# CANADIAN THESES ON MICROFICHE

# THÈSES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

## NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

## AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

NL 339 (r. 86/09)

# GEST TRANSLATOR WITHIN THE KNOWLEDGE-BASED MODELLING SYSTEM MAGEST

by

FELIX KWAME ATSU DOGBEY

A thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
Masters
in
Computer Science

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

# PART I

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

## ACKNOWLEDGEMENTS

I would like to thank Dr. Tuncer I. Oren, my thesis supervisor, for suggesting the thesis topic and seeing through its completion. Many thanks are also due Dr. K. Aytac and Ji-Hu Ma, Visiting Research Scholars for their tremendous help and co-operation in the development of some programs.

My warm gratitudes go to my wife, Vida, for her superb moral and spiritual support without which life would have been different.

Finally, I would like to say thank you to the A.U.C.C. for the much needed financial support.

## ABSTRACT

The increasing importance of simulation software development methodologies for large scale models is an index of the increasing complexity of business and scientific problems and the constraining force on simulationists' ability to construct and experiment with large scale multi-faceted simulation models. When simulating complex systems it is often advantageous, if not necessary, to decompose the simulated system. The subsystems are specified separately together with the description of their interconnection. Presently used simulation languages do not cope with this problem. In this thesis, a set of software tools (GEST translator, GEST template generator and GEST interactive model specification tool) has been developed to be used in conjunction with the GEST language (Oren 1982b). Such software tools are intended to be helpful in such procedures as specification of a GEST model; its certification for consistency, completeness, integrity and comprehensibility; and transformation of the model into machine readable form.

CONTENTS

PART I

# Chapter 1

# INTRODUCTION

It is commonly agreed that software developments tend to be high risk activities; simulation is recognized as being even more involved. Great emphasis is being placed to develop methodologies which lower the risk of software development.

Since a major portion of simulation activity is software oriented, it is natural to look to these modern software methodologies for solutions applicable to the modelling and simulation community. The lack of a formal methodology for modelling and simulation is receiving considerable attention in the simulation literature. The problems are numerous and complex, the proposed solutions varied.

## 1.1 OBJECTIVE AND ORGANIZATION OF THESIS

The main objective of this thesis project is to develop the necessary software tools for the GEST modelling language. This has been motivated by the need to have tools for modelling and simulation support as a result of the complexity of systems being simulated, on one hand, and the tremen-

dous progress in digital technology on the other (Oren 1982a).

Chapter 1 introduces the role of simulation, and the major approaches in implementing simulation studies, their problems and advantages. Chapter 2 presents current problems in modelling. The World view of GEST modelling is discussed in Chapter 3. Chapter 4 briefly describes the GEST certification program which certifies the GEST model before it is translated into the target code. A description of the software tools developed in this thesis project is presented in Chapter 5. In this chapter we also examine aspects of model reliability that were of concern during the software development. Examples of the usage of the translator and a demonstration run are given in Chapter 6. Suggestions of possible future work are given in Chapter 7.

## 1.2 THE ROLE OF SIMULATION

Simulation of business and economic systems has evolved as one of the most interesting and potentially powerful tools available for analysing business and economic problems. Through simulation techniques the business analyst, operations researcher, or economist has the means for observation and experimentation which have long been the essence of the approach of the physical scientist. Building and

running a simulation model permits observation of the dynamic behaviour of a system under controlled conditions, and experiments may be run to test hypothesis about the system under study. In other words, simulation provides a laboratory for analysis of problems that often cannot be solved by other means. Early uses of simulation techniques usually involved experimentation with physical models representing the phenomenon under investigation. As such, simulation was widely employed in engineering and scientific studies. Since administrative and economic processes are not easily represented by physical models, simulation by this method has been little used by social scientists and managers. However, simulation by means of digital computers has found wide acceptance in both engineering and scientific work and in the analysis of administrative and economic problems.

Experimentation by means of computer simulation can overcome some of the restrictions that exist when other forms of analysis are used. It opens up the possibility of dealing with the dynamics of processes too complex to be represented by more rigid mathematical models such as linear programming, and calculus maximization and minimization models.

## 1.3 CONTINUOUS VS DISCRETE-CHANGE MODELS

For a long time the nature of the models used in simulation divided simulationists into "discrete-" and "continuous-" model oriented groups (Oren 1982a). As the early terminology reflects it, they even viewed the World as divided into continuous and discrete parts. The terms "continuous system simulation" and "discrete system simulation" implicitly assume that the system under investigation is continuous or is discrete, respectively. Discrete-event simulation concerns the modeling of a system as it evolves over time by a representation in which the state variables change only at a countable number of points in time. These points in time are the ones at which an event occurs, where an event is defined to be an instantaneous occurrence which may change the state of a system.

Continuous simulation concerns the modelling over time of a system using a representation in which the state variables change continuously with respect to time. Typically, continuous simulation models involve one or more differential equations which give relationships for the rates of change of the state variables with respect to time. Greenspan (1973, 1980), among others, showed that systems traditionally modelled using calculus, therefore labelled as "continuous systems" could well be modelled using arithmetic and therefore be represented by discrete models. Since many

real-world systems are neither completely discrete nor completely continuous, the need occasionally arises to construct a model with aspects of both discrete-event and continuous simulation. Such a simulation is referred to as combined discrete-continuous simulation.

## 1.4 MAJOR APPROACHES IN IMPLEMENTING SIMULATION STUDIES

This section examines the major approaches in implementing simulation studies and analysing the behaviour of large-scale systems.

One approach is making use of already an existing program. A large number of special purpose programs have been written to simulate specific problems or types of systems. When a suitable special purpose program is available, the use of such a program is the quickest and cheapest way to implement a simulation study on the computer. There are problems associated with this approach. One problem is that existing programs usually do not have quite the capabilities that the user desires. Second ly, obtaining program descriptions, detailed documentation, and tapes are not easy. However it has an advantage that the use of an already existing program is by far the preferable way to carry out a simulation study (provided that the program does what it is supposed to do and no unanticipated bugs turn up.

When previously written programs with proper characteristics are not available, a simulation model must be programmed either in a general purpose language such as FORTRAN, COBOL, Pascal or PL/1 or in one of the simulation languages. In general, the simulation languages reduce programming effort by providing routines to perform certain operations peculiar to simulation which would otherwise have to be programmed in detail. One such capability provided by simulation languages is a mechanism for advancing the model ahead in time.

Despite the apparent advantages of simulation languages over general purpose languages, a simulation language is not necessarily the best programming vehicle. One must consider availability of simulation languages for the computing equipment to be used, the cost of learning a new language which may involve considerable investment of time by the problem solver, and whether the distinctive capabilities and structure of available languages are appropriate for the problem.

# Chapter 2

# CURRENT PROBLEMS IN MODELLING

A review of substantial current problems in modeling is presented in this chapter. The problems recognized are believed to exist in discrete event,continuous or combined simulation, mathematical programming, econometric, and other types of modeling (Balci 1983). Some of the problems are high cost of model development, lack of full cycle support, inadequate documentation, wide communication gap, constraints imposed by existing simulation programming languages and lack of user participation.

## 2.1 HIGH COST OF MODEL DEVELOPMENT

As reported by Roth,Gass, and Lemoine (1978) "The U.S. Government is the largest sponsor and consumer of models in the world. Estimates have indicated that over one-half billion dollars are being spent annually on developing, using, and maintaining mathematical, simulation, and econometric models in the decision-and-policy-making functions of the Federal Government".

A report to the U.S. Congress prepared by the General Accounting Office (GAO) (1976) said in part:

"GAO identified 519 federally funded models developed or used in the Pacific Northwest area of the United States. Development of these models cost about $39 million. Fifty-seven of these models were selected for detailed review, each costing over $100,000 to develop. They represent 55 percent of the $39 million of development costs in the models. Although successfully developed models can be of assistance in the management of Federal programs, GAO found that many model development efforts experienced large cost overruns, prolonged delays in completion, and total user dissatisfaction with the information obtained from the model."

An obvious important component of software development is computer programming. The cost of programming is becoming more dominant and apparent as hardware costs continue to decline. As indicated by Wasserman and Gutz (1982), "There is already a serious shortage of skilled programmers and the cost of such a person is expected to surpass $100,000 a year (salary, benefits, overhead) by the mid-1980's." This shortage is even more serious in the area of model development because programming is only one of several costly component activities.

## 2.2 LACK OF FULL LIFE CYCLE SUPPORT

In spite of the available simulation programming languages, model development is still labour intensive and error prone. The current simulation programming languages are supportive of only the programming process. Rarely do they even claim to provide effective tools for programmed model verification. At this time, automated support of model development throughout its entire life cycle (see Fig 1) is nonexistent. The life cycle of model development could conveniently be divided into six phases (Nance 1981) as depicted in Figure 1. The six phases are shown by oval symbols. The dashed arrows describe the processes which relate the phases to each other. The solid arrows refer to the procedures which evaluate these processes. The processes of model development should in no way be interpreted as sequential. Model development is iterative in nature and there is a back and forth movement between the phases during the development. Modelling starts with the given definition of the system under study and explicitly stated study objectives. System definition contains the formulated problem, system characteristics, and system boundary.

Model Formulation is the process by which the conceptual model is envisioned to represent the system under study. The Conceptual model (Nance 1981) is "that model which exists in the mind of the modeler. The form of the conceptual model is

influenced by the system, the perceptions of the system held by the modeler (which are affected by the modeler's background and experience and those external factors affecting the particular modeling task), and the objectives of the study."

Representation is the process of translating the conceptual model into a communicative model. Communicative model is defined by Nance as "a model representation which can be communicated to other humans, can be judged or compared against the system and the study objectives by more than one human. Several communicative models could be constructed during a study, each derived from a preceding communicative model (following the first) or different conceptual models". Entity cycle diagrams, flow charts, pseudocodes, flow diagrams, block and logic diagrams, or activity charts are examples of communicative models. Programming is the process of translating the communicative model into a programmed model which admits execution by a computer to produce results.

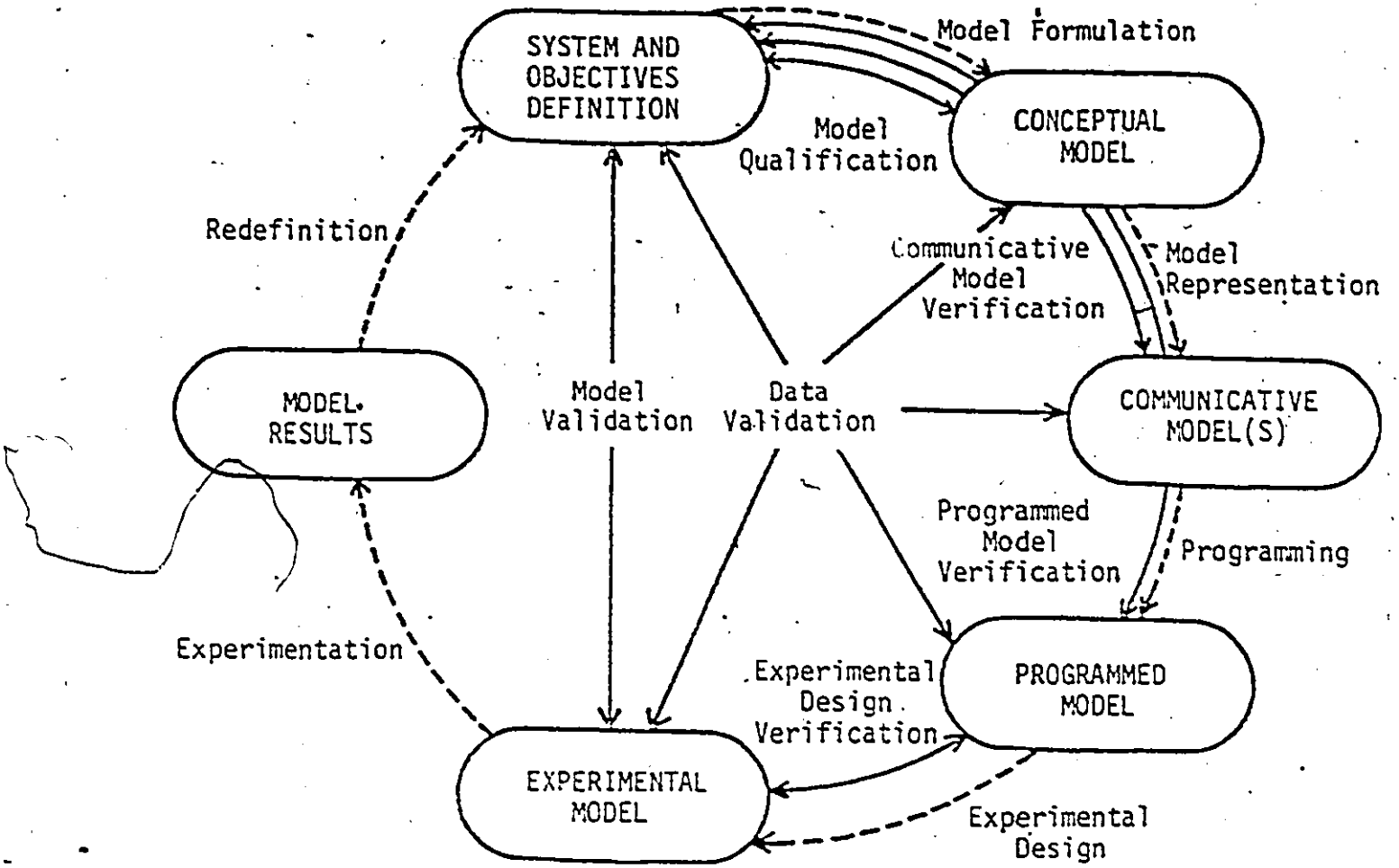



FIGURE 1. The Life Cycle of Model Development.

