIMPLE 1 @BLKS @IF FALSE</td>
</tr>
<tr>
<td>IF CONDITION @SIMPLE 1 @BLKS @IF TRUE</td>
</tr>
<tr>
<td>IF CONDITION @SIMPLE 1 @BLKS @IF FALSE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 1 - SUMMARY OF SIMPLE_1 BLOCK TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
</tr>
</tbody>
</table>
Table 1 - continued

Table 2 - Summary of SIMPLE.j functions
3) **ACTIVITY**: activities are undertaken by entities and involve the passage of time.

4) **BRANCH**: branching of entities between alternative pathways through the network model.

5) **KILL**: Disposal of entities in the system when they are no longer needed in the model.

6) **SET** variable values to describe changes in system state or entity attributes.

```
DECLARE;

DECLARATION OF USER-DEFINED VARIABLES:
1) GLOBAL variables
2) ENTITIES
3) SCREENS
4) FILES

END;

BEGIN;

INITIALIZE RUN:
1) READ/WRITE information to files
2) SET user defined variable values
3) SET run limits; stopping time, entity termination limits
4) CLEAR statistical accumulators

END;

DISCRETE;

MODEL OF DISCRETE PROCESSES IN SYSTEM:
1) Statements based on an activity on node network scheme
2) Character animation of simulation results
3) READ/WRITE data to disk, keyboard, monitor etc.

END;

CONTINUOUS;

MODEL OF CONTINUOUS PROCESSES IN SYSTEM:
Statements use node network scheme to define differential equation models of continuous system elements.

END;

POSTRUN;

ANALYSIS OF RUN RESULTS/RUN CONTROL:
1) Standard/custom reports on simulation results
2) Output of simulation results to files or devices
3) CLEARing of statistical accumulators
4) RESToring of model state
5) Calculation of revised run parameters.
6) STOPping program execution

END;
```

Figure 5 - Schematic Diagram of SIMPLE 1 Code

These six concepts plus a set of advanced modeling concepts comprise the basic building block processes used in SIMPLE 1 discrete system models. Detailed descriptions of all Simple 1 block concepts are available through the on-line tutorials.

**CONDITIONS block**: A key language element

The **CONDITIONS** block defines the state conditions required for entities to leave queues. In a basic queue/server relationship a **CONDITIONS** block is used to associate a specific **QUEUE** with an **ACTIVITY** block. Figure 6 illustrates a fragment from a network model describing the processing of computer mother boards through an insertion activity. A parameter of the **CONDITIONS** block in Figure 6 specifies that the number of active **INSERTION** activities must be less than one (**Idle**) in order for a board to be released from the **QUEUE** labeled **CPU_BOARDS**.

```
"SOningen" CPU_Boards **IDLE** CPU_Boards ... INSERTION INSERTION in ACTIVITY

Figure 6 - Basic Queue/Server SIMPLE 1 network fragment for CPU assembly process model.

In most situations you start off modeling the main processes and add embellishments to capture additional constraints on system operation. In a model of a CPU assembly process we would start modeling with a basic simulation of the CPU's mother board flow through the production process. The assembly aspects of system operation can have a dramatic bearing on the performance of the system and SIMPLE 1 has features especially useful for modeling assembly constraints in models of manufacturing processes. After construction of the initial model of the mother board's processing additional details can be added to the program to model assembly processes. Taking the basic queue/server code, a slight modification to the **CONDITIONS** block will model the assembly of the CPU BOARD with a CHIP_SET entity. To add in an assembly constraint for the operation we would add a queue to store the required chip sets and augment the conditions block. The revised network fragment is illustrated in Figure 7. In the revised situation an entity must be in the CPU BOARDS queue and the CHIP SETS queue as well as an **Idle** **INSERTION** activity in order for the **CONDITIONS** block to route the entities to the **INSERTION** activity. When the criteria for releasing the queues is met the conditions block routes the board and chip set entities to the insertion activity as a group. In the created group the board and chip set entities travel together and keep their unique attribute values, (they do not give up any attributes as a result of traveling together as a group). Figure 8 schematically illustrates the resultant entity group that is ultimately routed to the **INSERTION** activity.

```
"SOningen" CPU_Boards **IDLE** CPU_Boards ... INSERTION INSERTION in ACTIVITY

"CHIP" CHIP_SETS **IDLE** CHIP_SETS ... INSERTION INSERTION in ACTIVITY

Figure 7 - Revised Queue/Server SIMPLE 1 network fragment to model assembly of CHIP_SET and CPU_BOARD entities.

