GEST - A MODELLING AND SIMULATION LANGUAGE BASED ON SYSTEM THEORETIC CONCEPTS*

Tuncer I. Ören
Computer Science Department
University of Ottawa
Ottawa, Ontario, K1N 9B4
Canada

ABSTRACT

GEST is the first model and simulation specification language. Specifications of the model and the experiment are totally separated. The modelling world view is based on the axiomatic system theory of Wymore which provides an excellent basis for simulation modelling and symbolic model processing. This chapter has two aims: 1) To present the GEST language and the robust and rich modelling paradigm it provides even for non-simulation application areas, as well as 2) to foster design and development of other GEST-like modelling and simulation languages which would provide other modelling formalisms within comprehensive modelling and simulation systems.

----

* This study has been sponsored by the Natural Sciences and Engineering Research Council of Canada (NSERC) by the Operating Grant A8117.

NATO ASI Series, Vol. F10
Simulation and Model-Based Methodologies:
An Integrative View. Edited by T. I. Ören et al.

1. INTRODUCTION

GEST is a modelling and simulation language based on general system theoretic concepts. It was conceived in 1969 and first document about it was published in 1971 (Ören 1971). Its definition has been updated a few times. The last version of it is GEST 81 (Oren 1981, 1982a). GEST, even at its first version, departed radically from other simulation languages. It is the first model-based simulation language. In GEST, specifications of the model and the experiment are totally separated. The modelling world view of GEST is based on the axiomatic system theory of Wymore (1967, 1976).

Already other GEST-like languages exist. One such language is SEMA (SEquential MAchines) (Ören and Collie 1980). Another GEST-like language is designed by Subrahmanian and Cannon (1981). Futo and Gergely (1982) developed TS-PROLOG, an advanced modelling and simulation language based on the concepts advocated by Ören and Zeigler (1979).

This chapter has two aims: 1) To present the GEST language, and 2) to foster design and development of other GEST-like modelling and simulation languages which would provide other modelling formalisms within comprehensive modelling and simulation systems. The rationale for developing such advanced tools within comprehensive modelling and simulation software systems is given in chapter 1 of this volume (Ören 1984).

A list of references where GEST is treated directly, or where GEST has been referred to is given at the end of this chapter.

2. WORLD VIEW OF GEST 81

GEST is a model and simulation specification language. Therefore a GEST program is highly descriptive and acts as a documentation (for communication among humans) as well as a specification (for man-machine communication). This documentation ability of GEST will become apparent in the sequel. If looked only superficially, some elements of GEST may appear to be cumbersome to specify, such as "END COUPLING FOR model-identifier." However, GEST has to be conceived differently.

First of all GEST, due to its world view is a good basis for a comprehensive modelling and simulation software system. Within such a system, the computer-assisted modelling module would have all the necessary information to generate "END COUPLING FOR model-identifier" as soon as the specification of all the input-output relationships have been completed by the user, based on system-initiated prompts.

Conceived within a computer-assisted modelling system, some GEST instructions are totally or partially generatable by the system for the convenience of the user. Thus, in addition to be user-friendly and convenient, computer assistance in specifying GEST models can assure completeness as well as consistency checks of the specifications. Another model-based language where user input is minimized through computer-assistance is SEMA (Oren and Collie 1980).

In GEST, a program consists of three distinct parts, i.e.,

1) Mathematical model,
2) Experiment(s), and
3) Output module(s).

The "model" consists of a parametric model and associated set(s) of parameter values. The "experiment" is the specification of experimental conditions (or experimental frames) which have to to applied to a model. The "output module" is the specification of the output program to be used to display the result of the simulation study.

Expressed in Backus-Naur Form (BNF) the definition of a GEST program is as follows: (The meta-language used to describe GEST, is given in the appendix.)

```
program =
  "PROGRAM" identifier
     model
     experiment
     output-module-specifiation
  "END PROGRAM" identifier ";" .
```

3. MODEL

3.1 Background

A specific GEST model, is a pair of parametric model and model parameter set. As seen in Figure 1, consists of two parts: 1) a parametric model, and 2) model parameter set(s). The optional model parameter set may appear more than once. A BNF specification of "model" follows:

```
model =

    parametric-model-specification
    [{ model-parameter-set }] .
```

Parametric Model: A parametric model associated with a parameter set constitutes a specific model that one can use in a simulation study.

A modeller, during the formulation of a parametric model, needs only to specify the names of the parameters of a model. At this stage, the actual values of the parameters need not be specified.

A parametric model may consist of one or several component model(s). A component model may be continuous, discrete, or memoryless. A coupled model consists of a set of component model(s) and their input/ output interface which is also called the coupling specification.

Component Model: A component model consists of two sections: In the first section the **static structure** (or the descriptive structure) of the model, is expressed. In the second section the **dynamic structure** (or the predictive structure) of the model is specified. This latter section consists of the state transition and the optional output function(s).

The static structure of each component model is described basically in terms of model descriptive variables such as state-, input-, and output-variables. However, autonomous models, by definition, do not need inputs to operate. And in some cases explicit output variables may

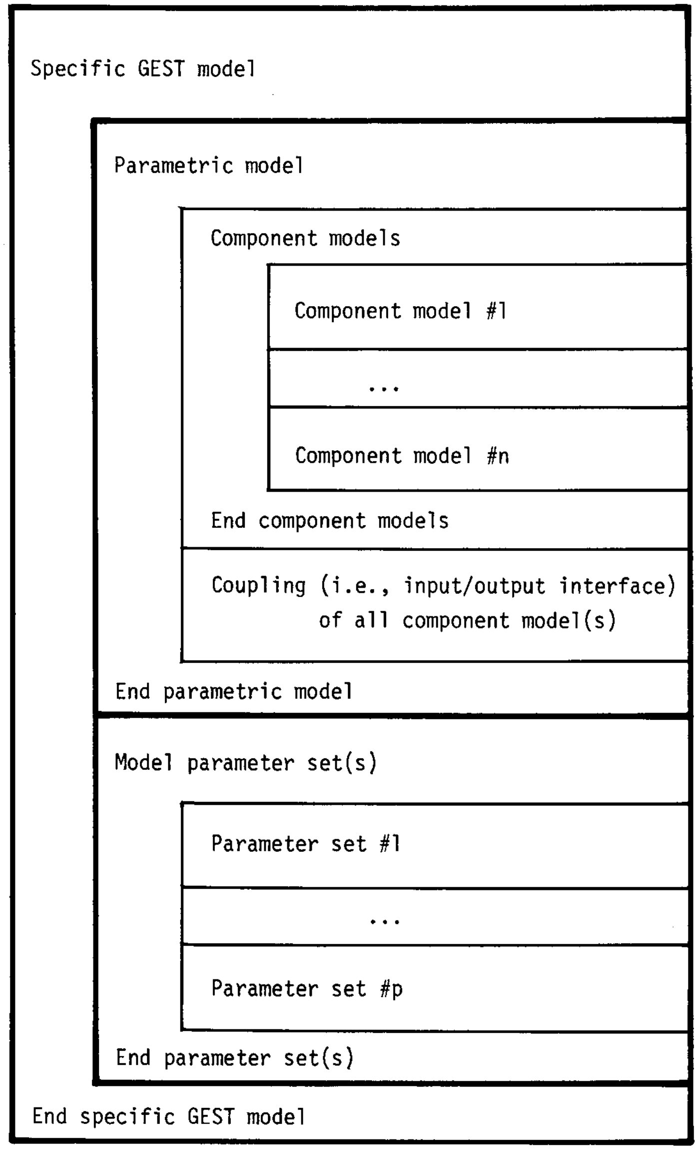


Figure 1. Parts of a specific GEST model

not exist. In this last case, some or all of the state-variables and/ or auxiliary variables may be considered to be the output-variables of the model. In memoryless component models, state variables do not exist. In this case, current output is computed based on the values of the current inputs.

Furthermore, the static structure of a model requires other declarations, such as type and range of values of the descriptive variables of the model. Several modelling formalisms can be used to express component models, such as ordinary differential equations with or without discontinuities in their state-variables and/or their derivatives, difference equations, or combined continuous and discrete-change models.

3.2 Continuous model

Basics

As is shown in Figure 2, the specification of a continuous component model consists of two parts: 1) The static structure and 2) The dynamic structure of the model.

The specification of the static structure of a continuous model consists basically of the declaration of the descriptive variables of the model under the following categories:

input variable
state variable
output variable
auxiliary variable
constant
parameter
auxiliary parameter
tabular function declaration, and
interpolated variable declaration

```
CONTINUOUS MODEL model-identifier

    STATIC STRUCTURE

        INPUT  declaration  (some or all stochastic)
                            [range declaration]
        STATE  declaration  [range declaration]
        OUTPUT declaration  [range declaration]
        AUXILIARY VARIABLE declaration [range declaration]

        CONSTANT declaration
            constant assignment
        PARAMETER declaration (some or all stochastic)
                              [range declaration]
        AUXILIARY PARAMETER declaration
            auxiliary parameter computation
        TABULAR FUNCTION declaration
            interpolated variable declaration

    END STATIC STRUCTURE

    DYNAMIC STRUCTURE

        DERIVATIVES

                statements

        END DERIVATIVES

        OUTPUT FUNCTION          (optional)

                statements

        END OUTPUT FUNCTION

    END DYNAMIC STRUCTURE

END MODEL model-identifier
```

Figure 2. Layout of a continuous component model specification in GEST

The type of every descriptive variable can be specified separately. The default type is accepted to be real. The ranges of the values of the descriptive variables can also be specified as part of a model in order to enforce some automatic consistency checks. Both external input variables (those variables which are not provided by some component models of a system) and parameters can be stochastic. In this case, it is possible to declare the distribution function to be used to generate them. Another possibility is the ability of declaring tabular functions and the associated interpolation requirements.

The dynamic structure consists of two blocks, i.e. 1) The derivative block which contains the specifications of the derivatives of the state variables and the computations of the necessary auxiliary variables. 2) The output block contains the transformations of the state and/or auxiliary variables into output variables.