## 2.3 CONSTRAINTS IMPOSED BY SIMULATION PROGRAMMING LANGUAGES

Model formulation and representation are usually done under the constraints imposed by the simulation programming languages to be used in the programming process. This may induce substantial errors within the model representation right at the beginning of the model development life cycle. These errors are either caught in much later phases resulting in a higher cost of correction or never detected resulting in the error of accepting the results of an invalid model at the model user's risk. If the modeler is constrained by the world view of a simulation programming language, the conceptual and communicative models may be invalidated due to the incompatibility of the world view for the system under study with that of the simulation programming languages. This invalidity may be propagated to later phases in the development and may even cause the error of accepting the results of an invalid model.

## 2.4 REDEFINITION DOES NOT USUALLY FOLLOW THE ENTIRE LIFE CYCLE

More often the experimental model (see Fig 1) is redefined which is the process of updating the model so that it represents the current form of the system, or altering it for obtaining another set of results or changing it for the purpose of maintenance or modifying it for other use(s). The

changes required are generally made on the programmed or experimental model skipping the formulation and representation processes. This practice may induce substantial errors especially for large scale complex models. The process of redefinition should follow the entire life cycle starting with the definition of the system and study objectives and culminating with the model results.

## 2.5 INADEQUATE DOCUMENTATION

In his article, Gass (1983) indicates that "we do not know of any model assessment or modeling project review that indicated satisfaction with the available documentation." He notes that serious problems exist regarding the production and availability of model documentation.

Most models evolve over an extended period of time. The model is redefined repeatedly to reflect the new and increased understanding of the system, changing objectives, and the availability of new data. This evolutionary change, however , causes the documentation often to become obsolete, incomplete, or inadequate shortly after it is written (Annino and Russell 1979). The longer the model development the more the documentation deteriorates under the current practice.

## 2.6 POOR COMMUNICATION

A modeling project typically involves people with different backgrounds and areas of expertise. Communication problems arise among these people mostly due to the lack of (1) a conceptual framework, (2) a uniform terminology, (3) effective communication tools, and (4) good documentation.

At the present time, the literature on modeling does not display a consistent terminology. Many terms (i.e. assessment, evaluation, verification, validation, quality assurance, credibility, documentation, portability, certification) are interpreted differently depending upon the background of the modeller or the specific area of application.

## 2.7 INABILITY TO ASSESS MODEL QUALITY

The inability to assess the quality of a model adequately raises the probability of committing the error of accepting the results of an invalid model. The consequences of this error are crucial especially when vital decisions are made on the basis of model results.

## 2.8 INADEQUATE USER PARTICIPATION

As Annino and Russell (1979) have pointed out "All too often, model developers simply go off by themselves for a year and then proudly drop the 'completed,' never to be used model on the sponsors desk." Insufficient user involvement causes the model to be unresponsive to user needs resulting in user dissatisfaction with the information obtained from the model (U.S. GAO 1976).

## 2.9 INEFFECTIVE UTILIZATION OF EARLIER MODELING PROJECTS

Earlier modeling projects are not studied in detail and are not fully utilized. An already existing usable model (part) is sometimes rebuilt from scratch duplicating the development effort and unnecessarily increasing the cost and time of development. Even if existing models or model parts are not usable in the new project, a modeller (especially an inexperienced one) can study them to learn from past experience. Such a study may be extremely beneficial especially for the management, planning, and resource allocation for the new project.

There exists other issues which affect the success of a modeling project. Most important of all are : (1) failure to define a set of achievable objectives, (2) insufficient problem formulation, (3) inadequate user participation in defining the problem, (4) failure to identify the best solu-

tion technique, and (5) ineffective presentation of model results. Nance and Balci (1984) emphasised that in addressing these problems in a model management system, the system should be concerned with the problem definition phases as well as the model development and decision support phases by stating the objectives and requirements to address these problems.

# Chapter 3

## GEST MODELLING

GEST is a modelling and simulation language based on general system theoretic concepts. It was conceived and developed by Oren (Oren 1981). This language departs radically from other simulation languages in a way that specifications of the model and the experiment are totally separated in GEST.

Other GEST-like languages also exist. SEMA, an acronym for SEquential MAchines, is one such language (Oren and Collie 1980). Another GEST-like language is designed by Subrahmanian and Cannon (1981). Futo and Gergely (1983) developed TS-PROLOG, an advanced modelling and simulation language based on the concepts advocated by Oren and Zeigler (1979). The rational for developing such advanced tools within comprehensive modelling and simulation software systems is given in Oren (1983).

## 3.1 WORLD VIEW OF GEST

The modelling world view of GEST is based on the axiomatic system theory of Wymore (1967, 1976). 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).

A GEST program consists of three distinct parts, i.e.,

1) Mathematical model
2) Experimental frames and
3) Output module(s)

The "model" consists of a parametric model and the associated set(s) of parameter values. The experimental frame is the specification of experimental conditions (initial values of state variables, termination conditions etc.) which have to be applied to a model. It is the set of circumstances under which a model or real system is to be observed and experimented with. The output module is the specification of the output program to be used to display the result of the simulation study.

## 3.2 GEST MODEL

A specific GEST model, is a parametric model together with an optional model parameter set. The optional model parameter set may appear more than once.

### 3.2.1 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 set of component models could be coupled together through an input/output interface during coupling specification to form a coupled model.

### 3.2.2 Component Model

A component model consists structurally of two parts, namely STATIC STRUCTURE and DYNAMIC STRUCTURE. The following descriptive variables of the model are declared in the static structure part of the component model:

input variable

state variable

output variable

auxiliary variable

constants

parameter

auxiliary parameter

For continuous models, tabular functions and interpolation variable declarations can also be specified. In the DYNAMIC part, the predictive structure (or dynamic structure) of the model is specified. It also consists of the state transition and the optional output function(s).

The static structure of the model requires other declarations, such as type and range of values of the descriptive variables of the model. The type of every descriptive variable can be specified separately. In a case where no variable type has been specified, 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 model 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.

The dynamic structure consists of two blocks, namely

1) the derivative block which contains the specifications of the derivatives of the state variables and the computa-

tions of the necessary auxiliary variables. 2) the output block which contains the transformations that relate the output variables to the state and/or auxiliary variables.

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.

### 3.2.3 Examples

Some elementary examples of continuous models expressed in GEST are given in Figures 2 and 3. Figure 2 is an example of an autonomous model, which by definition, does not need input(s) to operate. Hence no input variables are specified.

```
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 2. 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 3. A continuous model expressed in GEST

## 3.3 DISCRETE AND MEMORYLESS MODELS

Both discrete and memoryless GEST models, like continuous GEST models, have two parts: the static structure and the dynamic structure. 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 part 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 an output function specification. A memoryless model transforms instantaneously its inputs and parameters into some output or outputs. 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. Figure 4 is an example of a memoryless model.

Example :

```
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 4. A Memoryless model expressed in GEST

## 3.4 COUPLED MODEL FORMALISM OF GEST

Coupling is the specification for the input/output relationship among the component models. Coupled model formalism provided in GEST, facilitates top-down model conception and step-wise model conception. Coupling specification can be very useful in the documentation of systems consisting of a large number of interacting component systems. An example of a coupled model is given in Figure 5. Its GEST representation is given in Figure 6.

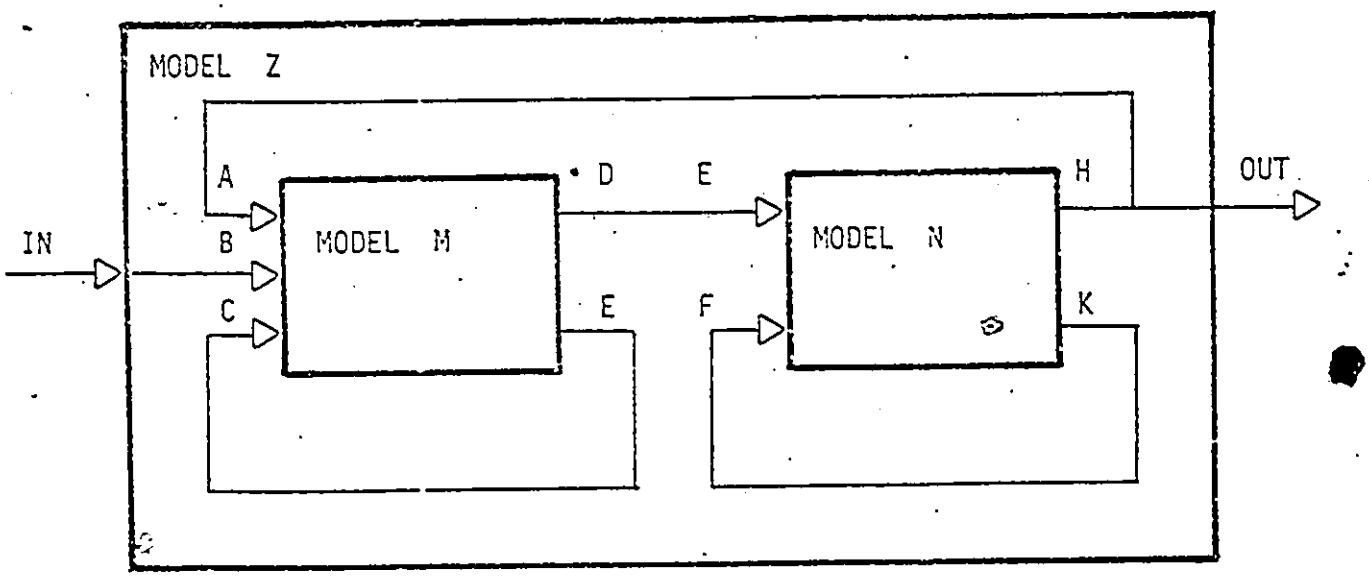



Figure 5. A coupled model

```
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 and N would appear here below:              *)
       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 6. GEST representation of the coupled model given in Fig 5.

Figures 7 a-e show steps of model conception and corresponding GEST modelling statements for the example model given in Figure 6.

In step 1, one specifies the name of the model, the input and output variables of the model, and their ranges of acceptable values (Fig. 7a).

In step 2, the names of the component models are specified (Fig. 7b).

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 (Fig. 7c).

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, for every output of the coupled model (i.e., for every external output), one would have to specify the names of the output variable and of the component model which provides the values (Fig. 7d).

In step 5, the coupling (i.e., the input/output relationships of the component models) is specified as follows (Figure 7e) :