173
In addition to the basic modeling block types `SIMPLE_1` models can employ blocks to manipulate groups of entities created with `CONDITIONS` blocks. The `SPLIT` block allows splitting specific entity types from a group and re-route them elsewhere and the `CLONE` block is useful for creation of exact duplicates of entity groups. As the name implies, the `PREEMPT` block is used to preemption of activities by entities.

Notably absent in the `SIMPLE_1` language is the concept of a resource. The reason `SIMPLE_1` does not employ resources is that by its nature, the `CONDITIONS` block can be used to model simplistic and complex resource situations. Key system resources in `SIMPLE_1` models are typically modeled as entities that are grouped with "customer" entities while in use and `SPLIT` from the customer and routed to a queue when the resource entity becomes idle. The advantage inherent in modeling resources as a separate type of entity in `SIMPLE_1` models is the ability to model explicitly the decision making processes of the resource. `SIMPLE_1`'s handling of complicated resource situations is in a fashion a highly generalized version of the selector node concept for resource modeling employed in MPS.

`SIMPLE_1` employs four specialized blocks for run control purposes. A `CLEAR` block is used to control clearing statistical accumulators and a `RESTART` can be used in the `POSTRUN` to eliminate all entities in existence in the discrete portion of the model. A standard report on system performance can be obtained using the `REPORT` block in the `POSTRUN`. The key run control block in `SIMPLE_1` is the `STOP` block. The `STOP` block is used in the `POSTRUN` to halt model execution and return to the main `SIMPLE_1` environment.

An original `GPSS` example of a basic TV inspection and adjustment situation illustrates how `SIMPLE_1` code is written. In this example we have TV's arriving to be inspected by one of two available inspectors. After inspection good sets are routed to shipping and defective sets are routed to an adjusting station. At the adjusting station the sets are re-aligned by a single adjustor and routed back to the inspectors for re-testing. Using Schriber's `GPSS` TV inspection and adjustment example the `SIMPLE_1` code for the model would be:

```plaintext
DECLARE;
    GLOBALS: TIME_IN_SYSTEM OBSERVE_STATS;
    ENTITIES: TV(1);
END;
PRERUN;
    SET STOP_TIME:=1440;
END;
DISCRETE;
    CREATE,1,T,TV,UNIFORM(3.5,5,7.5,1);
    SET TV(1):=STIME;
WAIT_INSP QUEUE,FIFO;
    CONDITIONS,
    NUM(INSP)',<2,WAIT_INSP,INSPECT;
INSPECT
    ACTIVITY UNIFORM(6,12,1);
    BRANCH 0.85,PACK:
    0.15,WAIT_ADJ;
WAIT_ADJ QUEUE,FIFO;
    CONDITIONS,
    NUM(ADJUST)<1,WAIT_ADJ,ADJUST;
    ACTIVITY UNIFORM(20,40,1);
    BRANCH,WAIT_INS;
PACK
    SET TIME IN SYSTEM:=STIME-TV(T(1));
    KILL;
END;
CONTINUOUS;
END;
POSTRUN;
    REPORT;
    STOP;
END;
```

The global variable `TIME IN SYSTEM` is declared with the key word `OBSERVE STATS` appended to signal collection of statistics. When the set block near the bottom of the code assigns the value of `TIME IN SYSTEM` with the expression:

```plaintext
    TIME IN SYSTEM:=STIME-TV(1)
```

The creation time for the TV and the current simulation time (`STIME`) are used to calculate the time in the system for the exiting TV. As a side affect of the the assignment `SIMPLE_1` updates observational statistics for `TIME IN SYSTEM`.

The `CONDITIONS` blocks in this model employ a built in function `NUM` which returns the current number of entity groups currently at a block in the model. `NUM` is one of an extensive number of built in `SIMPLE_1` functions available to the modeler. Built in functions of the language provide access arithmetic functions, random number generators etc. Table 2 is a summary of `SIMPLE_1` functions.