Examples:

Some elementary examples of continuous models expressed in GEST are given in Figures 3 and 4.

```
CONTINUOUS MODEL MIXED_LOGISTIC_GROWTH
   STATIC STRUCTURE
      STATES  Y1, Y2;
      OUTPUTS Y1, Y2;
      PARAMETERS R1, R2, A1, A2, B1, B2;
   END STATIC STRUCTURE;

   DYNAMIC STRUCTURE
      DERIVATIVES
         Y1' = R1*Y1*(1.0 - A1*Y1 - B1*Y2);
         Y2' = R2*Y2*(1.0 - A2*Y2 - B2*Y2);
      END DERIVATIVES;
   END DYNAMIC STRUCTURE;
END MODEL MIXED_LOGISTIC_GROWTH;
```

Figure 3. A continuous model expressed in GEST

```
CONTINUOUS MODEL FLOW
   STATIC STRUCTURE
      INPUT FLOW_IN;
      STATE  VOL;
      OUTPUT VOL;
      AUXILIARY VARIABLES FLOW_OUT, NIV;
      PARAMETERS R, S;
   END STATIC STRUCTURE;

   DYNAMIC STRUCTURE
      DERIVATIVES
         VOL' = FLOW_IN - FLOW_OUT;
         FLOW_OUT = NIV/R;
         NIV      = VOL/S;
      END DERIVATIVES;
   END DYNAMIC STRUCTURE;
END MODEL FLOW;
```

Figure 4. A continuous model expressed in GEST

Some relevant definitions in BNF follow:

```
continuous-model =

   "CONTINUOUS MODEL" model-identifier

       "STATIC STRUCTURE"

          [ {input-declaration}      [{ range-declaration }] ]
            {state-declaration}      [{ range-declaration }]
          [ {output-declaration}     [{ range-declaration }] ]

          [ {auxiliary-variable-declaration}
                                     [{ range-declaration }] ]

          [ {constant-declaration}
                {constant-assignment} ]

          [ {parameter-declaration} [{ range-declaration }] ]
          [ {auxiliary-parameter-declaration}
                                     [{ range-declaration }] ]
                {auxiliary-parameter-computation} ]

          [ {tabular-function-declaration}
                {interpolated-variable-declaration} ]

       "END STATIC STRUCTURE" ";"

       "DYNAMIC STRUCTURE"

           "DERIVATIVES"
                {statement}
           "END DERIVATIVES" ";"

         [ "OUTPUT FUNCTION
                {statement}
           "END OUTPUT FUNCTION" ";" ]

       "END DYNAMIC STRUCTURE" ";"

   "END MODEL" model-identifier ";" .
```

Definitions of type, random-variable-declaration, and range, expressed in BNF follow:

```
type = "INTEGER" | "REAL" | "BOOLEAN" | "LITERAL" .

random-variable-declaration =
   "RANDOM" ["REAL"|"INTEGER"] list-of-scalar-variable ";"
       { variable "=" distribution-name "(" list-of-parameter ")" ";" }
   "END RANDOM" ";" .

range-declaration =
   ( "RANGE OF" (scalar-variable | array-variable) "=" type
        "(" range-limit ")" ";" )
 | ( "RANGE OF" scalar-variable "=" "LITERAL"
        "(" list-of-identification ")" ";" )
 | ( "RANGE OF" array-variable "="
        "ARRAY" ("REAL" | "INTEGER" ) array-variable ";" ) .

range-limit =
    ([boundary] ".." boundary)  (boundary ".." [boundary]) .

boundary =  real | integer | scalar-variable .
```

The upper and lower boundaries of a range can be given in a model. In this case they are considered absolute and can not be modified in an experimental frame. However, if the boundary values may depend on experimental conditions they may be declared as parameters to be specified in experimental frames. In the last case, the range of their acceptable values should be given as the range of the associated parameters.

Examples:

```
RANGE OF STATUS = LITERAL(ON,OFF);
RANGE OF D      = REAL( .. 32.5 );

RANGE OF PRIORITY = INTEGER ( 1 .. MAX_PRIORITY );
RANGE OF VELOCITY = REAL    ( 0.0 .. 327.4 );
```

Definitions of input, state, and output variables follow:

```
input-declaration =
    ( "INPUT"["S"] [type] list-of-variable ";" )
  | ( "INPUT"["S"] random-variable-declaration ";" ).
```

Examples:

```
     INPUTS A,B,C;
        RANGE OF A = REAL ( 0.00 .. );
        RANGE OF B = REAL ( .. 32.80);
        RANGE OF C = REAL ( 0.00 .. 100.00);
     INPUTS ARRAY REAL K(1..15), L(1..10), M(1..15);
     INPUT AR, P(1..20);
     INPUT SWITCH;
        RANGE OF SWITCH = LITERAL (ON, OFF);
     INPUTS RANDOM REAL CUS_ARRIVAL, X;
                CUS_ARRIVAL = EXPONENTIAL(LAMBDA);
                X = NORMAL(XM,XS);
            END RANDOM;
```

```
state-declaration =
    ( "STATE"["S"] [type] list-of-variable ";" ).
```

Examples:

```
     STATE  POPULATION;
     STATES YEAST, ALCOHOL;
     STATE  MASS_FLOW_DISCHARGE;
     STATE  REAL POPULATION (1..10);
```

```
output-declaration =
    ( "OUTPUT"["S"] [type] list-of-variable ";" ).
```

Example: `OUTPUTS REAL L (1..15), N(1..LIM_N);`

Output variables do not depend on input variables. State variables or auxiliary variables which are used as output must be explicitly declared.

Definitions of auxiliary variables, constants, and parameters follow:

```
auxiliary-variable-declaration =
   "AUXILIARY VARIABLES" list-of-variables ; .
```

```
constant-declaration =
   "CONSTANT"["S"] [type] list-of-scalar-variable ";" .
```

Example:

```
      CONSTANTS REAL K, L ;
```

```
constant-assignment =
   scalar-variable "=" arithmetic-expression ";" .
```

Examples:

```
PI         = 3.1415926;
P80        = PI/180.0;
LATITUDE   = 52.0;
LONGITUDE  = -5.0;
```

```
parameter-declaration =
   ("PARAMETER"["S"] [type] list-of-variable ";" )
 | ("PARAMETER"["S"] random-variable-declarations ).
```

Examples:

```
     PARAMETER  REAL PA(1..3,1..5), P1;

     PARAMETER
         RANDOM REAL KE, KD;
             KE = NORMAL(200., 20.);
             KD = NORMAL(XM, SD);
         END RANDOM;
```

Auxiliary parameters are defined in terms of parameters and constants. Once the values of the parameters are given, the values of the auxiliary parameters can be computed by the system and made available for the convenience of the user.

```
auxiliary-parameter-declaration =
    ( "AUXILIARY PARAMETER"["S"] [type]
          list-of-variable ";"
              { auxiliary-parameter-computation }
      "END AUXILIARY PARAMETER" ";" ).
```

Example:

```
     AUXILIARY PARAMETERS  K, L;
         K = PI/OMEGA;
         L = 1.ØØ/DIST;
     END AUXILIARY PARAMETER;
```

```
auxiliary-parameter-computation =
     (scalar-variable = arithmetic-expression ";")
   | for-statement
   | (identifier "(" list-of-integer ")"        }
         "=" arithmetic-expression ";" ) .
```

Auxiliary parameters may depend on constants, parameters, and other auxiliary parameters.

```
tabular-function-declaration =
     ["DISCONTINUOUS"]
      "TABULAR" ( "FUNCTION"["S"] | "FUNCTION_2")
      list-of-function-name ";" .
```

DISCONTINOUS and FUNCTION_2 introduce discontinuous and two-dimensional tabular functions, respectively.

Examples:

```
TABULAR FUNCTIONS GROWTH_RATE, FK;
TABULAR FUNCTION_2 DIFF_COEF;
DISCONTINUOUS TABULAR FUNCTION RAIN;
```

```
interpolated-variable-declaration =
    ( "INTERPOLATION"
          scalar-variable "=" function-name
          "(" list-of-scalar-variable ")" ";" )
  | ( "INTERPOLATIONS"
        { scalar-variable "=" function-name
          "(" list-of-scalar-variable ")" ";" }
      "END INTERPOLATION" ";" ) .
```

Examples:

```
INTERPOLATION   RATE = CURVE_1 (TEMPERATURE) ;

INTERPOLATIONS
    RATE1 = ABC (T);
    RK    = FUN (T);
END INTERPOLATIONS;
```

3.3 Discrete and Memoryless Models

Basics

Both discrete and memoryless models, like continuous model, have two parts: 1) The static structure and 2) the dynamic structure.

Discrete models allow specification of systems expressed by a set of first order diffreence equations.

In discrete models, the static structure is like the static structure of a continuous model. However, a discrete model differs from a continuous model in the dynamic structure where the derivative block is replaced by the following block:

```
STATE TRANSITION
    statements
END STATE TRANSITION;
```

A **memoryless model** differs in two ways from a continuous model: 1) The static structure does not have declaration of any state variable, and 2) the dynamic structure has only output function specification.

A memoryless model transforms instantaneously its inputs and parameters into some output. For the convenience of naming, if a memoryless model has one output only, the same name can be used to designate the model and its output.

Example:

```
MEMORYLESS MODEL BIRTH_RATE

    STATIC STRUCTURE
        INPUTS
               S,    (* MATERIAL STANDARD OF LIVING  *)
               NE,   (* EFFECTIVE POLLUTION          *)
               P;    (* POPULATION                   *)
        OUTPUT BIRTH_RATE;

        PARAMETERS K2Ø, K21, K22, K23, K24, K25;

    END STATIC STRUCTURE;

    DYNAMIC STRUCTURE

        B = K2Ø - K21*S - K22*NE - K23*P;
        IF B >= K24 AND B <= K25
           THEN BIRTH_RATE = B;
           ELSE IF B < K24
                    THEN BIRTH_RATE = K24;
                    ELSE BIRTH_RATE = K25;
                END IF;
        END IF;

    END DYNAMIC STRUCTURE;

END MODEL BIRTH_RATE;
```

Figure 5. A memoryless model expressed in GEST

3.4 Coupling

Basics

A coupled model (or a resultant model) consists of a set of component model(s) and their coupling which specifies input/output relationships of the component models.