```
For every component model do
    For every input variable do
```

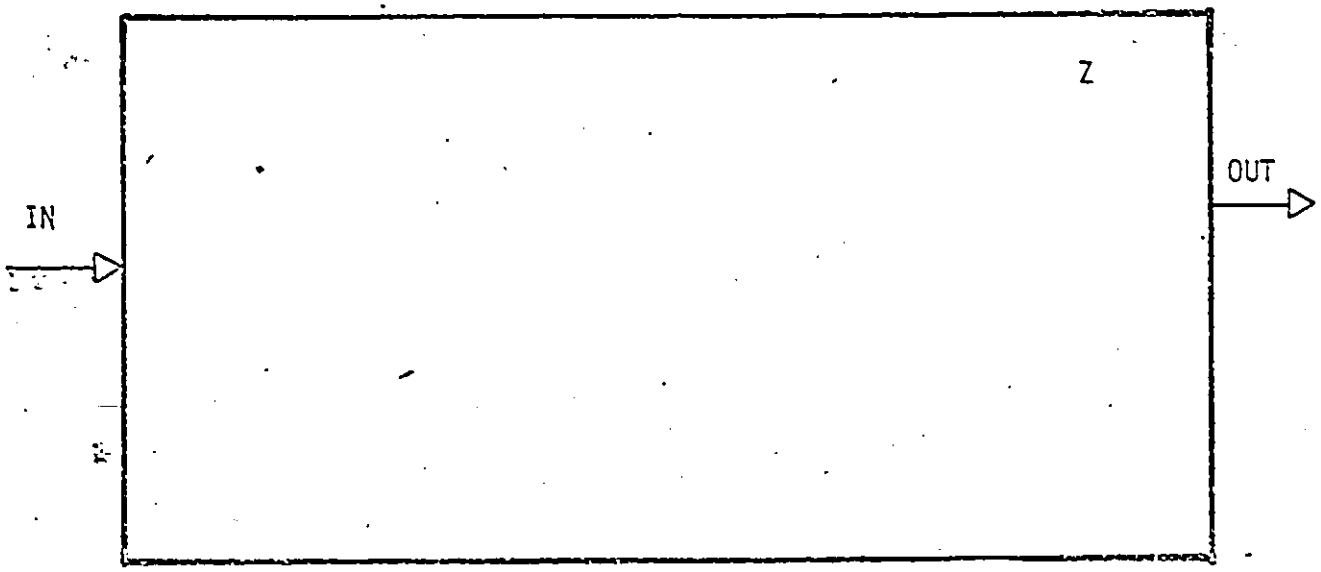
```
        Specify input-output relationship
    Loop
Loop
```

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:



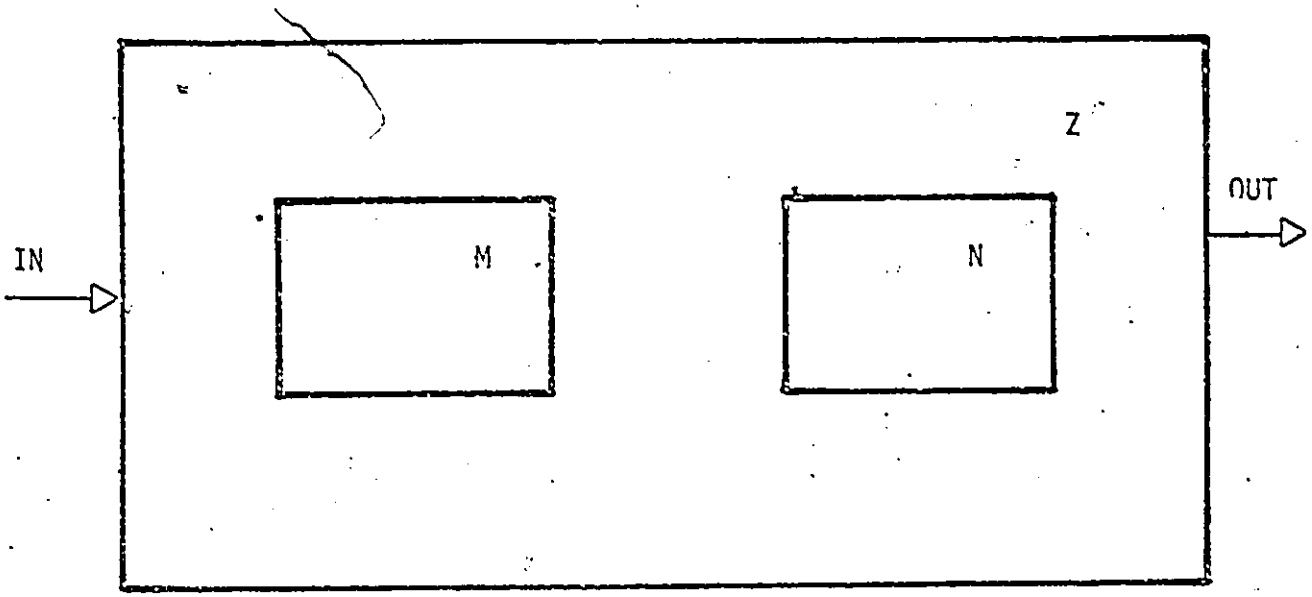

GEST MODELLING:

```
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 7a. Step 1 in top-down model conception and step-wise model refinement in GEST

STEP 2 - Specify names of component models

PICTORIAL REPRESENTATION:



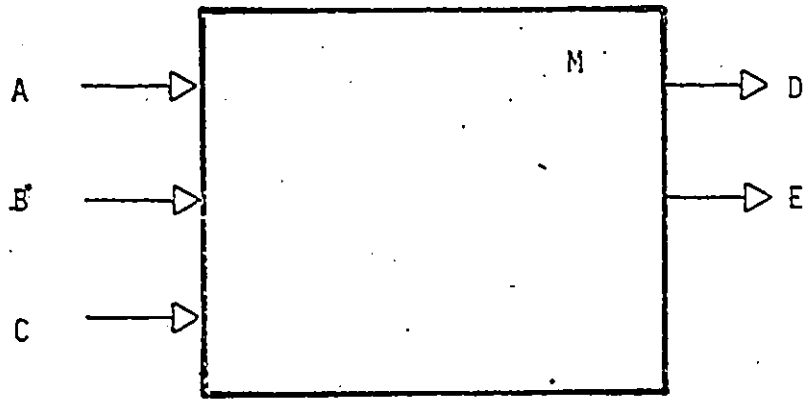

GEST MODELLING:

```
COMPONENT MODELS M, N;
```

Figure 7b. Step 2 in top-down model conception and step-wise model refinement in GEST

STEP 3 - Specify each component model separately

PICTORIAL REPRESENTATION:



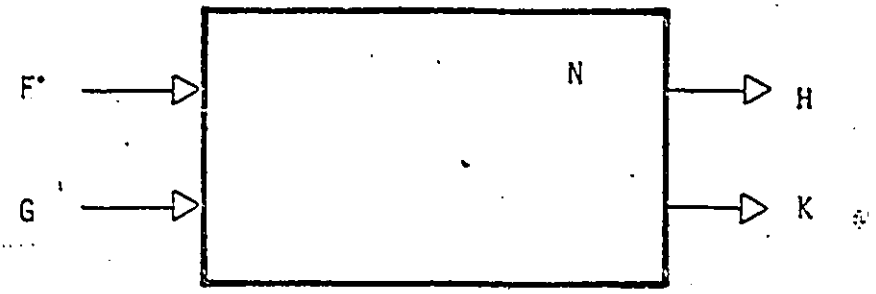

GEST MODELLING:

```
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 7c. 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:

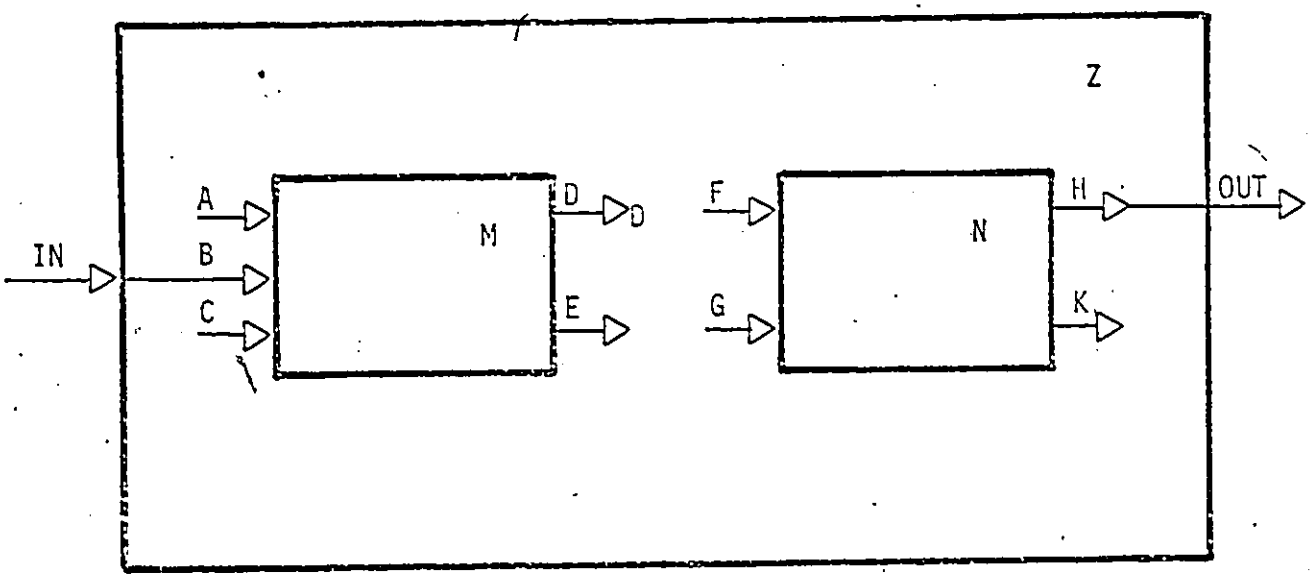
```
CONTINUOUS MODEL N;

    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 7d. 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:

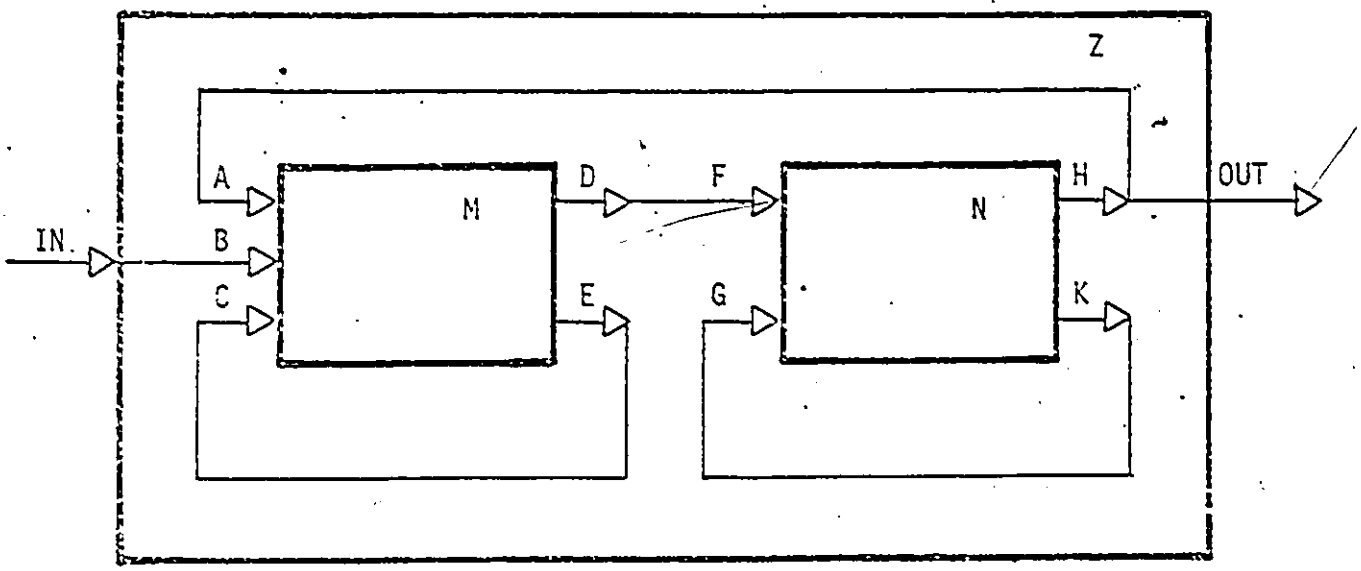
```
EQUIVALENCING
    INPUTS  Z.IN = M.B;
    OUTPUTS Z.OUT = N.H;
END EQUIVALENCING;
```

Figure 7e. 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 7f. Step 5 in top-down model conception and step-wise model refinement in GEST

## 3.5 EXPERIMENTATION

The specification of the experiments may comprise up to three sections which are as follows :

(1) experimental frames

(2) model/frame pairs (or simulation runs)

(3) post study section

Simulation experiments are specified in the experimental frame section, which also can be thought of as having two parts, namely global frame and specific frame.

In the global frame, overall specifications such as time unit declaration, termination condition specification, etc., are given. Specific frames could be given for every component model separately which would 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 the specification of the additional computations to be performed after the simulation and outputting the results of a simulation run. There is a resemblance between a post-study section and a post-run sec-

tion with an additional level of generalization in the post-study section. 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.

# Chapter 4

## GEST CERTIFICATION PROGRAM

The GEST translator developed in this thesis project, accepts as its input a "certified" GEST program. This, in particular, means that the input program has undergone a check to ensure that consistency, completeness and integrity requirements have been met. The program that performs that function is the GEST model certification program. It was developed as part of the GEST software system by Dr. K. Aytac. The program's input is a GEST model created by the use of the Template Generator or the Interactive GEST model generator which are described in the next chapter.

The certification program performs the following checks :

1) Syntactic checks :-

It checks for the correct syntax of the GEST statements, variable names, distribution functions (if any), model names,etc.

2) Integrity checks :-

Checks are performed to ensure that proper equivalencing and coupling have been done and that definitions of model parameter sets have been adequately provided. In the case of

tabular function definitions, checks are made to ensure that the number of points in the definition matches the number that has been specified in the TABULAR FUNCTION statement. The acceptability of certain values in the model are also checked. For example, the value specified in the TIME UNIT statement which will be used to advance time during the simulation run should not be zero or negative.

3) Consistency and Completeness checks :-

The existence of a model which has been declared as a component model of a coupled model is checked by this program. Moreover the specification of every component model must be complete. State variables declared in a model must be used at least once on the left hand side of an assignment statement of the derivative block. This requirement implies that the specification of state variables necessitates the presence of a derivative block and the system makes sure this is complied with. In a case where this requirement has been overlooked, the user is notified immediately and given the chance to complete the model by the use of HELP facility provided.

Output variables cannot be declared as random variables. Therefore all random variables are checked to conform to this requirement.

The type of component model implies the type of variables one has to specify during model construction. For example,

the presence of a state variable definition in a memoryless component model or an input variable definition in an autonomous model violates the integrity requirement of the system, and is checked for by this program. Other logical checks for variables are as follows ;

a) Variables with the same name and same type are not accepted. For example, in the following state variable definition,

```
STATE A, B, C, A;
```

the variable A would generate the error message : " variable already exists".

b) Redefinitions of some previously declared variables are not permissible. The user is notified of a non-permissible redefinition by an appropriate message.