### Input, Output and Animation:

The `SIMPLE_1` simulation language has input and output concepts for both file I/O and screen animation with the screen being updated while the model is running. `SIMPLE_1` supports I/O operations using specialized block constructs. The input and output operations supported in the language are for two types of operations. Block constructs in the language control I/O to the screen or keyboard and to DOS. Screen I/O constructs include mechanisms for writing ASCII characters and numbers coupled with template images. The character and number based display formats of `SIMPLE_1` combined with screen generation features of the language form a character based animation capability. In summary, `SIMPLE_1` supports file and screen I/O Operations associated with:

---

174
A Tutorial on the SIMPLE 1 Simulation Environment

1) SCREEN activation to display a text background.
2) SHOW block to display numeric values on a screen.
3) CHART block to display characters on a screen.
4) ACCEPT block for reading variable values from the keyboard.
5) READ and WRITE blocks for file input/output.
6) OPEN and CLOSE blocks for managing files during program execution.

VIDEO COLOR FIELDS

The screen I/O blocks: SCREEN, SHOW, and CHART have two optional fields to select the foreground and background colors to use on machines with a color monitor. The fields are optional and specify the foreground and background color to use when writing to the screen. Integer numbers are used to turn on specific colors as defined by the color numbers:

0: Black 6: Brown 11: Light Cyan
1: Red 7: Light Gray 12: Light Red
2: Green 8: Dark Gray 13: Light Magenta
3: Cyan 9: Light Blue 14: Yellow
4: Red 10: Light Green 15: White
5: Magenta

Revising the TV inspection and adjustment example illustrates the I/O concepts of SIMPLE 1 for both character animation of the simulation and generation of disk files. A screen will be used to form a schematic of the TV inspection system. SHOW and CHART blocks will be used to animate the state of the system using the schematic diagram of the system as a background. Figure 9 is a listing of the revised code for the TV repair model.

In the DECLARE section a CONTROL entity type has been added for managing the animation of the screen on 10 time unit intervals. The screen named PICTURE is associated with a schematic of the system. A FILE declaration is made in the DECLARE section to define file variable OUT1. OUT1 will be used to store time in system observations.

During the PRERUN phase an OPEN block will open "HIST.DAT". When a TV completes processing the length of time spent in the system by the TV will be written to the file for post processing with the histogram analysis program implemented in SIMPLE 1 that is supplied with the software. A CLOSE block is used in the POSTRUN to close the disk file when the model is finished. Prior to returning to the SIMPLE 1 modeling environment menu a standard report on run results is obtained using the REPORT block. The REPORT block at run time allows reports to be written to the screen or to file.

A CONTROL entity is created every 10 time units in the DISCRETE section to manage updating the screen. The CONTROL entity executes a series of SHOW and CHART blocks. The SHOW blocks are employed to write numbers for the time, queue sizes etc. The CHART blocks are used to write ASCII characters. The number of ASCII characters written by the CHART block is used to graphically represent the number of busy inspectors and adjusters in the system. In effect, the animation of simulation results using the CONTROL entity causes "SNAP SHOTS" of the system to be taken on fixed time intervals. Alternatively, SHOW and CHART blocks can be inserted between ACTIVITY and QUEUE blocks to update the screen as specific portions of the system change state. This alternative method produces screen results that are generally more active and representative of the activities being simulated however, additional coding overhead is generally required.

DECLARE;
GLOBAL: TIME IN SYSTEM OBSERVE STATS;
ENTITIES: TV(T); CONTROL(1);
DEF SCREEN: PICTURE,1,1,80,16,YES;
+ -

TIME:

---

11 TV INSPECT/ADJUSTMENT EXAMPLE 11

---

INSPECT TV

---

ADJUSTOR

---

***********

STATION

CREATE:

- NO -

TV -->

*** INSPECTORS --> PACKING

TOTAL:

** INPUT **

TOTAL:

***********

FILES: OUT1, WRITE;

END;

PRERUN;

OPEN, OUT1 AS HISTO IN;
SCREEN, PICTURE, 1, 15, 0;
SET STOP (TIME)=1440;

END;

DISCRETE;

CREATE, 1, CONTROL, 10, 10;
SHOW, 36, 2, TIME, 2, 12, 0;
CHART, 27, 1, 4, 177, IN(WAIT, INSPECT), 12;
CHART, 57, 6, 4, 177, IN(WAIT, ADJUST), 10;
CHART, 38, 15, 4, 179, IN(INSPECT), 2;
CHART, 61, 7, 4, 178, IN(ADJUST_Q), 2;
SHOW, 65, 13, COUNT(PACK), 5, 0;

KILL:

CREATE, 1, TV, UNIFORM, 1, 5, 5, 11;
SET TV(1) := TIME;

WAIT INSPECT

QUEUE, FIFO;

CONDITIONS, NUM(INSPECT) < 2;