An example of a coupled model is given in Figure 6. A GEST representation of the coupled model represented in Figure 6, is provided in Figure 7.

The layout of a coupled model specification in GEST is given in Figure 8.

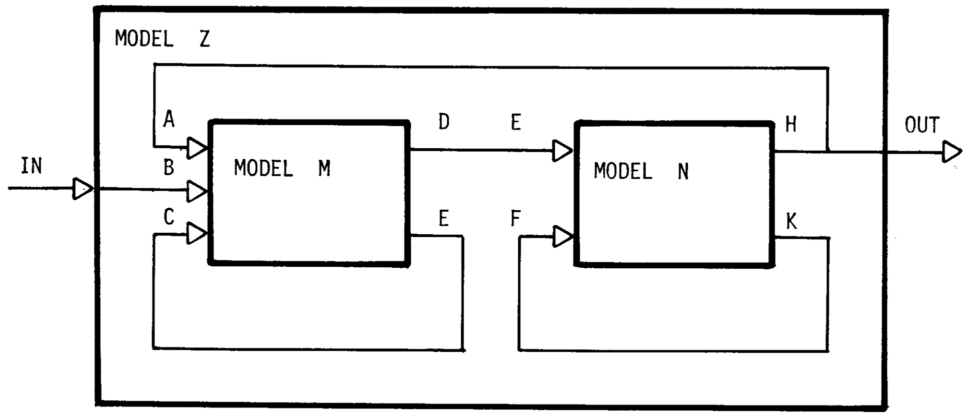


Figure 6. A coupled model

```
COUPLED MODEL Z
    EXTERNAL
        INPUT  IN;
            RANGE OF IN  = REAL (0.0 .. 10.0);
        OUTPUT OUT;
            RANGE OF OUT = REAL (40.0 .. 75.0);
    END EXTERNAL;

    COMPONENT MODELS M, N;
        (* Detailed specifications of the component models M, N
           would appear herebelow:                                  *)
        MODEL M
            ...

        END MODEL M;

        MODEL N
            ...

        END MODEL N;

    END COMPONENT MODELS;

    EQUIVALENCING
        INPUTS  Z.IN  = M.B;
        OUTPUTS Z.OUT = N.H;
    END EQUIVALENCING;

    COUPLING FOR Z
        M.A <--- N.H;
        M.B <--- Z.IN;    (* Z.IN IS AN EXTERNAL INPUT *)
        M.C <--- M.E;     (* FEED-BACK COUPLING        *)

        N.F <--- M.D;
        N.G <--- N.K;     (* FEED-BACK COUPLING        *)
    END COUPLING FOR Z;
END MODEL Z;
```

Figure 7. GEST representation of the coupled model given in Figure 6.

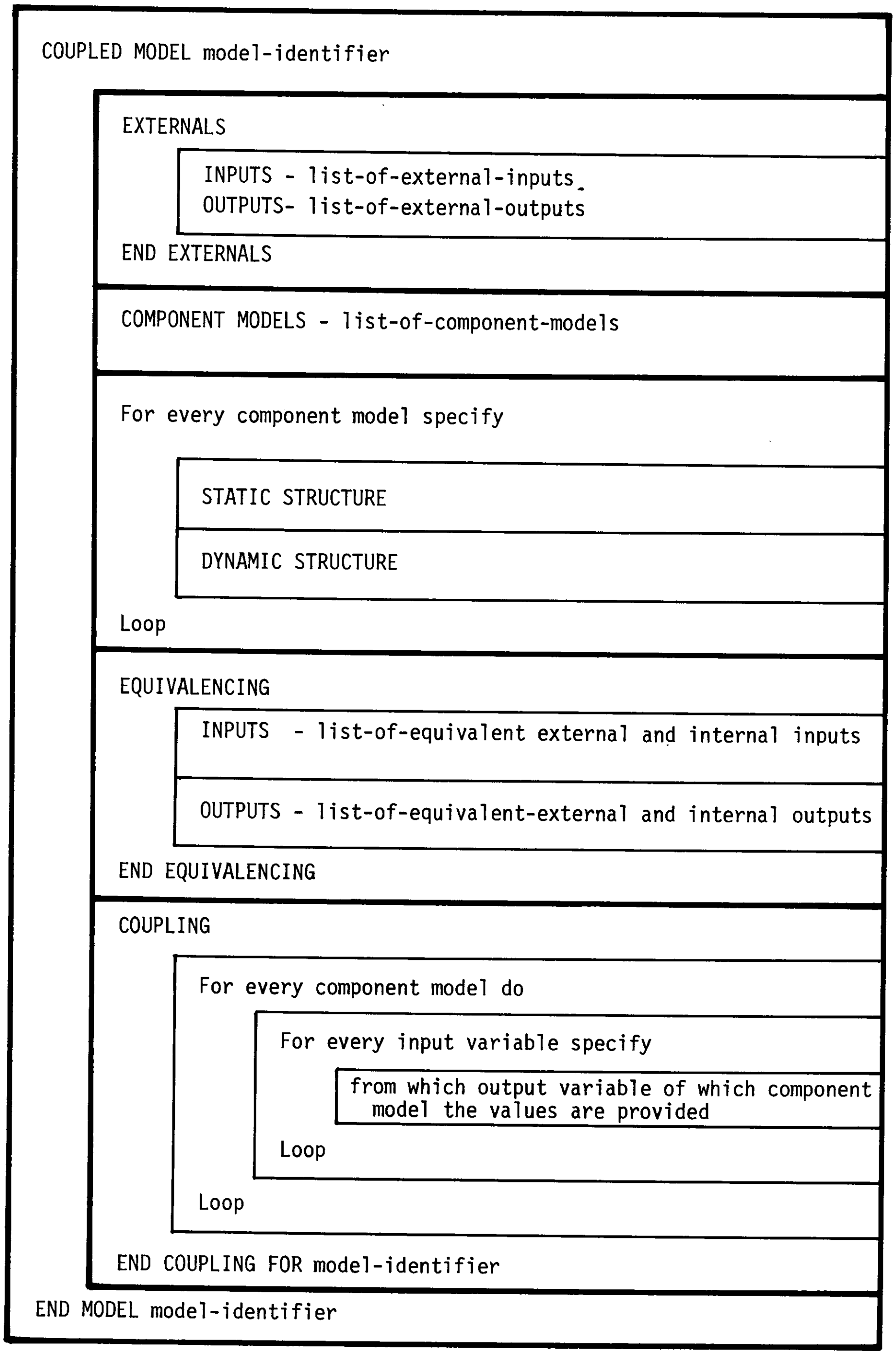


Figure 8. Layout of a coupled model specification in GEST

Some relevant BNF definitions follow:

```
coupled-model =
    "COUPLED MODEL" model-identifier
        [ external-variables ]

         "COMPONENT MODELS" list-of-component-models ";"
             { component-model }
         "END COMPONENT MODELS" ";"

        [ equivalencing-external-and-internal-variables ]

         coupling
    "END MODEL" model-identifier ";"

external-variables =
    "EXTERNAL"
         [ input-declaration  [{ range-declaration }] ]
         [ output-declaration [{ range-declaration }] ]
    "END EXTERNAL" ";" .
```

In the external variables section, the names of the external input and output variables of the coupled model are declared. For every input or output variable, range of the values may also be specified.

An implication of the ability to declare external input or output variables is the possibility to specify nested couplings where at least one of the component models is itself a coupled model.

Another implication is the ability to declare the ranges of the external input and output variables independent of the ranges of the corresponding variables of component models. The compatibility of external and corresponding internal variables can be checked algorithmically.

```
list-of-component-models =
  model-identifier [ ":" unsigned-integer "TO" unsigned-integer ]
 { "," model-identifier
                  [ ":" unsigned-integer "TO" unsigned-integer ]  } .
```

Example: COMPONENT MODELS M:1 TO 15, N, P:3 TO 7;

This statement causes the generation of 15 replicas of a model M. The generated copies are then named, by the system, M:1, M:2, ..., M:15. Similarly, five replicas of the component model P are generated and named P:3, P:4, ..., and P:7.

```
equivalencing-external-and-internal-variables =
     "EQUIVALENCING"
          equivalencing-inputs
          equivalencing-outputs
     "END EQUIVALENCING" ";" .
```

In equivalencing external and internal inputs, one has to consider that an input to the coupled model can be the input to one or several component models. However, every output of the coupled model is an output of one component model only.

```
equivalencing-inputs =
   "INPUTS"
        { model-identifier "." input-variable "="
          model-identifier [ ":" unsigned-integer ] "." input-variable
   [{ "," model-identifier [ ":" unsigned-integer ]
                       }       }   "." input-variable }] ";" }.

equivalencing-outputs =
   "OUTPUTS"
        { model-identifier "." output-variable "="
          model-identifier [ ":" unsigned-integer ]
                                     "." output-variable ";" } .


output-variable = scalar-variable | dimensioned-variable .
```

```
coupling =
     "COUPLING FOR" model-identifier
         { input-output-relationship }
     "END  COUPLING FOR" model-identifier ";" .

input-output-relationship =
    model-identifier [":" unsigned-integer] "." input-variable "<--"
  [ model-identifier [":" unsigned-integer] "." output-variable ] ";" .

input-variable = scalar-variable | dimensioned-variable .
```

Coupled Model Formalism of GEST as a Top-Down Model Conception and Stepwise Model Refinement Tool

"Coupled Model" formalism provided in GEST, facilitates top-down model conception and stepwise model refinement. Figures 9 a-f show steps of model conception and corresponding GEST modelling statements for the example model given in Figure 7.

In step 1, one specifies the name of the model, the input and output variables of the model, and their ranges of acceptable values (Figure 9c, 9d).

In step 2, the names of the component models are specified (Figure 9b).

In step 3, each component model is specified separately. The model can either be specified from scratch or can be fetched from a model base (Figure 9c).