The certification program contains a built-in editing function which the user can invoke during the certification process to correct some of the errors without interrupting the process. For instance, the editing function allows the user to specify any missing block or modify any block that contains an incorrect specification. When an error is detected on the current line being checked, the system prints that error line together with the error message underneath as follows:

```
*** ERROR -----> undeclared variable : name-of-variable
```

-----> Enter T for TEXT-EDIT or L for LINE-EDIT or S for STOP RUN

If the user decides to use this editing facility a menu is displayed from which to choose to correct the error. The menu appears as follows

EDITING THE CURRENT LINE MENU :

| | |
|---|---|
| C N/STRING/ | change characters with the string from Nth column |
| I N/STRING/ | insert the string into the line after Nth column |
| D N | delete the Nth character only |
| D N,NC | delete NC characters starting from the Nth column |
| EX | exit |

The certification process continues after all errors have been corrected.

Two main outputs generated by the certification program are a list of all variables used in the model together with their attributes, and an error file. Detected errors that may not be immediately corrected are written into this error file. The variables list together with the error list are of tremendous importance to the user whenever the model revision and/or debugging process takes place outside the framework of the GEST software system. When there are no detectable errors in the GEST model, the user has the option of either requesting a translation of the certified model into the target language (SIMSCRIPT) or of simply filing the

certified model for later use. In the former case the GEST translator is invoked.

# Chapter 5

## DESCRIPTION OF SOFTWARE TOOLS DEVELOPED

The main goal of this thesis project has been the development of a set of software tools to be used in conjunction with the GEST language. These tools are

-- Interactive GEST model specification tool
-- GEST Template Generator
-- GEST Translator

Before discussing these tools, their usefulness and the methodology used in the software development, it is appropriate to mention some of the issues of GEST model reliability addressed in their development. Model reliability is one of the important links in the quality assurance of system design and simulation studies (Oren 1982a). Model consistency, completeness, integrity and comprehensibility are the four main aspects of reliability that governed the development of these tools. Each of these is now briefly examined.

### 5.0.1 Model Consistency

Generation of the behavioural trajectory of a system model is possible and valid if there is a consistency in the use of variables in the component models. This implies that there should not be any ambiguity in the way variables are used in a model. An ambiguity exists if a state variable which is considered by the modeller as also representing an output variable is not explicitly declared as such. An input variable or parameter or tabular function which has already been declared , cannot be redefined as another type of variable.

The input-output relationships between the component models and the exchange of information amongst them is only possible if the variables are of the same type and range. One should not ignore the possibility of the user wrongly specifying an exchange of information between two models, one of which is non-existent. The software must check for these inconsistencies to enhance reliabilty.

### 5.0.2 Model Completeness

GEST modelling formalism requires a complete component model to have state variables and a corresponding derivative block. Also, if there exists a tabular function, there

should also be a corresponding interpolation block specification. Since GEST model specification is modular, the possibility of the user inadvertently not specifying the interpolation aspect of a tabular function should not be overlooked. Another issue arises when a system is being modelled as a memoryless model but state variables are specified by the user.

### 5.0.3 Model Integrity

The reliability of a simulation study is enhanced if the simulation software performs model integrity checks. These checks consist of ensuring the proper interaction between a coupled model and all component models (equivalencing); the interaction between the various component models (coupling) ; the parameter sets and component models; the experimental frame and component models.

An input to a coupled model could either be external or internal. An internal input to a coupled model could be an output variable of one of the component models (see Fig 5). In this case both input and output variables must agree in type and range. During the specification of the equivalencing and coupling the system must check for the validity of these interactions.

### 5.0.4 Model Comprehensibility

For a model to be reliable, it should not only meet consistency, completeness and integrity requirements but should also be comprehensible. Models (not traditional computer programs) can be made available for peer review by scientists who are involved in understanding and hence formulating models and estimating the parameters of the object system. Due to the modularity of the GEST language, programs can be read directly and subjected to peer review by scientists who should not be obliged to be computer programmers.

Comprehensibility of models could be enhanced by documenting static and/or dynamic structures of component models or their couplings at different levels. In a large scale model one can graphically document input/output relationships of component models and for any component model, one can graphically display input/output relationships with interfacing component models.

All the above aspects of reliability are handled efficiently by the software system. The overall structure of the system is presented in the following sections.

## 5.1 GEST TEMPLATE GENERATOR

The modelling and simulation activities start with the specification of the GEST model. The GEST Template Generator is one of the tools available to the user for carrying out this procedure. It is a collection of all necessary GEST statements. There are at present four types of the template that could be generated; namely, a template for: coupled model specification (CPM), continuous model specification (CNT), discrete model specification (DIS), and memoryless model specification (MEM). Depending on the type of model formalism that the user wants to use (this is communicated to the system by entering the appropriate code as stated above), the template corresponding to the requested formalism is displayed on the screen. All the user has to do is to complete the necessary blocks of interest e.g the static and dynamic structure blocks, tabular function specification, state transition function blocks, equivalencing and coupling blocks etc. Those statements and/or blocks that are not required and therefore have not been completed by the user are automatically deleted during the model certification process.

The importance of the template generator lies in the help that it provides the user in understanding the structure and syntax of the GEST language. The program listing of the template generator is shown in Appendix D. An example of a

coupled model specification template that would appear on the screen for the user to complete is shown in Figure 8.

```
PROGRAM ....
 COUPLED MODEL ....
  EXTERNAL
     INPUT  ....
     OUTPUT ....
  END EXTERNAL;
  COMPONENT MODELS ....
 CONTINUOUS MODEL   ....
 STATIC STRUCTURE
   INPUT  ....
   STATE  ....
   OUTPUT ....
   AUXILIARY VARIABLES   ....
   AUXILIARY PARAMETERS  ....
   END AUXILIARY PARAMETERS;
   PARAMETERS   ....
   TABULAR FUNCTIONS    ....
   INTERPOLATIONS       ....
   END INTERPOLATIONS;
 END STATIC STRUCTURE
 DYNAMIC STRUCTURE
    DERIVATIVES
    END DERIVATIVES;
    OUTPUT FUNCTION
    END OUTPUT FUNCTION;
 END DYNAMIC STRUCTURE;
```

```
END MODEL ........
EQUIVALENCING
  INPUTS   ....
END EQUIVALENCING;
COUPLING FOR ....
END COUPLING FOR  ....
END MODEL ....
PARAMETER SET 1 FOR  ....
END PARAMETER SET 1;
FRAME 1 FOR ....
  GLOBAL
     TIME UNIT IS ....
     SIMULATE UNTIL  ....
     START TIME ....
     INTEGRATE BY RUNGE_KUTTA;
     COMMUNICATE AT EVERY ....
  END GLOBAL;
  MODEL ....
      INITIALISE STATE ....
      SAVE ....
      INTERPOLATION LINEAR;
  END MODEL  ....
END FRAME 1;
RUN 1
END RUN 1;
OUTPUT MODULE 1
END OUTPUT MODULE 1;
```

END PROGRAM ....

Figure 8. Coupled Model Specification Template

## 5.2 INTERACTIVE GEST MODEL GENERATOR

The purpose of this tool is to guide and aid the user in specifying the GEST model. It helps the user to bypass some of the difficulties that might arise from the time and effort required to learn the GEST language, its structure and syntax.

In using this tool, the user participates in a dialogue with the system. During the definition dialogue, the system builds the model data structures. It is our conviction that the process of system modelling can be made much more effective, reliable and greatly speeded up by interactive computer assistance. Some of the reliability issues are handled during the dialogue by prompting the user to give specific definitions to already declared variables. For instance, in coupling specification, since all component models with their descriptive variables are known to the system, the software system prompts all necessary questions and when the connected variables are specified by the user, the system can verify whether or not such variables do exist, and if they do whether their ranges are compatible. If a tabular function is declared to be discontinuous, the corresponding parameter set is verified to have a tabular function with at least one discontinuity. In the case of dynamic structure of models, descriptive variables specified in static structure and not used in the dynamic structure can be detected. At

the end of the dialogue, the specified model could be stored for later use or the parameter set together with the experimental frame could also be specified and simulation run initiated. It should be noted that the resultant GEST model has to be certified irrespective of what option the user uses.

An illustration of the dialogue to specify a GEST model using this tool is presented in section 6.3.

## 5.3 GEST TRANSLATOR

The translator takes as input a certified GEST program and translates it into SIMSCRIPT II.5 code (henceforth, simply SIMSCRIPT). Its goal is to produce a target code which is as complete and error-free as possible without the need for further modifications by the user. SIMSCRIPT has been choosen as the target language for the following reasons :

1) It is a rich and versatile computer programming language well suited to general programming problems, although designed originally for discrete-event simulation applications.

2) It is as powerful as other programming languages like FORTRAN, PL/1 and Algol.

3) It allows access to external non-SIMSCRIPT routines.

4) It is modular in structure and the source code is self-documenting.

An effective and often used method for modularising a program is to divide it into program functional elements such as data declarations (and structure), initialization, input data handling, events (in the case of simulation models), output processing, support routines, and low level utility routines. This division technique was followed in this simulation software development. The modularization hierarchy exhibited in GEST is a schematic conceptualization which includes system static and dynamic structure descrip-

tions, experimental frames --- model parameter set, global frame, initialization data, run instruction and output modules.

The translation phase is not a straight forward process since the implementation language and the target language used in this project have different structures. The specification and key word oriented nature of GEST does not lend itself to easy and direct translation to SIMSCRIPT. One has to sift out information from various portions of the GEST program and assemble them together into the target module. The translator performs an initial check on the input GEST program to see whether it has been certified or not. This is necessary because a user's program must be checked for completeness, consistency and reliability before translation begins. In the case where certification has not been done before translation is requested, control is automatically given to the certification program.

Three levels of activities are recognised by the model translator namely,

1) pre-run activities
2) run-time activities
3) post-run activities

Pre-run activities involve the declaration of variables, mode of computation, functions and library routines; the

initialization of state variables; the computation of auxiliary parameters and output functions; and the reading in of tabular function values (if any). Information necessary to establish these activities have to be taken from the GEST model at various points.

Run-time activities include the computation of interpolation and output functions, the establishment of simulation time duration and time increment, and the integration procedure. Detection, location and handling of discontinuities are also initiated at this level. In other words, this level of activity comprises the actual simulation experimentation.

The Post-run activities include the display and analysis of the simulation results. The user could specify the results to be listed, plotted or print-plotted. The information employed in this level is mainly derived from the output module specification section of the GEST model.

While scanning through the GEST text for the necessary information, the set of statements that would form the derivative function to be used during the integration process have to be built. Since we are not using modular integration methods for the simulation, it is only convenient to lump up the respective derivative functions into one module. However future enhancements to the software system would consider the possibility of modular simulation. Such simulation needs coordination algorithms to keep the various subsystems in

time synchronization and to provide them with their inputs which come from other subsystems.

Three files are maintained during the translation process each corresponding to the three levels of activities mentioned above. The three files are merged together via the EXEC file to form a single SIMSCRIPT program file for compilation and execution.

A description of the subroutines that are used by the translator program is provided below. This will not only serve as a partial documentation but will greatly enhance the understanding of the translation process.

### 5.3.1 HIPO Description of routines used in program

#### 5.3.1.1 SKIPCOMMENTS routine

Comments could be inserted into the GEST program during or after model generation, but before certification. These comments are ignored during the translation. It is this routine that is responsible for this activity.

Hierarchy

This routine is called by almost all routines each time a READ statement is executed. It is also recursive.

Input

There are no explicit arguments to this routine. However it operates on the contents of LNTEXT (see list of program variables below).

Process

1) Determine whether the input text line contains comments which is indicated by (* and *) as representing the beginning and ending of comments respectively.

2) If beginning comment indicator exists and ending comment indicator does not exist, set XBCFLAG to 1 and replace comment statements with blanks.

3) If beginning and ending comment indicators do not exist and XBCFLAG has been set to 1, then the whole input text line is a comment line and has to be ignored. Another input text is read in and the subroutine invoked recursively.

4) If beginning comment indicator is not present but ending comment is present and the value of XBCFLAG has been set to 1, then this signifies that this is the last of a series of comments enclosed in one pair of comment indicators, hence XBCFLAG is reset to 0.

Output

The implied output is the current text line free of comments. This is then used by other routines to retrieve the necessary information.

### 5.3.1.2 GLOBAL ROUTINE

This routine prepares the statements that appear in the GLOBAL section of the GEST model experimental frame specification. Information regarding simulation time duration, communication time interval, initial starting time for the simulation and time unit used are derived from the experimental frame by this routine. Starting time is assumed to be zero unless explicitly given another value.

Hierarchy

Called in the main program

Calls CHECK.ARRAYS, CHECK.INTERPOLATION and SKIPCOMMENTS routines.

Input

No explicit imports

Process

1) Check for the various types of GEST statements that may appear in this portion of the program.

2) Retrieve the appropriate information from the particular statement type being processed and generate equivalent SIMSCRIPT statement.

3) Create a vector of all state variables used in the model. This vector will be used in the derivative block.

4) Check whether a state variable is dimensioned or not by invoking the CHECK.ARRAY routine. If it is dimensioned generate the number of elements in the array to reflect the total number of state variables in the created vector. For example, a list of state variables could appear as

STATE A, STN, FLOW(1..5), DF(1..10) ;

In the above state variable declaration there are two non-scalar state variables and two dimensioned variables. During the translation the total number of state variables in the created vector should appear as 17 and storage space should be allocated using the RESERVE statement in SIMSCRIPT.

Output

An augmented file of statements that will make up the MAIN portion of the generated target program.