WAIT INSPECT, IN(INSPECT);

INSPECT

ACTIVITY, UNIFORM(6, 12, 11);

BRANCH, 0.65, PACK;

0.15, WAIT ADJUST;

WAIT ADJUST

QUEUE, FIFO;

CONDITIONS, NUM(ADJUST Q) < 1;

WAIT ADJUST, IN(ADJUST Q);

ADJUST Q

ACTIVITY, UNIFORM(30, 40, 11);

BRANCH, WAIT INSPECT;

PACK

SET TIME IN SYSTEM := TIME-TV(1);

WRITE, IN, TIME IN SYSTEM, 8, 3;

KILL;

END;

CONTINUOUS;

END;

POSTRUN;

CLOSE, OUT1;
REPORT;
STOP;

END;

Figure 9 - Revised SIMPLE 1 code for TV repair model.
Running this example will produce the file: HISTO.LNP which contains the individual time in system observations for TV's. Using a histogram program written in SIMPLE I a run was performed on the data and histogram generated. The histogram results are illustrated in figure 10. The report generated by the REPORT block was saved to a disk file and is reproduced in figure 11.

---

**Figure 10** - Histogram generated from data created by TV model. Results were obtained using a 160 line program written in SIMPLE I.

---

**Figure 11** - SIMPLE I standard summary report generated by TV model.

Simple I will model continuous systems definable as a set of first order differential equations. A simple rocket model illustrates SIMPLE I's approach to continuous modeling. The height of the rocket attained over time will be integrated and is based upon the initial fuel load of the rocket. In this example we would define velocity height, weight etc. in the declare section. The SIMPLE I key word INTEGRATED follows the declaration of variables whose values are obtained by numerical integration. SIMPLE I integrates continuous variables using a Runge-Kutta fourth order fixed step procedure. The SIMPLE I code for this example is illustrated in figure 12.

---

**Figure 12** - SIMPLE I model of a simple ROCKET.
A Tutorial on the SIMPLE_1 Simulation Environment

A benefit of SIMPLE_1's DECLARE section is the ability to define and use variables with identifiers related to the physics of the problem such as height, velocity, drag, etc.

In this model the PRERUN establishes the initial state variables prior to the run. A discrete section is used to periodically update the monitor to display the rocket's state over time both numerically and using the character graphics capabilities of the language. Figure 13 illustrates the information displayed on the monitor while execution of the model is progressing.

![Figure 13 - Screen display during execution of rocket model.](image)

Applications of SIMPLE_1:

Since announcement of SIMPLE_1 at the 1985 Winter Simulation Conference held in San Francisco, SIMPLE_1 has been applied in manufacturing, academia, and by the United States Military. Applications of SIMPLE_1 to date have ranged from manufacturing systems, robotics justification, health care systems, emergency planning, and analysis of logistic support systems.

Summary

SIMPLE_1 has a number of innovative features not found in current simulation software. The system combines a full screen editor with compilation and run time systems to speed up the edit-debug cycles involved in model building. The language supports a "tool box" ability whereby support programs can be written in SIMPLE_1 to post process simulation data. SIMPLE_1 utilizes a built in capability to animate simulation results using a character graphics methodology which stresses a "quick and dirty" approach to model animation. The language supports reading and writing of data sets via standard ASCII text files in addition to the animation and key board data input capabilities. SIMPLE_1 is not just a pretty picture: the language supports extensive collection of statistics. Statistics collection capabilities of SIMPLE_1 include the ability to easily obtain statistics on user defined arrays.

The implementation of SIMPLE_1 combines the compilation and run time systems of the software into an integrated environment. The SIMPLE_1 environment includes on-line tutorials and full screen editor coupled to the compiler and run time system. Errors detected by the compiler or run time system initiate a signal to the editor to isolate the error and speed up the edit-compile-debug cycle of modeling.

References


Philip Cobb in is the owner of Sierra Simulations & Software and is the developer of SIMPLE_1. Phil has developed and taught simulation to undergraduates as an adjunct professor of Industrial Engineering at San Jose State University. He holds a Master of Science in Industrial Engineering from Purdue University, a Bachelor of Science in Industrial Engineering and Operations Research from the University of Massachusetts at Amherst, and an Associate in science degree in Manufacturing Engineering Technology from Waterbury Connecticut State College. Phil is a native of Los Angeles and has been previously employed by the General Products Division of the International Business Machines corporation performing simulation modeling and material handling engineering activities.