In step 4, the equivalencing of external and internal variables are specified.

In this step, an input to the coupled model (i.e., an external input to the resultant model) provides values to an input of one or several component models.

Then, one specifies, for every output of the coupled model (i.e., for every **external output**), the names of the output variable and of the component model which provides the values (Figure 9e).

In step 5, the coupling (i.e., the input/output relationships of the component models) is specified as follows (Figure 9f):

```
For every component model do
    For every input variable do
        Specify input-output relationship
    Loop
Loop
```

Some Implications

In a **nested coupling** at least one component model is a coupled model. Since the resultant model has its input(s) and output(s) declared it can act as a component model in a nested coupling. The concept of nested coupling introduced in GEST in 1970 (Ören 1970) allows both top-down model refinement and bottom-up model synthesis.

Several copies of **similar models** can be created automatically. For example, suppose that a model called, say "M" has already been specified. If the user wants to create n (to be specific let n=15) replicas of M, in the list of component models all one has to specify is M:1 TO 15. The created n replicas may be identical or similar models depending whether or not they have identical parameter values or not. Furthermore similar models thus created (i.e., replicas of the same generic model) can have same or different specific frames (see section 4.2).

Computer-assistance is straightforward in both specifying and checking the consistencies of coupled models. In a computer-assisted modelling system even some of the checks need not be done manually, due to the guidance of the modelling system. In such a system, for every component system, every input variable is listed (by the system) after the name of the component model. Therefore it is even not possible to misspell the names of component models or the input variables. Furthermore if an input variable has been declared as an external input (input of the resultant system) then the system can finish the specification of input/output relationship. For example in the model given in Figure 8, the line

```
M.B <--- Z.IN;    (* EXTERNAL INPUT  *)
```

has to be generated fully by the modelling system, thus eliminating also the possibility of a wrong input/output connection. If the ranges of the values of the input and output variables are also specified, for every (input, output) pair specified in the coupling, the system can also check whether or not the range of the output is a subset of the range of the input and hence detect inconsistencies.

Documentation

Due to GEST's well-structured nature, computer generated structured flowcharts of GEST models can easily be obtained.. ORGEST (Organized Representation of GEST programs) is being implemented for this purpose (Oren et al. 1983). As an example, Figure 10 depicts a structured flowchart of the memoryless model of Figure 5.

Coupling specification can be very useful in the documentation of systems consisting of large number of interacting component systems. A documentation module would display each component model separately according to Figure 11 where external input(s), output(s), feedback and other input/output interface are easily displayed.

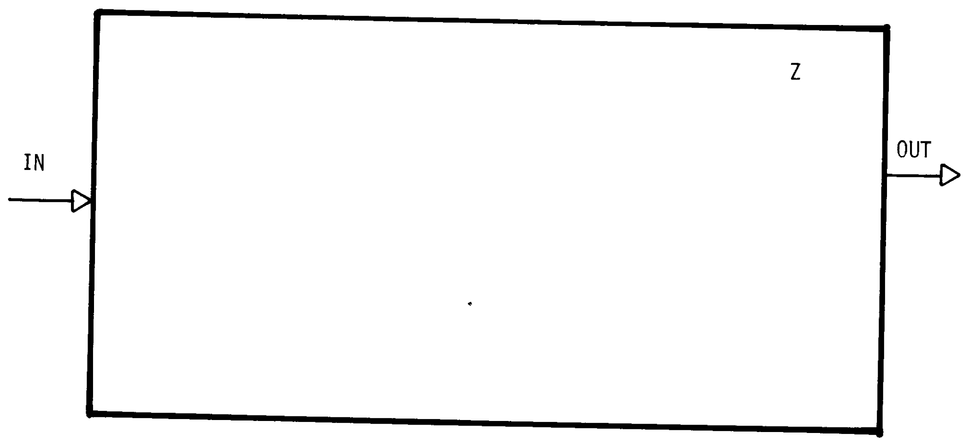
Documentation of models specified partially by part of their static structures (input and output variables only) can be very useful in several application areas such as documentation of organizations, large software, or hardware systems.

STEP 1 - Specify: 1) name of the model,

2) input and output variables, and

3) ranges of acceptable values of input and output variables

PICTORIAL REPRESENTATION:



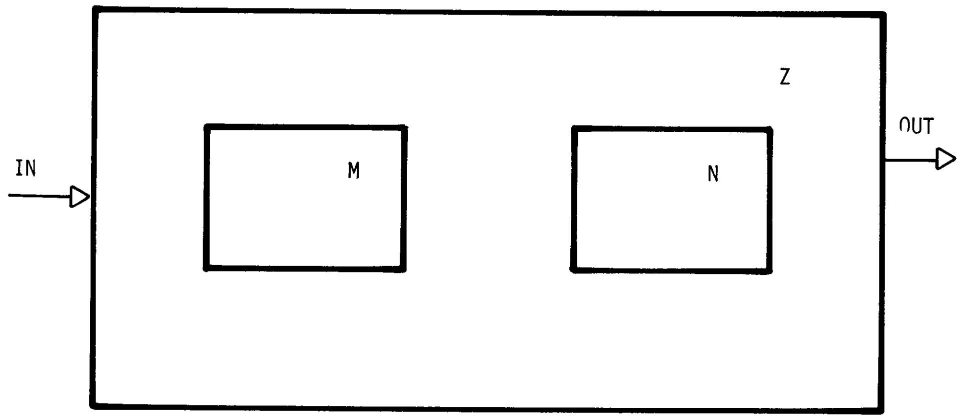
GEST MODELLING:

```
COUPLED MODEL Z

EXTERNALS

   INPUT IN;
      RANGE OF IN  = REAL(>= 0.00, <= 100.00);

   OUTPUT OUT;
      RANGE OF OUT = TEAL(>= 40.00, < 75.00);

END EXTERNALS;
```

Figure 9a. Step 1 in top-down model conception and step-wise model refinement in GEST

STEP 2 - Specify names of component models

PICTORIAL REPRESENTATION:



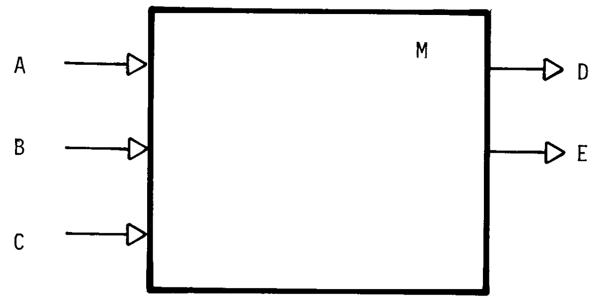
GEST MODELLING:

COMPONENT MODELS M, N;

Figure 9b. Step 2 in top-down model conception and step-wise model refinement in GEST

STEP 3 - Specify each component model separately

PICTORIAL REPRESENTATION:



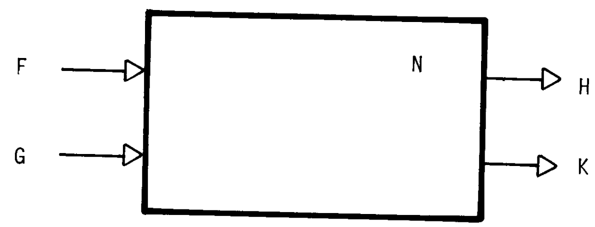
GEST MODELLING:

```
CONTINUOUS MODEL M;

    STATIC STRUCTURE

        INPUTS  A, B, C;
        STATE   ...
        OUTPUTS D, E;

          ...

    END STATIC STRUCTURE;

    DYNAMIC STRUCTURE

        DERIVATIVES

            ...

        END DERIVATIVES;

        OUTPUT FUNCTION
            ...
        END OUTPUT FUNCTION;

    END DYNAMIC STRUCTURE;

END MODEL M;
```

Figure 9c. Step 3 in top-down model conception and step-wise model refinement in GEST

STEP 3 - Specify each component model separately

PICTORIAL REPRESENTATION:



GEST MODELLING:

```
CONTINUOUS MODEL N;

    STATIC STRUCTURE

        INPUTS  F, G;
        STATES  ...
        OUTPUTS H, K;

          ...

    END STATIC STRUCTURE;

    DYNAMIC STRUCTURE

        DERIVATIVES

            ...

        END DERIVATIVES;

        OUTPUT FUNCTION
            ...

        END OUTPUT FUNCTION;

    END DYNAMIC STRUCTURE;

END MODEL N;
```

Figure 9d. Step 3 (for the second component model of Z)

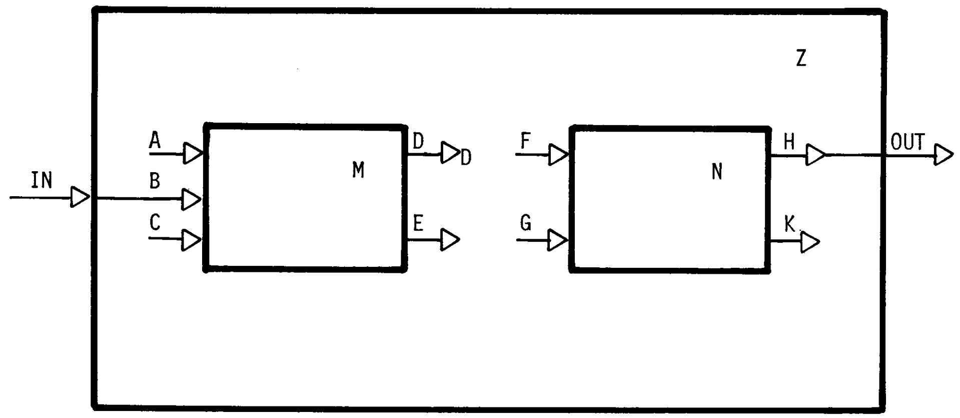
STEP 4 - Specify 1) for every external input

corresponding internal input(s)

2) for every external output

corresponding internal output

PICTORIAL REPRESENTATION:



GEST MODELLING:

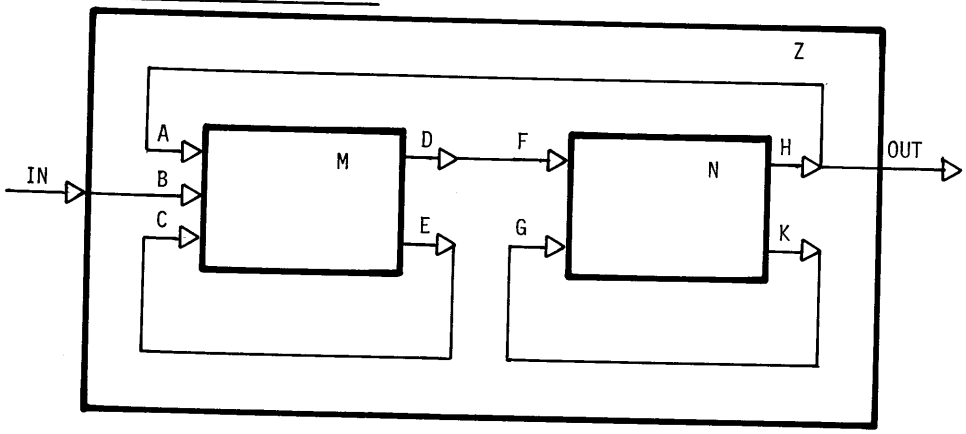
```
EQUIVALENCING
    INPUTS  Z.IN = M.B;
    OUTPUTS Z.OUT = N.H;
END EQUIVALENCING;
```

Figure 9e. Step 4 in top-down model conception and step-wise model refinement in GEST

STEP 5 - Specify coupling of component models, i.e.,

```
for every component model
          for every internal input
                    specify
                        from which output variable of which
                        component model the values are
                        provided
          loop
loop
```

PICTORIAL REPRESENTATION:



GEST MODELLING:

```
COUPLING FOR Z

    M.A  <--  N.H;
    M.B  <--  Z.IN;    (* EXTERNAL INPUT *)
    M.C  <--  M.E;

    N.F  <--  M.D;
    N.G  <--  N.K;

END COUPLING FOR Z;
```

Figure 9f. Step 5 in top-down model conception and step-wise model refinement in GEST

```
MEMORYLESS MODEL BIRTH_RATE
    STATIC STRUCTURE
        INPUTS
               S,    (* MATERIAL STANDARD OF LIVING  *)
               NE,   (* EFFECTIVE POLLUTION          *)
               P;    (* POPULATION                   *)
        OUTPUT BIRTH_RATE;
        PARAMETERS K20, K21, K22, K23, K24, K25;
    END STATIC STRUCTURE;
    DYNAMIC STRUCTURE
        B = K20 - K21*S - K22*NE - K23*P;
        IF B >= K24 AND B <= K25
            THEN  BIRTH_RATE = B;
            ELSE  IF B < K24
                      THEN  BIRTH_RATE = K24;
                      ELSE  BIRTH_RATE = K25;
                  END IF;
        END IF;
    END DYNAMIC STRUCTURE;
END MODEL BIRTH_RATE;
```

Figure 10. Structured documentation of the memoryless model given in Figure 5

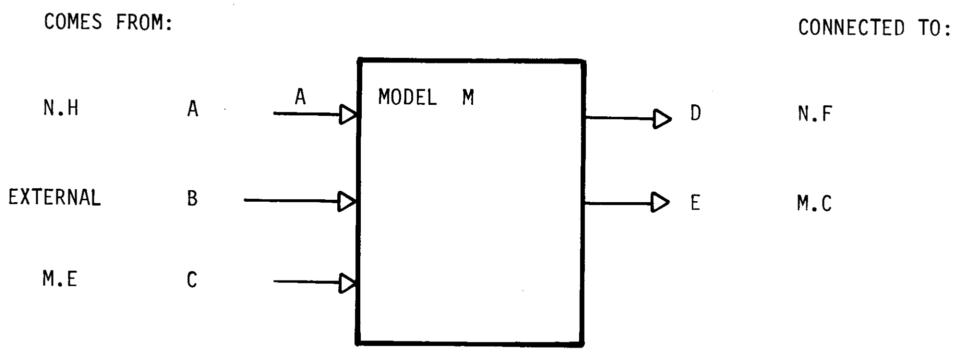


Figure 11. Documentation of the input/output relationships of one of the component models of a coupled model (which is represented in Figure 6)

3.5 Model Parameter Sets

In a parameter set basically two things are done: 1) The values of the model parameters are specified and 2) The tabular functions used in a model are defined point by point. There are four possibilies for the specification of model parameter sets, i.e., specification of single or multiple parameter set for a model consisting of one or several component models. Examples of model parameter set specifications are included in the system models given in section 5. It is also possible to specify discontinuous tabular functions. For every discontinuity point, two points have to be given with the same abscissa as follows:

(Xc,Yl) / (Xc,Yr)

where

- Xc is the common abscissa
- Yl is the left-hand value of the ordinate
- Yr is the right-hand value of the ordinate

In a computer-aided documentation system two- or three-dimensional tabular functions can also be displayed graphically for the user's convenience. Graphic display of the tabular functions may also help the user to detect some of the specification errors.

4. EXPERIMENTATION

4.1 Basics

The specification of the experiments, as shown in Figure 12, may comprise up to three sections which are:

(1) experimental frames
(2) model/frame pairs (or simulation runs)
(3) post study section

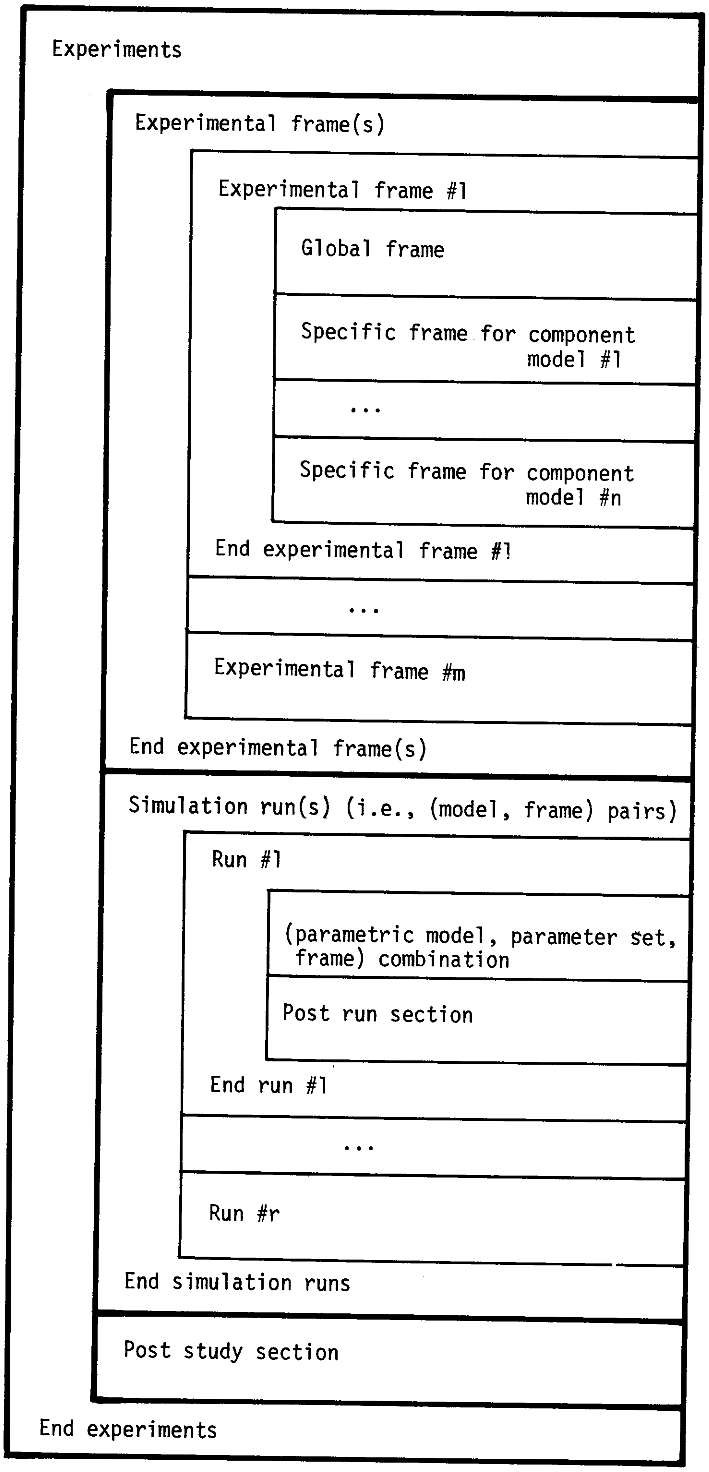


Figure 12. Elements of specifications of experiments in GEST

An experimental frame specifies the simulation experiments. It can be conceived of having two parts which are global frame and specific frame.

In the global **frame,** overall specifications such as time unit declaration, termination condition specification, etc, are given.

Specific frames given for every component model separately, include initialization of the state variables, data collecting requirements, and specification of communication intervals and input values.

A **model-frame pair** specifies the combination of the parametric model, the parameter set, and the experimental frame to be used in a simulation study. It also includes a post run section.

The **post run section** includes specification of the aditional computations to be performed after the simulation and outputting the results of a simulation run. The post study section resembles to a post-run section, with an additional level of generalization. In a post-study section, one can refer to data generated during different simulation runs. The output module consists of the specifications for displaying the results of the simulation study on different types of output units. Examples are included in the three system models given in section 5.