### 5.3.1.3 INITIALISE routine

The INITIALISE routine is responsible for generating the code for initialising the state variables prior to the simulation run. Initialization could be done by one or combination of five ways, namely,

i) by data-specification : Examples in GEST are

```
INITIALISE STATES S(1..10) = 3 * 0.0, 5 * 1.0, 2 * 0.0 ;
INITIALISE STATES A = B(3) = C7 = 14.0 ;
```

ii) by data-reading : This could be achieved in GEST by the statement

```
INITIALISE STATES BY READ A, B, C ;
```

iii) by saved values : which could be specified as

```
INITIALISE STATES BY SAVED VALUES A, B, C ;
```

iv) initialization-by-computation : An example is

```
INITIALISE STATES BY COMPUTING
V = SQRT ((VA - VF * SIN(THETA)) ** 2
W = V * COS(THETA) ** 2
END INITIALISE BY COMPUTING
```

v) by setting values to zero : Initialization by setting values to zero could be specified in GEST as

```
INITIALISE STATES TO ZERO
INITIALISE STATES A, B, C TO ZERO
INITIALISE STATES Y(1..15) TO ZERO
```

Depending on the type of initialization, this routine is responsible for analysing the statement and generating the corresponding set of SIMSCRIPT statements.

Hierarchy

Called by the main program

Calls DATA.STATEMENT, and SKIPCOMMENTS routines.

Input

No explicit imports

Process

1) Determine the form of the initialise statement and extract the state variables named in the statement (if any) and check whether they do exist in the list of state variables of the model.

2) If it is initialization-by-data-specification then call the DATA.STATEMENT routine to decompose and assemble the assignments.

Output

Generated SIMSCRIPT code representing the initialization of the model.

### 5.3.1.4 SAVE OUTPUT routine

This routine handles the SAVE statement in GEST by generating the code for saving the outputs of some variables. The SAVE statement could be of two forms

SAVE STATES or INPUTS or OUTPUTS

SAVE A, B, C AT EVERY 0.03 SECOND

In the case of the first form the system has to rely on information that has been extracted earlier on from the model and generate the appropriate code.

Hierarchy

Called by INITIALISE routine.

Calls RETRIEVE.ROUTINE.

Input

There are no explicit imports. The operation is done on the contexts of the global variable LNTEXT (i.e the current input text line).

Process

1) Check whether the SAVE statement is of type I or type II.

2) If it is type I fetch the information stored and generate the code.

3) If it is type II then invoke the RETRIEVE.ROUTINE to extract the variable names to be saved as output. Check the list of variables to see whether they exist and generate the code. Terminate execution if any of the variables is not in the list.

Output

Translated version of the GEST statement.

5.3.1.5 SORT routine

This routine is responsible for ordering the set of statements contained in the derivative and output function blocks. These statements may be specified in a manner not convenient for direct computation and have to be pre-sorted before execution. For example

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

According to the above output function specification, U cannot be evaluated without knowing the values of P and D. Let us assume that the other variables are known parameters. In order to evaluate U the set of statements in the block should be rearranged as follows:

```
P = G * E ;
D = -GD * (Y - X) ;
U = P + I + D ;
```

by the SORT routine.

Hierarchy

Called by DYNAMIC.SECTION and OUTPUT.FUNCTION routines.

Input

The import parameters are INTEXT and N. INTEXT is an array of the statements to be sorted and N represents the number of statements in the array.

Process

1) Check whether variables on the right hand side of the current arithmetic expression are all known. If yes then evaluate this statement else if any variable is not known then defer evaluation until that variable is known.

2) Get next arithmetic statement

3) Repeat 1 and 2 until all values are known.

Output

Sorted (computable) set of statements

### 5.3.1.6 DATA.STATEMENT routine

This routine is responsible for decomposing value assignment statements and generating the equivalent code. A GEST value assignment statement could be specified as follows

```
TCM(2..18) = 5*1.0, 5*1.5, 3*2.5, 3*5.0, 10.0 ;
```

Since there is no single data definition statement in SIMSCRIPT to allow for one-to-one translation, the above statement has to be decomposed and translated automatically into series of statements as follows :

```
I = 2
FOR J = 1 TO 5 DO
  LET  TCM(I) = 1.0
  ADD 1 TO I
LOOP
FOR J = 1 TO 5 DO
  LET  TCM(I) = 1.5
  ADD 1 TO I
LOOP
FOR J = 1 TO 3 DO
  LET  TCM(I) = 2.5
  ADD 1 TO I
LOOP
FOR J = 1 TO 3 DO
  LET  TCM(I) = 5.0
  ADD 1 TO I
LOOP
  LET  TCM(I) = 10.0
```

Hierarchy

Called by MAIN.PORTION and INITIALISE routines

Calls RETRIEVE.ROUTINE

Input

The import parameter is N which is a pointer to the first character on the right side of the " = " sign in the expression.

Process

1) Find out whether the expression is a single or composite assignment statement. If it is a single statement of the form

scalar-variable = constant or simple expression

then exit from the routine. Otherwise identify and expand the expression by generating the target code.

Output

Set of simple assignment statements.

### 5.3.1.7 CHECK.ARRAYS routine

This routine checks to see whether variables declared are scalar or subscripted variables.

Hierarchy

Called by the main program, SKIP.BLANKS, and GLOBAL routines.

Input

The input parameter STATEVECT contains the variable to be checked.

Process

The process here is simple. The presence of an open and close brackets implies that the variable is subscripted. We then extract the lower and upper bounds enclosed in the brackets.

Output

The lower and upper limits of the subscripted variables are returned.

### 5.3.1.8 CHECK.INTERPOLATION routine

This routine prepares GEST Interpolation specification statements into the form acceptable to the interpolation routine used in the translated program during simulation.

Interpolation may be specified in GEST as

RATE1 = ABC (T) where ABC is a function name, RATE1 and T are scalar variables. This specification would be expressed in the target program as RATE1 = INTERPOLATION (T, ABC(*,*)) where ABC(*,*) is a pointer to the storage position of the 2-dimensional function table.

Hierarchy

Called by GLOBAL routine.

Input

Takes the number of function points, POINTS, as import parameter.

Process

1) Extract the LHS variable, the tabular function name and the interpolation variable.

2) Check whether the extracted tabular function name exists. Terminate the translation process if it does not exist with the appropriate error message.

3) Re-assemble the extracted variables into the SIMSCRIPT function statement.

Output

Translated interpolation statements

### 5.3.1.9 RETRIEVE.ROUTINE routine

This routine extracts user-defined variable names from GEST statements.

Hierarchy

Called by the main program

Calls SKIP.BLANKS and SKIPCOMMENTS routines

Input

There is no explicit input parameter. However a global variable I serves as a pointer to the beginning of the variables to be retrieved.

Process

1) Skip over the GEST key word(s) and locate the position of a comma or semicolon in the current text line.

2) Extract the characters from the pointer to the position of the comma or semi colon and advance the pointer.

3) Invoke SKIP.BLANKS to suppress leading/trailing blanks and/or SKIPCOMMENTS routine to ignore comments.

Output

Variables extracted are stored in a 1-dimensional text array, MAINVAR for subsequent processing.

### 5.3.1.10 WRITE.ROUTINE routine

In our target language, global variables, function names and arrays must be declared in the PREAMBLE section. This declaration is effected by this routine after the elements have been extracted from the GEST statement definitions.

Hierarchy

Called by the main program normally as a follow-up to the RETRIEVE.ROUTINE.

Input

It takes a 1-dimensional text array VARABLE as its import parameter.

Process

1) Check to see whether the variables to be declared are scalar or structured. There are two integer variables, INDEX and ARRAY.INDEX associated with scalar and structured variables respectively. The value of any one of them signifies the number of elements to be declared.

2) If the value of INDEX is non-zero then declare the set of variables as scalar variables of the appropriate type.

3) If the value of ARRAY.INDEX is non-zero then declare the variable set as appropriate.

Output

The output is the declaration statements in the PREAMBLE section of the translated SIMSCRIPT program.

### 5.3.1.11 DYNAMIC.SECTION routine

This is the routine that organizes the various derivative functions of the component models to be used during the integration.

Hierarchy

Called in the main program when 'DYNAMIC STRUCTURE statement is encountered whilst scanning through the program.

Calls SKIPCOMMENTS and SORT routines.

Input

There are no explicit imports.

Process

1) Read and store each assignment statement in an array, DYN.SECT.

2) Check to see if the statements are in a form such that the left-hand side can easily be evaluated. If the derivatives have been specified in such a way that it needs presorting then invoke the SORT routine.

3) Combine the sorted set of statements with any previous set of derivative functions.

Output

Output is an organised combined set of statements to be used as the derivative block for the total system.

### 5.3.1.12 OUTPUT.FUNCTION routine

This routine combines the output functions of the respective component models. This combined block is executed during the pre-run activity level to establish the initial inputs to the various component models and executed after each integration step to establish the outputs.

Hierarchy

Invoked in the main program on encountering OUTPUT FUNCTION statement.

Calls SKIPCOMMENTS and SORT routines.

Input

No explicit imports.

Process

1) Read and store the statements in an array, OUT.FUNC.

2) Check to see if the left-hand side expression can be evaluated easily. If not, invoke the SORT routine to sort the statements.

Output

A combined set of statements that would form the output function block for the total system.

5.3.1.13 MAIN.PORTION routine

This routine assembles the run-time activity statements. Initialization statements, output function block statements, tabular function block values are organized for the simulation run by this routine.

Hierarchy

Called in the main program.

Calls DATA.STATEMENT, and SKIPCOMMENTS routines.

Input

No explicit inputs.

Process

1) Reserve storage for the arrays to be used

2) Get specification of the step size for the integration if not stated in the model.

3) Establish values for the external inputs (if any). It should be noted that these values may not be specified during model construction.

4) Read in parameter set and tabular function values. Remove commas and brackets that separate pairs of values in the tabular function specifications to allow for easy manipulation.

5) In a case where composite data statements are used (i.e

W(1..18) = 5*0.2718 , 9*1.2135 , 4*0.0105 )

the subroutine DATA.STATEMENT is invoked to handle these statements.

Output

An output file consisting of generated SIMSCRIPT statements is created.

# Chapter 6

## EXAMPLES OF THE USAGE OF THE TRANSLATOR

In this chapter two GEST models are used to illustrate the usage of the translator. Neither of the examples contains discontinuity because the discontinuity handling feature in GEST has not yet been fully integrated into the GEST software system. The example problems were both solved using a fourth order Runge-Kutta integration algorithm. The translator was executed on the University of Ottawa Amdahl 470/V6 machine running under VM/CMS.

### 6.1 EXAMPLE PROBLEM 1 DEER POPULATION MODEL

This model has been formulated as a continuous model. Its purpose is t to determine whether a policy could be designed to maintain a stable deer population within a particular geographical area. The mnemonics used in the model description are as follows :

DP = DEER POPULATION
DNI = DEER NET INCREASE
NIR = NET INCREASE RATE
DPR = DEER PREDATION RATE
DKPP = DEER KILL PER PREDATOR
DKPPT = DKPP TABLE

PP = PREDATOR POPULATION

PPT = PP TABLE

The equations and numerical parameters that define all inter-relationships are as follows :

DP' = DNI - DPR

DNI = NIR * DP

DPR = PP * DKPP

DD = DP / AREA

The initial value of DP is set to 4000. Two parameters AREA and NIR have values of 800000 and 0.2 respectively. We are interested in generating the trajectories of DP, PP and DKPP over a period of time from 1881 to 1970. In this formulation the problem independent variable has the units of years.

## 6.2 EXAMPLE PROBLEM 2 MOTOR MODEL

Our second example is a coupled model consisting of two component models namely PID_CONTROLLER and MOTOR model. The descriptive variables of the PID_CONTROLLER model are represented as follows :

YREF and Y are input variables, I and X represent state variables, U represents output from the model, E, P, D are

auxiliary variables, G, GD, TD are the parameters of the model. The equations describing the system state are as follows :

$$I' = E / TI$$

$$X' = -GD / TD * (X - Y)$$

$$E = YREF - Y$$

These equations appear in the dynamic structure portion of the GEST program. The initial values of I and X are all set to zero. The output, U, is computed using the formula

$$U = P + I + D$$

$$P = G * E$$

$$D = -GD * (Y - X)$$

The descriptive variables of the MOTOR model are represented as follows : U for input, TH and THDOT for state variables, Y for output, ME and I for auxiliary variables, KM, R, J, and CT represent the parameters of the model. The state of the model is described by the following equations

$$TH' = THDOT$$

$$THDOT' = ME / J$$

$$ME = KM * I$$

$$I = (U - KM * THDOT) / R$$

The initial values of TH and THDOT are set to zero. The output of this model is computed using the equation

```
Y = CT * TH
```

The coupling or input-output specification is done as follows : There is an external input to the coupled model represented by YREF which is also one of the inputs to the PID_CONTROLLER model. The other input is an output from the MOTOR model represented by Y. The output from the PID_CONTROLLER model provides the input to the MOTOR model.

The coupled GEST model has been translated into the target language and run. The model, its translated version and simulated results are listed in Appendix H, I, and J respectively.