4.2 BNF Definitions

Some relevant definitions in BNF follow:

```
experiment =
    { experimental-frame }
    { model-frame-pair  } .
```

```
experimental-frame =
     single-experimental-frame
   | multiple-experimental-frame .


single-experimental-frame =
      "FRAME" identification "FOR" model-identifier

           global-frame


         { specific-frame }


      "END FRAME" identification ";" .



multiple-experimental-frame =
      "MULTIPLE FRAME" multiple-identification
                "FOR" model-identifier

          global-frame


         { specific-frame }


      "END MULTIPLE FRAME" multiple-identification ";" .


global-frame =
   "GLOBAL"
         time-unit-declaration
       | termination-condition
       | integration
       | activation-of-discrete-models
       | parameter-asignment
       | data-for-tabular-function
       | interpolation-type
       | communication-interval
       | renaming-declaration
   "END GLOBAL" ";" .
```

```
specific-frame =
   "MODEL" model-identifier
          [ ":" unsigned-integer "TO" unsigned-integer ]
        parameter-assignment
      | data-for-tabular-function
      | interpolation-type
      | communication-interval
      | renaming-declaration
      | initialization
      | input-scheduling
      | activation-of-discrete-model
      | output-sampling
      | data-collecting
   "END MODEL" model-identifier ";" .

model-frame-pair =
    single-run
  | multiple-run [ post-study-specification ] .

single-run =
   "RUN" identification
        "TO OBSERVE MODEL" model-identifier
      [ "WITH PARAMETER SET" identification ]
        "IN FRAME" identification
      [ "WITH POST RUN"
            { statement | output-module-reference }
        "END POST RUN" ";" ]
   "END RUN" identification ";" .
```

Examples:

```
      RUN 1 TO OBSERVE MODEL POPULATION_GROWTH
            WITH PARAMETER SET HIGH_BIRTH_RATE
            IN FRAME LOW_EMPLOYMENT;
      END RUN 1;

      RUN 7 TO OBSERVE MODEL EVAPORATION_FROM_SOIL
            WITH PARAMETER SET CLAY
            IN FRAME RAINY_SEASON ;
      END RUN 7;
```

5. EXAMPLES:

The following examples of complete GEST programs are provided. In all of them comments are kept at a minimum level, i.e., to provide a list of variables only.

Example 1 - DEER POPULATION

Study consists of:

One component model
One parameter set, and
One experimental frame

Example 2 - EVAPORATION FROM SOIL

Study consists of:

One component model
One parameter set, and
One experimental frame

Example 3 - MOTOR CONTROLLER

Study consists of:

A coupled model
One parameter set, and
One experimental frame

```
PROGRAM STUDY_OF_DEER_POPULATION

(* This model is adopted from:
   P.W. House and J. McLeod (1977).
   Large-scale models for policy evaluation.
   Wiley-Interscience, New York, pp. 148-153    *)


CONTINUOUS MODEL DEER_POPULATION

    STATIC STRUCTURE
        STATE  DP;              (* DEER POPULATION                  *)
        OUTPUT DP, PP, DKPP;
        AUXILIARY VARIABLES
               DNI,             (* DEER NET INCREASE                *)
               DPR,             (* DEER PREDATION RATE              *)
               DD;              (* DEER DENSITY                     *)
        PARAMETERS AREA,
               NIR;             (* NET INCREASE RATE                *)
        TABULAR FUNCTIONS
               PPT,             (* PREDATOR POPULATION TABLE        *)
               DKPPT;           (* DEER KILL PER PREDATOR TABLE     *)
        INTERPOLATIONS
               PP,              (* PREDATOR POPULATION              *)
               DKPP;            (* DEER KILL PER PREDATOR           *)
            PP = PPT(TIME);
            DKPP = DKPPT(DD);
        END INTERPOLATIONS;
    END STATIC STRUCTURE;

    DYNAMIC STRUCTURE
        DERIVATIVES
            DP' = DNI - DPR;
            DNI = NIR*DP;
            DPR = PP*DKPP;
            DD  = DP/AREA;
        END DERIVATIVES;
    END DYNAMIC STRUCTURE;
END MODEL DEER_POPULATION;
```

```
PARAMETER SET 1 FOR DEER_POPULATION
    AREA = 800000.0; NIR  = 0.2;
    TABULAR FUNCTION PPT WITH 2 POINTS
        (* GIVES PP   - PREDATOR POPULATION AS A FUNCTION OF TIME *)
        (1880., 300.)(1960., 300.)
    END TABULAR FUNCTION PPT;
    TABULAR FUNCTION DKPPT WITH 6 POINTS
        (* GIVES DKPP - DEER KILL PER PREDATOR
                        AS A FUNCTION OF DD - DEER DENSITY   *)
        (0.0,    0.0) (0.005,  3.0) (0.010, 13.0)
        (0.015, 32.0) (0.020, 51.0) (0.025, 56.0)
    END TABULAR FUNCTION DKPPT;
END PARAMETER SET 1;

FRAME 1  FOR DEER_POPULATION
    GLOBAL
        TIME UNIT IS YEAR
        SIMULATE UNTIL TIME = 1970, START TIME = 1880;
        INTEGRATE BY RUNGE KUTTA, REL_ERROR = 0.001;
    END GLOBAL;
    MODEL DEER_POPULATION
        INITIALIZE STATE DP = 4000.;
        SAVE DP, PP, DKPP AT EVERY YEAR ;
        INTERPOLATION LINEAR PP, DKPP;
    END MODEL DEER_POPULATION;
END FRAME 1;

RUN 1 TO OBSERVE MODEL DEER_POPULATION WITH PARAMETER SET 1 IN FRAME 1
    WITH POST RUN
        OUTPUT MODULE 1 ON PRINTER;
    END POST RUN;
END RUN 1;

OUTPUT MODULE 1
    PRINT 1 HEADING LINE;
    STUDY OF DEER POPULATION
    PLOT DP, PP, DKPP VERSUS TIME;
END OUTPUT MODULE 1;
END PROGRAM STUDY_OF_DEER_POPULATION;
```

```
PROGRAM STUDY_OF_EVAPORATION_FROM_SOIL
    (* This model is adopted from:
       C.F. Shaykewich and L. Stroosnijder (1977).
       The concept of matric flux potential applied to simulation
       of evaporation from soil. Neth. J. Agric. Sci., PP. 63-82.
       GEST 78 Version of this model appeared in Oren and Den Dulk 1978.
       (To ease the comparison with the original version,
       the units used in the original version are kept.
       However, for SI units, note that 1 mbar = 100 Pascal)

    LIST OF IDENTIFIERS:
    EA       - WATER VAPOR PRESSURE IN ATMOSPHERE   (MBAR) (MILLIBAR)
    EC       - WATER VAPOR PRESSURE AT SOIL SURFACE
                (CURRENT PRESSURE)                  (MBAR)
    ES       - WATER VAPOR PRESSURE AT SOIL SURFACE
                (AT SATURATION)                     (MBAR)
    F        - FACTOR ACCOUNTING FOR THE EFFECT OF
               WIND ON EVAPORATION                  (CM/DAY.MBAR)
    FK       - HYDRAULIC CONDUCTIVITY AS A FUNCTION
               WATER CONTENT
    FLW      - FLUX OF WATER                        (CM**3/(CM**2),DAY)
    FLW(1)   - EVAPORATION (I.E. FLOW FROM TOP COM-
               PARTMENT                             (CM**3/(CM**2),DAY)
    FPSI     - PSI AS A FUNCTION OF WATER CONTENT
    K(-)     - HYDRAULIC CONDUCTIVITY               (CM/DAY)
    MFLP(-)  - MATRIX FLUX POTENTIAL                (CM**3/DAY) * -1
    MFLPT    - MFLP AS A FUNCTION OF WATER CONTENT
    PSI(-)   - WATER POTENTIAL                      (CM H20) * 1
    RDF(-)   - RECIPROCAL OF THE DISTANCE BETWEEN
               CENTRES OF COMPARTMENTS              (1/CM)
    RTC(-)   - RECIPROCAL OF COMPARTMENT THICKNESS  (1/CM)
    TCM(-)   - COMPARTMENT THICKNESS                (CM)
    TEVAP    - ACCUMULATED EVAPORATION              (CM)
    W(-)     - WATER CONTENT (FRACTION)             (CM**3/CM**3)

      COMMENT ON THE PHYSICAL LAW FOR FLUX:
     ALL FLUXES ARE CALCULATED WITH DARCY'S LAW FOR
     VERTICAL FLOW OF WATER IN THE SOIL:
             FLUX = COND * (( -D_PSI/D_Z) -1)          *)
```

```
CONTINUOUS MODEL EVAPORATION_FROM_SOIL
    STATIC STUCTURE
        STATE
            TEVAP,              (* ACCUMULATED EVAPORATION        *)
            W(1..18);           (* CURRENT WATER CONTENT          *)
        OUTPUT
            TEVAP,              (* ACCUMULATED EVAPORATION        *)
            W(1..18),
            PSI(1..8),          (* WATER POTENTIAL                *)
            K(1..8),            (* HYDRAULIC CONDUCTIVITY         *)
            FLW(1);             (* FLUX OF WATER FROM
                                   TOP COMPARTMENT,I.E.
                                   EVAPORATION                    *)
        AUXILIARY VARIABLE  FLW(1..19), K(1..19),
                                 MFLP(1..18), PSI(1..18);
        PARAMETER
            TCM(1..18);         (* COMPARTMENT THICKNESS          *)
        AUXILIARY PARAMETERS
            RTC(1..18),         (* RECIPROCAL OF COMPARTMENT
                                   THICKNESS                      *)
            RDF(1..18);         (* RECIPROCAL OF THE DISTANCE
                                   BETWEEN CENTERS OF
                                   COMPARTMENTS                   *)
          FOR I=1 TO 18 DO
              RTC(I) = 1.0/TCM(I);
          LOOP

          RDF(1) = RTC(1);

          FOR I=2 TO 18 DO
              RDF(I) = 2.0/ (TCM (I-1) + TCM (I));
          LOOP
        END AUXILIARY PARAMETERS;

        TABULAR FUNCTIONS
            FK,     (* HYDRAULIC CONDUCTIVITY AS
                       A FUNCTION OF WATER CONTENT           *)
            FPSI,   (* PSI AS A FUNCTION OF WATER CONTENT    *)
            MFLPT;  (* MFLP AS A FUNCTION OFWATER CONTENT    *)
```

```
        INTERPOLATIONS
            K(I)    = FK(W(I));
            PSI(I) = FPSI(W(I));
            MFLP(I)= MFLPT(W(I));
        END INTERPOLATIONS;
    END STATIC STRUCTURE;

    DYNAMIC STRUCTURE
        DERIVATIVES
            TEVAP' = FLW(1);

            FOR I = 1 TO 18 DO
                W(I)' = (FLW(I+1) - FLW(I))*RTC(I);
            LOOP
               (* CALCULATION OF EVAPORATION
                  (I.E. FLOW FROM TOP COMPARTMENT = FLW(1))
                  EQUATION IS:
                  EVAPORATION = F*(EC-EA)/(ES-EA)
                  F         = Ø.8 CM/DAY (DEPENDS ON WINDSPEED)
                  ES        = 31.45 MBAR  (25C, AT SATURATION)
                  EA        = 7.Ø6 MBAR  (2ØC, 3Ø] RELATIVE HUMIDITY)
                  F/(ES-EA)= Ø.8/(31.45-7.Ø6) = Ø.Ø328
                  EC        = ES/EXP(PSI*7.127E-7)          *)

          FLW(1) = MAX ( Ø.Ø,
                        Ø.Ø328*(31.45/EXP(7.127E-7 * PSI(1)) - 7.Ø6)

               (* CALCULATION OF FLUX, USING WATER POTENTIAL
                  AND HYDRAULIC CONDUCTIVITY INDEPENDENTLY  *)

          FOR I = 2 TO 18 DO
              FLW(I) = (RDF(I) * (PSI(I-1) - PSI(I)) - 1.)
                                              * (K(I-1) + K(I))/2.Ø;
          LOOP

          FLW(19) = Ø.Ø;
       END DERIVATIVE;
   END DYNAMIC STRUCTURE;
END MODEL EVAPORATION_FROM_SOIL;
```

```
PARAMETER SET 1 FOR EVAPORATION_FROM_SOIL

    TCM(1..18) = 5*1.0, 5*1.5, 3*2.5, 4*5.0, 10.0;

    TABULAR FUNCTION FK          WITH 17 POINTS
       (* GIVES THE HYDRAULIC CONDUCTIVITY K(-)
       AS A FUNCTION OF WATER CONTENT W(-)*)
       (0.03    ,2.4E-10)(0.06   ,2.4E-9)(0.085  ,2.4E-8)(0.12   ,2.4E-7)
       (0.145   ,2.4E-6)(0.175   ,2.4E-5)(0.19   ,6.0E-5)(0.205  ,2.4E-4)
       (0.2175  ,6.0E-4)(0.235   ,2.4E-3)(0.25   ,6.5E-3)(0.27   ,2.4E-2)
       (0.2925  ,1.0E-1)(0.305   ,2.4E-1)(0.33   ,9.0E-1)(0.35   ,2.4    )
       (0.4     ,7.0    )
    END TABULAR FUNCTION FK;

    TABULAR FUNCTION FPSI   WITH 13 POINTS
       (* GIVES THE MATRIX WATER POTENTIAL (*-1) PSI(-)
       AS A FUNCTION OF WATER CONTENT W(-)*)
       (0.01  ,1.0E7 )(0.04    ,1.0E6)(0.095 ,1.0E5 )(0.135 ,2.7E4 )
       (o.165 ,1.0E4 )(0.2075  ,2.0E3 )(0.225,1.0E3 )(0.24   ,700.0 )
       (0.275 , 300.0)(0.32    ,100.0)(0.37   ,10.0  )(0.39   ,   1.0)
       (0.41  ,-600.0)
    END TABULAR FUNCTION FPSI;

    TABULAR FUNCTION MFLPT  WITH 30 POINTS
       (* GIVES THE MATRIC FLUX POTENTIAL (*_1) MFLP(-)
       AS A FUNCTION OF WATER CONTENT W(-)*)
       (0.0     ,43.0   )(0.144 ,42.688)(0.165   ,42.698)(0.1785,42.468)
       (0.189   ,42.348)(0.197 ,42.24  )(0.2075  ,42.14  )(0.216 ,41.88  )
       (0.2275  ,41.65  )(0.239 ,41.0   )(0.245   ,40.1   )(0.2525,39.68  )
       (0.261   ,38.88  )(0.268 ,37.38  )(0.274   ,35.94  )(0.277 ,34.58  )
       (0.281   ,33.68  )(0.284 ,32.64  )(0.2875  ,31.4   )(0.292 ,29.9   )
       (0.297   ,27.7   )(0.299 ,24.9   )(0.301   ,23.2   )(0.303 ,21.4   )
       (0.307   ,19.0   )(0.309 ,16.2   )(0.313   ,13.1   )(0.3165, 9.5   )
       (0.32    , 5.3   )(0.34  ,0.0    )
    END TABULAR FUNCTION MFLPT;

END PARAMETER SET 1;
```

```
FRAME 1 FOR EVAPORATING_FROM_SOIL

    GLOBAL
       SIMULATE UNTIL TIME = 5.0;
       INTEGRATE BY RUNGE_KUTTA;
    END GLOBAL;

    MODEL EVAPORATION FROM SOIL
       INITIALIZE STATES
          TEVAP    = 0.;
          W(1..18) = 18*0.2925;

       COMMUNICATE AT EVERY 0.25 TIME UNIT;

       SAVE W(1..16), PSI(1..8), K(1..8), FLW(1), TEVAP;

       INTERPOLATION LINEAR K, PSI, MFLP;

    END MODEL EVAPORATION_FROM_SOIL;

END FRAME 1;

RUN 1 TO OBSERVE MODEL EVAPORATION_FROM_SOIL
   WITH PARAMETER SET 1 IN FRAME 1
   WITH POST RUN
      OUTPUT MODULE 1 ON PRINTER
   END POST RUN
END RUN 1

OUTPUT MODULE 1
   PRINT FOR FIRST PAGE 1 HEADING LINE;
   STUDY OF EVAPORATION FROM SOIL - VERSION 1
   LIST TIME, W(1..16), PSI(1..8), K(1..8), FLW(1);
   PLOT TEVAP VERSUS TIME;
   PLOT FLW(1) VERSUS TIME WHILE (TIME <= 1.0);
   PLOT W(1) VERSUS TIME WHILE (TIME <= 0.5);
END OUTPUT MODULE 1;

END PROGRAM STUDY_OF_EVAPORATION_FROM_SOIL;
```

```
PROGRAM STUDY_OF_MOTOR_CONTROLLER

(* This model is adapted from:
   Elmquist, H. (1975). SIMNON User's Manual,
   Report 7502, Dept. of Automatic Control,
   Lund Institute of Technology, Sweden          *)

    COUPLED MODEL MOTOR_CONTROLLER
        EXTERNAL
             INPUT    YREF;
        END EXTERNAL;

        COMPONENT MODELS PID_CONTROLLER, MOTOR;

            CONTINUOUS MODEL PID_CONTROLLER;

                STATIC STRUCTURE
                   INPUTS YREF, Y;
                   STATES I, X;
                   OUTPUT U;
                   AUXILIARY VARIABLES E, P, D;
                   PARAMETERS G, GD, TD, TI;
                END STATIC STRUCTURE;

                DYNAMIC STRUCTURE
                   DERIVATIVES
                      I' = E/TI;
                      X' = -GD/TD*(X - Y);
                      E = YREF - Y;
                   END DERIVATIVES;

                   OUTPUT FUNCTION
                      U = P+I+D;
                      P = G*E;
                      D = -GD*(Y - X);
                   END OUTPUT FUNCTION;
                END DYNAMIC STRUCTURE;
            END MODEL PID_CONTROLLER;
```

```
      CONTINUOUS MODEL MOTOR
          STATIC STRUCTURE
             INPUT U;
             STATES TH, THDOT;
             OUTPUT Y;
             AUXILIARY VARIABLES ME, I;
             PARAMETERS KM, R, J, CT;
          END STATIC STRUCTURE;

          DYNAMIC STRUCTURE
             DERIVATIVES
                TH' = THDOT;
                THDOT' = ME/J;
                ME     = KM*I;
                I      = (U-KM*THDOT)/R;
             END DERIVATIVES;

             OUTPUT FUNCTION
                Y = CT*TH;
             END OUTPUT FUNCTION;
          END DYNAMIC STRUCTURE;
      END MODEL MOTOR;

  END COMPONENT MODELS;

  EQUIVALENCING
     INPUTS
        MOTOR_CONTROLLER.YREF = PID_CONTROLLER.Y_REF;
  END EQUIVALENCING;

  COUPLING FOR MOTOR_CONTROLLER
      PID_CONTROLLER.YREF <--- MOTOR_CONTROLLER.YREF;
                                         (* EXTERNAL INPUT *)
      PID_CONTROLLER.Y    <--- MOTOR.Y;

      MOTOR.U             <--- PID_CONTROLLER.U;
  END COUPLING FOR MOTOR_CONTROLLER;

END MODEL MOTOR_CONTROLLER ;
```

```
PARAMETER SET 1 FOR MOTOR_CONTROLLER

    MODEL PID_CONTROLLER
       G   = 1.0;
       GD  = 1.0;
       TD  = 1.0;
       TI  = 1.E10;
    END MODEL PID_CONTROLLER;

    MODEL MOTOR
       KM  = 6.2 E-3;
       R   = 5.3;
       J   = 7.5 E-7;
       CT  = 0.033;
    END MODEL MOTOR;

END PARAMETER SET 1 ;

FRAME 1 FOR MOTOR_CONTROLLER
   GLOBAL
      TIME UNIT IS SECOND;
      SIMULATE UNTIL TIME = 5.0;
      INTEGRATE BY RUNGE KUTTA, REL_ERROR = 0.0001;
      COMMUNICATE AT EVERY 0.01 SECOND;
   END GLOBAL;

   MODEL PID_CONTROLLER
      INITIALIZE STATES TO ZERO;
      SAVE U;
   END MODEL PID_CONTROLLER;

   MODEL MOTOR
      INITIALIZE STATES
         TH    =  0.0;
         YHDOT=  0.0;
      SAVE Y;
   END MODEL MOTOR;

END FRAME 1;
```

```
RUN 1 TO OBSERVE MODEL MOTOR_CONTROLLER
    WITH PARAMETER SET 1 IN FRAME 1;

    WITH POST RUN
         OUTPUT MODULE 1 ON PRINTER;
    END POST RUN;
END RUN 1;

OUTPUT MODULE 1

   PRINT FOR FIRST PAGE 1 HEADING LINE;
      STUDY OF MOTOR_CONTROLLER
   PLOT AND LIST U, V VERSUS TIME;

END OUTPUT MODULE 1;

END PROGRAM STUDY_OF_MOTOR_CONTROLLER;
```


Aytaç, K., Ören
MAGEST: A Kno

Bleha, L.J. (1977
Combined Digi
versity of Ot

Dogbey, F. (1984
continous-Cha
of Simulation
Ottawa, Ont.,

Elzas, M.S. (1979
odology in Sy
Elzas, G.J.
57-91.

Kettenis, D.L. (1
ulation of Sy
Congress, Sor
pp. 291-298.

Ören, T.I. (1971a
bined Digital
of Arizona, T

Ören, T.I. (1971b
Large Scale S
tion Internat
lation of Co
B-11/4.

LIST OF REFERENCES
(And BIBLIOGRAPHY OF GEST)

Aytaç, K., Ören, T.I. (1984 - In Preparation). **Architecture of MAGEST: A Knowledge-Based Modelling and Simulation System.**

Bleha, L.J. (1977). **An Implementation of the Discrete Portion of the Combined Digital Simulation Language GEST.** Master's Thesis. University of Ottawa, Ottawa, Ont., Canada, 155 p.

Dogbey, F. (1984 - In Progress). **Simulation with Continuous or Discontinous-Change Models: A Software System to Enhance Reliability of Simulation Studies.** Master's Thesis. University of Ottawa, Ottawa, Ont., Canada.

Elzas, M.S. (1979). **What is needed for Robust Simulation?** In: Methodology in Systems Modelling and Simulation, B.P Zeigler, M.S. Elzas, G.J. Klir, T.I. Ören (Eds). North-Holland, Amsterdam, pp. 57-91.

Kettenis, D.L. (1979). **Combined Simulation with SIMULA '67.** In: Simulation of Systems, L. Dekker et al. (Eds), Preprints of IMACS Congress, Sorrento, Italy, Sept.24-28, North-Holland, Amsterdam, pp. 291-298.

Ören, T.I. (1971a). **GEST: General System Theory implementor - A Combined Digital Simulation Language.** Ph.D. Dissertation, University of Arizona, Tucson, Arizona, USA.

Ören, T.I. (1971b). **GEST: A Combined Digital Simulation Language for Large Scale Systems.** Proceedings of the Tokyo 1971 AICA (Association Internationale pour le calcul Analogique) Symposium on Simulation of Complex Systems, Tokyo, Japan, September 3-7, pp. B-11/4.

Ören, T.I. (1972a). **Systems Approach in Ecological System Simulation.** Proceedings of the Fourth Annual International Symposium of the American Society of Cybernetics, Washington, DC, 1970, Oct. 8-9, Spartan Books, New York, pp. 303-309.

Ören, T.I. (1972b). **Implementation of Synchronous Variables and Another Parallel Computation Feature of GEST.** Proceedings of the 1972 Summer Simulation Conference, San Diego, CA., June 14-16,pp. 17-23

Ören, T.I. (1973). **Application of System Theoretic Concepts to the Simulation of Large Scale Adaptive Systems.** Proceedings of the sixth Hawaii International Conference on Systems Sciences, Honolulu, Hawaii, January 9-11.

Ören, T.I. (1974a). **Simulation of Large Scale Adaptive Multicriteria Decision Making Systems by GEST.** Proceedings of the Seventh Hawaii International Conference on Systems Sciences, Honolulu, Hawaii, January 8-10.

Ören, T.I. (1974b). **Deductive General Systems Theories and Simulation of Large Scale Systems.** Proceedings of the 1974 Summer Computer Simulation Conference, Houston, TX, July 9-11, pp. 13-16 .

Ören, T.I. (1975). **Simulation of Time-Varying Systems.** Proceedings of the International Congress of Cybernetics and Systems, J. Rose (Ed.), Oxford, England, August 28-Sept. 1, 1972. Gordon & Breach Science Publishers ltd., England, 1975, pp. 1229-1238.

Ören, T.I. (1978). **Rationale for Large Scale System Simulation Software Based on Cybernetics and System Theories.** In: Cybernetics and Modelling and Simulation of Large Scale Systems, T.I. Oren (Ed.) International Association For Cybernetics, Namur, Belgium. pp. 151-179.

Ören, T.I. (1979). **Concepts for Advanced Computer-Assisted Modelling.** In: Methodology in Systems Modelling and Simulation, B.P. Zeigler, M.S. Elzas, G.J. Klir, T.I. Ören (Eds.), North-Holland, pp. 29-55.

Ören, T.I. (1981). **User's Manual of GEST 81.** Technical Report TR81.13, Computer Science Dept., University of Ottawa, Ottawa, Ontario, Canada.

Ören, T.I. (1982). **Model Reliability in Interactive Modelling in GEST 81.** In: Proceedings of 10th IMACS World Congress on System Simulation and Scientific Computation, August 8-13, 1982, Montreal, PQ, Canada. Vol. 4, pp. 201-203.

Ören, T.I. (1984a). **Model-Based Activities: A Paradigm Shift.** In: Simulation and Model-Based Methodologies: An Integrative View. T.I. Oren, B.P. Zeigler, M.S. Elzas (Eds). Springer-Verlag, Heidelberg, W. Germany (Chapter 1 of this Book).

Oren, T.I. (1984b). **GEST 81: A Model and Simulation Specification Language - 1984 Update.** Technical Report TR-84-08, Computer Science Dept., University of Ottawa, Ottawa, Ontario, Canada.

Ören, T.I., Aytac, K. (1984 - In Preparation). **Model Reliability in MAGEST: A Knowledge-Based Modelling and Simulation System.**

Ören, T.I., N. Hokayem, K. Saleh, K. Maksoud (1983). **ORGEST - Organized Representation of GEST Programs.** Technical Report TR-83-12, Computer Science Department, University of Ottawa, Ottawa, Ontario, Canada.

Ören, T.I., B. Collie (1980). **Design of SEMA: A Software System for Computer-Aided Modelling and Simulation of SEquential MAchines.** In: Proceedings of 1980 Winter Simulation Conference, T.I. Oren, C.M. Shub, P.F. Roth (Eds). IEEE, New York, pp. 113-123.

Ören, T.I., J. den Dulk (1978). **Ecological Models Expressed in GEST 78.** Technical Report prepared for the Department of Theoretical Plant Ecology, Dutch Agricultural University, Wageningen, the Netherlands.

Ören, T.I., N. Hokayem, K. Saleh, K. Maksoud (1983). **ORGEST - Organized Representation of GEST Programs.** Technical Report, Computer Science Department, University of Ottawa, Ottawa, Ontario, Canada.

Ören, T.I., B.P. Zeigler (1979). **Concepts for Advanced Simulation Methodologies.** Simulation, Vol. 32, No. 3, pp. 69-82.

Subrahmanian, E., R.L. Cannon (1981). **A Generator Program for Models of Discrete-Event Systems.** Simulation, Vol. 36. No. 3, pp. 93-101.

Wymore, A.W. (1967). **A Mathematical Theory of Systems Engineering: The Elements.** John Wiley, New York, 353 p.

Wymore, A.W. (1976). **Systems Engineering Methodology for Interdisciplinary Teams.** John Wiley, New York, 431 p.

Zeigler B.P. (1976). **Theory of Modelling and Simulation.** Wiley-Interscience, New York.

Zeigler, B.P. (1978). **Structuring the Organization of Partial Models.** In: Cybernetics and Modelling and Simulation of Large-Scale Systems, T.I. Oren (Ed.), International Association for Cybernetics, Namur, Belgium, pp. 127-139.

APPENDIX: THE METALANGUAGE USED TO DEFINE GEST 81

- Terminal symbols of the GEST 81 language are enclosed within quotation marks and are written in UPPER CASE CHARACTERS.

- Non-terminal symbols of the GEST 81 language are written in lower case characters.

- When a non-terminal symbol requires more than one word, hyphen "-" is used between-the-words.

- A blank separates two syntactic units.

- A period "." indicates end of a rule.

- Left and right sides of any rule of the grammar are separated by the equivalence symbol "=" which can be read "is" or "can be formed from."

- Exclusive or is represented by the symbol "|" which can be read "or"

- Parentheses "( )" indicate grouping without repetition.

- Square brackets "[ ]" indicate option, that is, zero or one occurence.

- Curly brackets "{ }" indicate repetition, that is, one or more occurences.

- Zero or more occurences of a syntatic unit is represented by a combination of square and curly brackets, i.e., by "[{ }]".

TUNCER ÖREN

Simulation and Model-Based Methodologies: An Integrative View

Edited by

Tuncer I. Ören

Computer Science Department, University of Ottawa
Ottawa, Ontario, Canada

Bernard P. Zeigler

Computer Science Department, Wayne State University
Detroit, Illinois, USA

Maurice S. Elzas

Computer Science Department, Dutch Agricultural University
Wageningen, The Netherlands

Springer-Verlag Berlin Heidelberg New York Tokyo 1984

Published in cooperation with NATO Scientific Affairs Division

Proceedings of the Nato Advanced Study Institute on Simulation and Model-Based Methodologies: An Integrative View held at Ottawa, Ontario/Canada, July 26–August 6, 1982

ISBN 3-540-12884-0 Springer-Verlag Berlin Heidelberg New York Tokyo
ISBN 0-387-12884-0 Springer-Verlag New York Heidelberg Berlin Tokyo

Printed in Germany

Printing: Beltz Offsetdruck, Hemsbach; Bookbinding: J. Schäffer OHG, Grünstadt
2145/3140-543210

TABLE OF CONTENTS