## 6.3 DEMONSTRATION RUN FOR THE MOTOR MODEL

In order to initiate execution of the software system, the user types the command :

GEST

This command executes the EXEC file (a listing of which appears in Appendix B). The EXEC file contains CMS commands that control the set of activities to be performed using this software system. The GEST command produces the following output on the screen

```
        WELCOME TO GEST SIMULATION SOFTWARE
    THIS SOFTWARE SYSTEM PROVIDES FACILITIES FOR

 1    INTERACTIVE SPECIFICATION OF GEST MODEL
 2    CERTIFICATION OF A MODEL
 3    TRANSLATION OF A SPECIFIED MODEL
 4    EXECUTION OF A TRANSLATED MODEL
 5    LIST MODELS IN THE MODEL BASE

ENTER THE FACILITY YOU WANT PLEASE
```

The user could request for any of the above facilities. If one wants to construct a model from scratch, then '1' is entered. Models could be retrieved from the model base for direct use or for modification.In this case the user has to enter '5' in order to know the models that exist in the model base. An existing model which does not suit the user very well could be modified using the CMS XEDIT function.

Facility 1 also provides three other facilities. For example, if 1 is entered the following will appear on the screen :

```
THIS FACILITY PROVIDES FOR THE SPECIFICATION
                OF THE FOLLOWING

1  COMPONENT MODELS
2  COUPLED MODEL SPECIFICATION
3  PARAMETER SET SPECIFICATION
```

The user again has a choice to make. Once the model has been specified, control is given back to the calling EXEC program to prompt the user for the next action to be taken. This is necessary because the user has the freedom to stop at the specification level; that is, if no translation or execution is required at the moment.

The following is an example dialogue between the system and the user. during coupled model specification. Any requested information that may not be needed could be ignored by hitting the return key.

```
2


COUPLED MODEL SPECIFICATION BEGINS.....

ENTER COUPLED MODEL NAME
motor-controller
```

```
ENTER EXTERNAL INPUT(S) IF ANY
yref
ENTER EXTERNAL OUTPUTS  IF ANY

ENTER THE NUMBER OF COMPONENT MODELS
2

ENTER NAME OF COMPONENT MODEL          1
pid-controller
ENTER NAME OF COMPONENT MODEL          2
motor


 CONTINUOUS MODEL SPECIFICATION FOR    PID-CONTROLLER BEGINS.....


ENTER INPUT VARIABLES
yref,y
ENTER STATE VARIABLES
i,x
ENTER OUTPUT VARIABLES
u
ENTER AUXILIARY VARIABLES
e,p,d
ENTER PARAMETER VARIABLES
g,gd,td,ti
ENTER TABULAR FUNCTIONS
```

```
ENTER INTERPOLATIONS

ENTER DERIVATIVES
i'=e/ti
x' = -gd/td*(x - y)
e = yref - y

ENTER OUTPUT FUNCTION
u = p + i + d
p = g * e
d = -gd * (y - x)
```

CONTINUOUS MODEL SPECIFICATION FOR MOTOR BEGINS.....

```
ENTER INPUT VARIABLES
u
ENTER STATE VARIABLES
th,thdot
ENTER OUTPUT VARIABLES
y
ENTER AUXILIARY VARIABLES
me,i
ENTER PARAMETER VARIABLES
km,r,j,ct
```

```
ENTER TABULAR FUNCTIONS

ENTER INTERPOLATIONS

ENTER DERIVATIVES
th' = thdot
thdot' = me/j
me = km * i
i = (u - km * thdot)/r

ENTER OUTPUT FUNCTION
y = ct * th

EQUIVALENCING
 MOTOR-CONTROLLER. YREF =
pid-controller.yref

COUPLING FOR  MOTOR-CONTROLLER


PID-CONTROLLER.YREF  <-----   (ENTER THE SOURCE)
motor-controller.yref

PID-CONTROLLER.Y  <-----   (ENTER THE SOURCE)
motor.y

MOTOR.U  <-----   (ENTER THE SOURCE)
pid-controller.u

PARAMETER SET SPECIFICATION FOR  MOTOR-CONTROLLER;
```

```
  P I D - C O N T R O L L E R

G =  (ENTER THE EXPRESSION)
1.0

GD =  (ENTER THE EXPRESSION)
1.0

TD =  (ENTER THE EXPRESSION)
1.0

TI =  (ENTER THE EXPRESSION)
1.0 *10**10
  M O T O R

KM =  (ENTER THE EXPRESSION)
6.2 * 10 ** (-3)

R =  (ENTER THE EXPRESSION)
5.3

J =  (ENTER THE EXPRESSION)
7.5 * 10 ** (-7)

CT =  (ENTER THE EXPRESSION)
0.033
EXPERIMENTAL FRAME SPECIFICATION FOR  MOTOR-CONTROLLER
ENTER TIME UNIT
second
ENTER SIMULATION STARTING TIME
```

```
1.0
ENTER FINAL TIME
5.0
ENTER COMMUNICATION TIME INTERVAL
0.01 second
ENTER INITIALISE AND SAVE STATEMENTS FOR
PID-CONTROLLER
initialize states to zero
save u

ENTER INITIALISE AND SAVE STATEMENTS FOR
MOTOR
initialise states
th = 0.0
thdot = 0.0
save y

PREPARE THE OUTPUT MODULE STATEMENTS
print 1 heading line
study of motor-controller
print u, y versus time

    END OF A SESSION.....
```

At the end of a session the model that has been specified could be displayed on the screen as below for verification and/or update.

```
PROGRAM  MOTOR-CONTROLLER;
EXTERNAL
  INPUT YREF;
END EXTERNAL;
COMPONENT MODELS  PID-CONTROLLER, MOTOR
CONTINUOUS MODEL PID-CONTROLLER;
  STATIC STRUCTURE
    INPUT  YREF,Y;
    STATE  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 COMPONENT MODEL  PID-CONTROLLER
  CONTINUOUS MODEL MOTOR;
    STATIC STRUCTURE
      INPUT  U;
      STATE  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 COMPONENT MODEL  MOTOR
EQUIVALENCING
  INPUTS
  YREF =  PID-CONTROLLER.YREF;
  END EQUIVALECING
```

```
COUPLING FOR MOTOR-CONTROLLER
  PID-CONTROLLER.YREF <----  MOTOR-CONTROLLER.YREF;
  PID-CONTROLLER.Y <----  MOTOR.Y;
  MOTOR.U <---- PID-CONTROLLER.U;
END COUPLING FOR  MOTOR-CONTROLLER;

 END MODEL MOTOR-CONTROLLER;


 PARAMETER SET  FOR  MOTOR-CONTROLLER;

MODEL  PID-CONTROLLER          ;
   G =  1.0;
   GD =  1.0;
   TD =  1.0;
   TI =  1.0 *10**10;

MODEL  MOTOR                   ;
   RM =  6.2 * 10 ** (-3);
   R =  5.3;
   J =  7.5 * 10 ** (-7);
   CT =  0.033;
 END PARAMETER SET 1
 FRAME 1 FOR  MOTOR-CONTROLLER
  GLOBAL
    TIME UNIT IS  SECOND;
    START TIME IS  1.0;
    SIMULATE UNTIL TIME >  5.0;
    INTEGRATE BY RUNGE-KUTTA
```

```
    COMMUNICATE AT EVERY  0.01 SECOND;
 END GLOBAL
MODEL  PID-CONTROLLER
 INITIALIZE STATES TO ZERO;
 SAVE U;
END MODEL  PID-CONTROLLER
MODEL  MOTOR
 INITIALISE STATES;
 TH = 0.0;
 THDOT = 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 1 HEADING LINE;
    STUDY OF MOTOR-CONTROLLER;
    PRINT U, Y VERSUS TIME;
 END OUTPUT MODULE 1
 END PROGRAM  MOTOR-CONTROLLER
```

At this point the model could be certified and translated as directed by the EXEC file.

# Chapter 7

## CONCLUSIONS

Simulation software technology in general is deficient in that no single methodology covers the entire development process from the beginning to end.

An attempt has been made to develop a GEST translator (and the associated model specification tools) to translate a certified GEST model to a SIMSCRIPT code for execution. Until the realisation of a compiler for the GEST language this project should satisfy a practical need i.e, it will make the GEST system accessible to the simulation community for simulation studies of continuous-change system.

The translator can only handle continuous-change GEST models at the present time. In the context of simulation translation activity, the model may include discontinuities but before the execution of such models can take place, it will be necessary to incorporate a suitable discontinuity handling integration algorithm into the run-time library.

One challenging aspect in the development of the translator was that of formulating a means to handle the derivatives of different submodels in a coupled model during the integration process. There are two approaches available. One

approach is to use a modular integration method for the simulation. The second approach is to lump up all the derivatives together into one module and carry on with the simulation. The possibility of modular simulation was ruled out because such simulation needs co-ordination algorithms to keep the various subsystems in time synchronization and these were not available. The only practical approach therefore was to settle for the second alternative. However the possibility of modular simulation should be considered in future enhancements to the software system.

One other area that had to be considered while developing the translator was the way equations of the derivative and output functions were defined in the GEST model. As an example, the derivatives of the Deer model are specified in such a way that the equations cannot be correctly executed without re-arrangement of the sequence. This problem is effectively handled by the SORT procedure as explained in subsection 5.3.1.

Values can be assigned to arrays in the GEST program using a FORTRAN-type data statement ; e.g.

```
TCM(2..18) = 5*1.0, 5*1.5, 3*2.5, 3*5.0,10.0
```

Since there is no such equivalent statement in SIMSCRIPT, care has to be taken to decompose the data statement into a series of statements as described in the DATA.STATEMENT routine.

One limitation of the SIMSCRIPT language which the author has observed during the implementation of the translator is that an arithmetic statement in the form of LET DM = 1.0E15 causes difficulty for the SIMSCRIPT compiler because it appears to consider 1.0E15 as an undefined variable.

Possible areas of future work are the following :

1) The translator could be modified to handle models expressed in discrete or combined discrete-continuous formalisms

2) The software could be modified to explicitly allow users to define switch functions for characterizing discontinuous systems. Currently, discontinuities are defined in GEST using the IF_THEN_ELSE construct and therefore switch functions have to be generated by the translator before the problem can be executed.

## Appendix A

## TRANSLATOR PROGRAM LISTING

```
''*************************************************************
''*                                                           *
''*  TRANSLATOR FOR THE GEST MODEL SPECIFICATION LANGUAGE     *
''*                                                           *
''*                    BY FELIX DOGBEY                        *
''*                                                           *
''*************************************************************
PREAMBLE
   DEFINE NUMOFLINES, FLAG3  AS INTEGER VARIABLES
   DEFINE I, J, N, M, F,P, NOCM,NOTF,NOOF,NODF AS INTEGER VARIABLES
   DEFINE  LNTEXT, VAR,SLNTEXT,XTENT,STEP AS TEXT VARIABLES
    DEFINE INDEX,TNOSV,NOOS,SAFE,NOPOINTS AS INTEGER VARIABLES
    DEFINE STN,POINTS AS 1-DIM INTEGER ARRAY
    DEFINE MSTATE AS 2-DIM TEXT ARRAY
    DEFINE FUNCNAME,SPACE,BEQUAL,AFTEQUAL,PARENT,CHR AS TEXT VARIABLES
    DEFINE INPUT, STATE, OUTPUT,STATE.VECTOR AS 1-DIM TEXT ARRAYS
    DEFINE AUXY,EXAUXY, TABULAR, AND INTERPOL AS 1-DIM TEXT ARRAYS
    DEFINE PARAMETER, OUT.FUNC,DYN.SECT AS 1-DIM TEXT ARRAYS
    DEFINE TAB,VAR.LIST,EXINPUT,STATE2 AS 1-DIM TEXT ARRAYS
    DEFINE FFLAG,BFLAG,NEXTIN,ARRAY.INDEX AS INTEGER VARIABLE
    DEFINE SAVE, SSAVE,ARRAY,EXINTERPOL AS 1-DIM TEXT ARRAYS
    DEFINE NOINT,NOAV, NUMBER,XBCFLAG,FLAG2 AS INTEGER VARIABLES
```

```
    DEFINE CLOSBRACKET,OPBRACKET,TWODOTS,BLANK AS TEXT VARIABLES
    DEFINE STCODE,ARRAY2.INDEX AS INTEGER VARIABLE
    DEFINE BEG,END,SEMICOLON,COMMA,EQUALSIGN AS TEXT VARIABLES
    DEFINE AMPERSAND,QUOTE,LTET,LTET2 AS TEXT VARIABLES
    DEFINE R.ARRAY AS 1-DIM TEXT ARRAY
END
MAIN
   RESERVE INPUT, STATE, OUTPUT,EXINPUT AS 10, ARRAY,R.ARRAY AS 20
   RESERVE AUXY, TABULAR, INTERPOL, STATE2 AS 10
   RESERVE PARAMETER,SAVE,SSAVE,POINTS,EXINTERPOL AS 10
   RESERVE VAR.LIST AS 30
   RESERVE STATE.VECTOR,EXAUXY AS 15
   RESERVE TAB,OUT.FUNC,DYN.SECT AS 80
   RESERVE STN AS 5
   RESERVE MSTATE AS 5 BY 10
  USE 80 FOR INPUT
'' LET EOF.V = 1
   LET F = 1
   LET CLOSBRACKET = ")"
   LET OPBRACKET = "("
   LET TWODOTS = ".."
   LET BLANK = " "
   LET SEMICOLON = ";"
   LET COMMA = ","
   LET AMPERSAND = "&"
   LET EQUALSIGN = "="
   LET QUOTE = '"
```

```
  WRITE AS " PREAMBLE",/ USING 21
  WRITE AS " NORMALLY, MODE IS REAL",/ USING 21
  WRITE AS " DEFINE W, DW AS 1-DIM ARRAY",/ USING 21
  WRITE AS " DEFINE H AS A VARIABLE",/ USING 21
  WRITE AS " DEFINE TNOSV2 AS INTEGER VARIABLE",/ USING 21
  WRITE AS " DEFINE INTERPOLATE AS FUNCTION",/ USING 21
  WRITE AS " ''*****************************",/ USING 20
  WRITE AS " ''*     DERIVATIVE          **",/ USING 20
  WRITE AS " ''*      FUNCTION           **",/ USING 20
  WRITE AS " ''*****************************",/ USING 20
  WRITE AS " ROUTINE DERIV(W, DW,TME)",/ USING 20
  WRITE AS " DEFINE W, DW AS 1-DIM ARRAY",/ USING 20
  WRITE AS " DEFINE L,B,K AS INTEGER VARIABLES",/ USING 20
  WRITE AS " DEFINE TME AS VARIABLE",/ USING 20
  LET FFLAG = 0
  UNTIL FFLAG NE 0
   DO
     READ LNTEXT AS T 80,/
 CALL SKIPCOMMENTS
''-----------------------------------
''  EXTERNAL INPUT STATEMENT
''       PROCESSING
''-----------------------------------
     LET I = MATCH.F(LNTEXT,"INPUT",0)
     IF I NE 0
       ADD 6 TO I
       CALL RETRIEVE.ROUTINE GIVEN EXINPUT(*)
```

```
      CALL WRITE.ROUTINE(EXINPUT(*))
      LET NEXTIN = INDEX
    ALWAYS
    LET I = MATCH.F(LNTEXT,"CONTINUOUS MODEL",0)
    IF I NE 0
      ADD 14 TO I
      CALL RETRIEVE:ROUTINE GIVEN TAB(*)
      LET NOCM = INDEX
      LET FFLAG = 1
    ALWAYS

    LET I = MATCH.F(LNTEXT,"COMPONENT MODEL",0)
    IF I NE 0
      ADD 13 TO I
      CALL RETRIEVE.ROUTINE GIVEN TAB(*)
      LET NOCM = INDEX
      LET FFLAG = 1
    ALWAYS
   LOOP
''-----------------------------------
''
'' GET THE NECESSARY INFORMATION FROM
''       ALL THE COMPONENT MODELS
''
''-----------------------------------
    FOR M = 1 TO NOCM
    DO
      LET FFLAG = 0
```

```
UNTIL FFLAG NE 0
DO
  READ LNTEXT AS T 80,/
  LET N = MATCH.F(LNTEXT,"STATIC STRUCTURE",0)
  IF N NE 0
     LET FFLAG = 1
  ALWAYS
LOOP
LET FFLAG = 0
UNTIL FFLAG NE 0
DO
READ LNTEXT AS T 80,/
   CALL SKIPCOMMENTS
LET I = MATCH.F(LNTEXT,"END DYNAMIC",0)
IF I NE 0
      LET FFLAG = 1
       LEAVE
ALWAYS
LET I = MATCH.F(LNTEXT,"INPUT",0)
IF I NE 0
      ADD 6 TO I
      CALL RETRIEVE.ROUTINE GIVEN INPUT(*)
      CALL WRITE.ROUTINE(INPUT(*))
ALWAYS
''-------------------------------------
''   STATE VARIABLES PROCESSING
''-------------------------------------
```

```
LET I = MATCH.F(LNTEXT,"STATE",0)
IF I NE 0
      ADD 6 TO I
      LET STCODE = 1
      CALL RETRIEVE.ROUTINE GIVEN STATE(*)
      FOR I = 1 TO INDEX
      DO
        LET MSTATE(M,I) = STATE(I)
      LOOP
      FOR J = 1 TO ARRAY.INDEX
      DO
        LET MSTATE(M,I) = STATE2(J)
        LET I = I + 1
      LOOP
      LET STN(M) = INDEX + ARRAY.INDEX
      LET STCODE = 0
      CALL WRITE.ROUTINE(STATE(*))
  ALWAYS
''-------------------------------------
''  OUTPUT FUNCTIONS PROCESSING
''-------------------------------------
   LET I = MATCH.F(LNTEXT,"OUTPUT FUNCTION",0)
   IF I NE 0
        CALL OUTPUT.FUNCTION
    ALWAYS
   LET I = MATCH.F(LNTEXT,"OUTPUT",0)
   LET J = MATCH.F(LNTEXT,"FUNCTION",0)
```

```
      IF I NE 0 AND J EQ 0
            ADD 6 TO I
            CALL RETRIEVE.ROUTINE(OUTPUT(*))
            CALL WRITE.ROUTINE(OUTPUT(*))
      ALWAYS
''-----------------------------------------
''  AUXILIARY VARIABLES AND PARAMETERS CHECKING
''-------------------------------------
      LET I = MATCH.F(LNTEXT,"AUXILIARY VARIABLE",0)
      IF I NE 0
            ADD 19 TO I
            CALL RETRIEVE.ROUTINE(AUXY(*))
            CALL WRITE.ROUTINE(AUXY(*))
      ALWAYS
      LET I = MATCH.F(LNTEXT,"AUXILIARY PARAMETER",0)
      IF I NE 0
            ADD 20 TO I
            CALL RETRIEVE.ROUTINE(AUXY(*))
            CALL WRITE.ROUTINE(AUXY(*))
            LET FLAG2 = 0
            WHILE FLAG2=0
            DO
              READ LNTEXT AS T 80,/
              CALL SKIPCOMMENTS
              LET I = MATCH.F(LNTEXT,"END AUXILIARY",0)
              IF I EQ 0
                LET N = MATCH.F(LNTEXT,SEMICOLON,0)
```

```
            IF N NE 0
             LET SUBSTR.F(LNTEXT,N,1) = BLANK
            ALWAYS
            ADD 1 TO NOAV
            LET EXAUXY(NOAV) = LNTEXT
          ELSE LET FLAG2 = 1
              LET LNTEXT = SUBSTR.F(LNTEXT,I,14)
        ALWAYS
      LOOP
  ALWAYS
  LET I = MATCH.F(LNTEXT,"PARAMETER",0)
  IF I NE 0
      ADD 10 TO I
      CALL RETRIEVE.ROUTINE(PARAMETER(*))
      CALL WRITE.ROUTINE(PARAMETER(*))
  ALWAYS
''-----------------------------------------
'' TABULAR FUNCTIONS AND INTERPOLATIONS
''          HANDLING
''-----------------------------------------
  LET I = MATCH.F(LNTEXT,"TABULAR FUNCTION",0)
  IF I NE 0
      ADD 17 TO I
      CALL RETRIEVE.ROUTINE(TABULAR(*))
      LET NOTF = INDEX
IF INDEX NE 0
WRITE AS " DEFINE " USING 21
```

```
ALWAYS
FOR I = 1 TO INDEX
DO
   IF I EQ INDEX
      WRITE TABULAR(I) AS T *," AS 2-DIM ARRAY",/ USING 21
   ELSE
      WRITE TABULAR(I) AS T *,", "  USING 21
   ALWAYS
LOOP
   ALWAYS
   LET I = MATCH.F(LNTEXT,"INTERPOLATION",0)
   IF I NE 0
         ADD 14 TO I
         CALL RETRIEVE.ROUTINE(INTERPOL(*))
         CALL WRITE.ROUTINE(INTERPOL(*))
         LET I = 0
         WHILE I = 0
         DO
         READ LNTEXT AS T 80,/
             CALL SKIPCOMMENTS
         LET I = MATCH.F(LNTEXT,"END INTERPOLATION",0)
         LET J = MATCH.F(LNTEXT,EQUALSIGN,0)
           IF J NE 0
             ADD 1 TO NOINT
             LET EXINTERPOL(NOINT) = LNTEXT
           ALWAYS
         LOOP
```

```
    ALWAYS
    LET I = MATCH.F(LNTEXT,"DYNAMIC STRUCTURE",0)
    IF I NE 0
         READ LNTEXT AS T 80,/
         CALL SKIPCOMMENTS
         CALL DYNAMIC.SECTION
    ALWAYS
LOOP
LOOP
USE 23 FOR OUTPUT
 FOR I = 1 TO NODF
 DO
 WRITE DYN.SECT(I) AS " ",T *,/
 LOOP
 WRITE AS " FOR K = 1 TO TNOSV2",/,
" DO",/,"  LET DW(K) = W(K)",/,
" LOOP",/
  LET FFLAG = 0
   UNTIL FFLAG NE 0
   DO
    READ LNTEXT AS T 80,/
              CALL SKIPCOMMENTS
    LET I = MATCH.F(LNTEXT,"PARAMETER SET",0)
    IF I NE 0
       LET FFLAG = 1
    ALWAYS
    LOOP
```

```
    CALL MAIN.PORTION
LET FFLAG = 0
 UNTIL FFLAG NE 0
 DO
  READ LNTEXT AS T 80,/
  CALL SKIPCOMMENTS
  LET I = MATCH.F(LNTEXT,"GLOBAL",0)
  IF I NE 0
     LET FFLAG = 1
  ALWAYS
  LOOP
  IF I NE 0
     CALL GLOBAL
     READ LNTEXT AS T 80,/
     CALL SKIPCOMMENTS
  ALWAYS
  LET BFLAG  = 0
  UNTIL BFLAG NE 0
  DO
  LET I      = MATCH.F(LNTEXT,"MODEL",0)
   IF I NE 0
     LET BFLAG = 1
   ALWAYS
     READ LNTEXT AS T 80,/
     CALL SKIPCOMMENTS
  LOOP
  IF BFLAG NE 0
```

```
        CALL INITIALIZE
      READ LNTEXT AS T 80,/
      CALL SKIPCOMMENTS
    ALWAYS
''------------------------------
'' PROCESSING OF OUTPUT MODULE
''------------------------------
 LET FLAG3 = 0
 UNTIL FLAG3 NE 0
 DO
   READ LNTEXT AS T 80,/
   CALL SKIPCOMMENTS
   LET I = MATCH.F(LNTEXT,"OUTPUT MODULE",0)
   IF I NE 0
     READ LNTEXT AS T 80,/
     CALL SKIPCOMMENTS
     LET I = MATCH.F(LNTEXT,"PRINT",0)
     IF I NE 0
       WRITE LNTEXT AS T * USING THE BUFFER
       READ SPACE, NUMOFLINES  USING THE BUFFER
       LET LTET = " USE 86 FOR OUTPUT"
       WRITE LTET AS T *,/ USING 21
       LET LTET2 = " LINES AS FOLLOWS"
       WRITE SPACE, NUMOFLINES, LTET2 AS T *,S 1,I 2,T *,/ USING 21
       FOR I = 1 TO NUMOFLINES
       DO
         READ LNTEXT AS T 80,/
```

```
            WRITE LNTEXT AS T *,/ USING 21
          LOOP
        LET FLAG3 = 1
        ALWAYS
      ALWAYS
    LOOP
''--------------------------------
''  PROCESS SUB-HEADINGS FOR OUTPUT
''--------------------------------

USE 87 FOR OUTPUT
 WRITE  AS "  TIME      "
FOR I = 1 TO NOOS
DO
 IF I = NOOS
 WRITE SSAVE(I)  AS T *,/
  ELSE
   WRITE SSAVE(I) AS T *
   IF LENGTH.F(SSAVE(I)) LT 12
      LET SKP = 12 - LENGTH.F(SSAVE(I))
   ELSE
     LET SKP = 1
   ALWAYS
  WRITE AS " ",S SKP
ALWAYS
LOOP
USE 22 FOR OUTPUT
WRITE AS " WRITE TIME,"
```

```
FOR I = 1 TO NOOS
DO
 IF I = NOOS
WRITE SSAVE(I),NOOS AS T *," AS D(8,2),S 3,",I 2," D(12,6),/ USING 88",/
  ELSE
   WRITE SSAVE(I) AS T *,","
ALWAYS
LOOP
WRITE AS " LOOP ",/
WRITE AS " END ''MAIN",/
WRITE TNOSV AS " RESERVE W, DW AS ",I 3,/ USING 20
USE 20 FOR OUTPUT
FOR I = 1 TO TNOSV DO
LET BEG = BLANK
LET END = BLANK
LET VAR = BLANK
CALL CHECK.ARRAYS(STATE.VECTOR(I))
IF BEG NE BLANK
  WRITE BEG,END,VAR AS
  " FOR I = ",T *," TO ",T *," DO",/,
  " LET W(I) = ",T *,"(I)",/,
  " LOOP",/
ELSE
IF STATE.VECTOR(I) NE BLANK
WRITE STATE.VECTOR(I),I AS " LET ",T *," = W(",I 2,")",/
ALWAYS
ALWAYS
```

```
 LOOP
  WRITE AS " END ''DERIV ",/ USING 23
END

''**********************************
''* THIS ROUTINE RETRIEVES ALL     *
''*   VARIABLES FROM THE MODEL     *
''**********************************
  ROUTINE RETRIEVE.ROUTINE(MAINVAR)
    DEFINE IN.OK  AS TEXT VARIABLES
    DEFINE MAINVAR AS 1-DIM TEXT ARRAY
    RESERVE MAINVAR AS 10
    LET INDEX = 1
    LET ARRAY.INDEX = 0
    LET IN.OK = "TRUE"
    WHILE IN.OK = "TRUE"
    DO
      LET J = MATCH.F(LNTEXT,COMMA,I)
      IF J NE 0
           LET VAR = SUBSTR.F(LNTEXT,I,J-I)
           CALL SKIP.BLANKS(MAINVAR(*))
           ADD 1 TO INDEX
           LET I = J + 1
      ELSE
           LET J = MATCH.F(LNTEXT,SEMICOLON,I)
           IF J NE 0
                LET VAR = SUBSTR.F(LNTEXT,I,J-I)
                CALL SKIP.BLANKS(MAINVAR(*))
```

```
            LET IN.OK = "FALSE"
        ELSE
            READ LNTEXT AS T 80,/
            CALL SKIPCOMMENTS
            LET I = 1
        ALWAYS
    ALWAYS
  LOOP
  RETURN
END

''*****************************
''* SKIP-BLANKS               *
''*        ROUTINE            *
''*****************************
ROUTINE SKIP.BLANKS(ANYVAR)
 DEFINE ANYVAR AS 1-DIM TEXT ARRAY
 DEFINE CHR,FOUND,TEMPVAR AS TEXT VARIABLES
 DEFINE K,K2,LOWER,UPPER AS INTEGER VARIABLES
 RESERVE ANYVAR AS 10
  LET ANYVAR(INDEX) = BLANK
 FOR K = 1 TO LENGTH.F(VAR)
 DO
  LET CHR = SUBSTR.F(VAR,K,1)
  IF CHR NE BLANK
  LET ANYVAR(INDEX) = CONCAT.F(ANYVAR(INDEX),CHR)
  ALWAYS
 LOOP
```

```
IF STCODE = 1
  ADD 1 TO TNOSV
  LET STATE.VECTOR(TNOSV) = ANYVAR(INDEX)
  LET BEG = BLANK
  LET END = BLANK
  CALL CHECK.ARRAYS(ANYVAR(INDEX))
  IF BEG NE BLANK
    WRITE BEG,END AS T *,S,1,T * USING THE BUFFER
    READ LOWER,UPPER AS I 2,I 2 USING THE BUFFER
    LET TNOSV = TNOSV + (UPPER - LOWER)
  ALWAYS
ALWAYS
LET K2 = MATCH.F(ANYVAR(INDEX),OPBRACKET,0)
IF K2 NE 0
  LET TEMPVAR = SUBSTR.F(ANYVAR(INDEX),1,K2-1)
ELSE
  LET TEMPVAR = ANYVAR(INDEX)
ALWAYS
LET K = 1
 LET FOUND = "NO"
 UNTIL FOUND = "YES" OR K > P
 DO
   IF TEMPVAR = VAR.LIST(K)
      LET FOUND = "YES"
      IF SAFE EQ 0
      LET INDEX = INDEX - 1
      ALWAYS
```

```
    ELSE
       ADD 1 TO K
    ALWAYS
  LOOP
  IF FOUND = "NO"
  IF K2 NE 0
    ADD 1 TO  ARRAY.INDEX
    LET ARRAY(ARRAY.INDEX) = SUBSTR.F(ANYVAR(INDEX),1,K2-1)
    ADD 1 TO  ARRAY2.INDEX
    LET R.ARRAY(ARRAY2.INDEX) = ARRAY(ARRAY.INDEX)
    LET STATE2(ARRAY.INDEX) = ANYVAR(INDEX)
     ADD 1 TO P
     LET VAR.LIST(P) = SUBSTR.F(ANYVAR(INDEX),1,K2-1)
     LET INDEX = INDEX - 1
   ELSE
     ADD 1 TO P
     LET VAR.LIST(P) = ANYVAR(INDEX)
  ALWAYS
  ALWAYS
RETURN
END

''*******************************
''* SKIPCOMMENTS                *
''*         ROUTINE             *
''*******************************
ROUTINE SKIPCOMMENTS
 DEFINE BCFLAG,ECFLAG AS INTEGER VARIABLES
```

```
LET BCFLAG = MATCH.F(LNTEXT,"(*",0)
LET ECFLAG = MATCH.F(LNTEXT,"*)",0)
IF BCFLAG NE 0
     IF ECFLAG = 0
        LET SUBSTR.F(LNTEXT,BCFLAG,LENGTH.F(LNTEXT)) = BLANK
        LET XBCFLAG = 1
     ELSE
        LET SUBSTR.F(LNTEXT,BCFLAG,LENGTH.F(LNTEXT)) = BLANK
        LET XBCFLAG = 0
     ALWAYS
ELSE
     IF XBCFLAG = 1 AND ECFLAG = 0
       LET SUBSTR.F(LNTEXT,1,LENGTH.F(LNTEXT)) = BLANK
     ALWAYS

     IF XBCFLAG = 1 AND ECFLAG NE 0
       LET SUBSTR.F(LNTEXT,1,LENGTH.F(LNTEXT)) = BLANK
       LET XBCFLAG = 0
     ALWAYS
ALWAYS
IF LNTEXT = BLANK
  READ LNTEXT AS T 80,/
  CALL SKIPCOMMENTS
ALWAYS
RETURN
END

''**************************
```

```
''*        WRITE          *
''*         ROUTINE       *
''*************************
  ROUTINE WRITE.ROUTINE(VARABLE)
  DEFINE VARABLE AS 1-DIM TEXT ARRAY
  RESERVE VARABLE AS 10
    IF INDEX NE 0
    WRITE AS " DEFINE " USING 21
    ALWAYS
    FOR I = 1 TO INDEX
     DO
       IF I EQ INDEX
          WRITE VARABLE(I) AS T *," AS VARIABLES",/ USING 21
       ELSE
          WRITE VARABLE(I) AS T *,", "  USING 21
       ALWAYS
   LOOP
    IF ARRAY.INDEX NE 0
    WRITE AS " DEFINE " USING 21
    ALWAYS
    FOR I = 1 TO ARRAY.INDEX
     DO
       IF I EQ ARRAY.INDEX
          WRITE ARRAY(I) AS T *," AS 1-DIM ARRAY",/ USING 21
       ELSE
          WRITE ARRAY(I) AS T *,", "  USING 21
       ALWAYS
```

```
   LOOP
   RETURN
   END

''*****************************
''*   DYNAMIC.SECTION      *
''*         ROUTINE        *
''*****************************
  ROUTINE DYNAMIC.SECTION
   DEFINE FLAG AS INTEGER VARIABLES
   LET FLAG = 0
   WHILE FLAG = 0
   DO
     READ LNTEXT AS T 80,/
     CALL SKIPCOMMENTS
     LET J = MATCH.F(LNTEXT,"END DERIVATIVE",0)
     IF J NE 0
        LET FLAG = 1
     ELSE
        ADD 1 TO NODF
        LET DYN.SECT(NODF) = LNTEXT
    ALWAYS
    LOOP
   CALL SORT(DYN.SECT(*),NODF)
      FOR I = 1 TO NODF
      DO
        LET N = MATCH.F(DYN.SECT(I),SEMICOLON,0)
        IF N NE 0
```

```
        LET SUBSTR.F(DYN.SECT(I),N,1) = BLANK
      ALWAYS
      LET J = MATCH.F(DYN.SECT(I),QUOTE,0)
      IF J NE 0
         LET DYN.SECT(I) = SUBSTR.F(DYN.SECT(I),J+1,80)
  WRITE F,DYN.SECT(I) AS " W(",I 2,")",T * USING THE BUFFER
       READ DYN.SECT(I) AS T 80 USING THE BUFFER
         ADD 1 TO F
      ALWAYS
  LOOP
RETURN
END

''**************************
''*   OUTPUT.FUNCTION      *
''*         ROUTINE        *
''**************************
 ROUTINE OUTPUT.FUNCTION
  DEFINE FLAG AS INTEGER VARIABLE
  LET FLAG = 0
  WHILE FLAG = 0
  DO
    READ LNTEXT AS T 80,/
    CALL SKIPCOMMENTS
    LET J = MATCH.F(LNTEXT,"END OUTPUT FUNCTION",0)
    IF J NE 0
      LET FLAG = 1
    ELSE
```

```
      LET N = MATCH.F(LNTEXT,SEMICOLON,0)
      IF N NE 0
        LET SUBSTR.F(LNTEXT,N,1) = BLANK
      ALWAYS
      ADD 1 TO NOOF
      LET OUT.FUNC(NOOF) = LNTEXT
    ALWAYS
  LOOP
 CALL SORT(OUT.FUNC(*),NOOF)
 RETURN
 END

''**************************
''*    EQUIVALENCING       *
''*         ROUTINE        *
''**************************
  ROUTINE EQUIVALENCING
    DEFINE J AS INTEGER VARIABLE
    LET J = 0
    UNTIL J NE 0
    DO
      START NEW RECORD
      READ LNTEXT AS T 80
      LET J = MATCH.F(LNTEXT,"ZZZZ",0)
    LOOP
  RETURN
 END
```

```
''**************************
''*   MAIN.PORTION         *
''*         ROUTINE        *
''**************************
 ROUTINE MAIN.PORTION
 DEFINE FLAG,NN,M AS INTEGER VARIABLE
USE 21 FOR OUTPUT
 WRITE AS " END ''PREAMBLE",/
WRITE AS " ''****************************",/
WRITE AS " ''*   MAIN.PORTION          **",/
WRITE AS " ''*     OF PROGRAM          **",/
WRITE AS " ''****************************",/
 WRITE AS " MAIN ",/
 WRITE TNOSV AS " RESERVE W, DW AS ",I 3,/
   IF ARRAY2.INDEX NE 0
   WRITE AS " RESERVE "
   ALWAYS
   FOR I = 1 TO ARRAY2.INDEX
    DO
      IF I EQ ARRAY2.INDEX
         WRITE R.ARRAY(I) AS T *," AS 20",/
      ELSE
         WRITE R.ARRAY(I) AS T *,", "
      ALWAYS
  LOOP
WRITE TNOSV AS " LET TNOSV = ",I 2,/
WRITE  AS " LET TNOSV2 = TNOSV",/
```

```
WRITE AS " SPECIFY THE STEP SIZE FOR THE INTEGRATION ",/ USING 6
READ H USING 5
WRITE H AS " LET H = ",D(3,2),/
IF NEXTIN NE 0
   WRITE AS " SPECIFY VALUES FOR EXTERNAL VARIABLES",/ USING 6
   WRITE AS " ",/ USING 6
ALWAYS

FOR I = 1 TO NEXTIN DO
 WRITE EXINPUT(I) AS T *,/ USING 6
 READ VALUE USING 5
 WRITE EXINPUT(I),VALUE AS " LET ",T *,"= ",D(6,4),/
LOOP
LET FLAG = 0
WHILE FLAG = 0
DO
  READ LNTEXT AS T 80,/
  CALL SKIPCOMMENTS
  LET J = MATCH.F(LNTEXT,"END PARAMETER SET",0)
  IF J NE 0
     LET FLAG = 1
  ELSE
  LET N = MATCH.F(LNTEXT,EQUALSIGN,0)
    IF N NE 0
      LET FLAG2 = 0
      CALL DATA.STATEMENT(N)
      IF FLAG2 EQ 0        '' THIS IS NOT DATA STATEMENT
      LET N = MATCH.F(LNTEXT,SEMICOLON,0)
```

```
 IF N NE 0
   LET SUBSTR.F(LNTEXT,N,1) = BLANK
 ALWAYS
WRITE LNTEXT AS " LET ",T 80,/
ALWAYS
ELSE
  LET N = MATCH.F(LNTEXT,"TABULAR FUNCTION",0)
  IF N NE 0
  WRITE LNTEXT AS T * USING THE BUFFER
  READ SPACE,SPACE,FUNCNAME,SPACE,NOPOINTS USING THE BUFFER
  LET N = 0
  WHILE N = 0
  DO
    READ LNTEXT AS T 80,/
    CALL SKIPCOMMENTS
    FOR M = 1 TO 80
    DO
      LET TAB(M) = SUBSTR.F(LNTEXT,M,1)
   LOOP
   LET NN=  MATCH.F(LNTEXT,"END",0)
   IF NN NE 0
WRITE FUNCNAME,NOPOINTS AS " RESERVE ",T *," AS ",I 2," BY 2",/
   USING 21
   WRITE FUNCNAME AS " READ ",T *," USING 24",/ USING 21
   ADD 1 TO NUMBER
   LET POINTS(NUMBER) = NOPOINTS
   LET N = 1
```

```
        ELSE
          FOR M = 1 TO 80
          DO
            IF TAB(M) = "(" OR TAB(M) = ")" OR TAB(M) = ","
               LET SUBSTR.F(LNTEXT,M,1) = BLANK
            ALWAYS
        LOOP
        WRITE LNTEXT AS T 80,/ USING 24
       ALWAYS
      LOOP
        ALWAYS
       ALWAYS
      ALWAYS
    LOOP
    RETURN
    END

''**************************
''*   GLOBAL               *
''*         ROUTINE        *
''**************************
    ROUTINE GLOBAL
   DEFINE FLAG, SKP AS INTEGER VARIABLES
   LET BUFFER.V = 80
   LET FLAG = 0
   UNTIL FLAG NE 0
   DO
    READ LNTEXT AS T 80,/
```

```
 CALL SKIPCOMMENTS
 LET FLAG = MATCH.F(LNTEXT,"END GLOBAL",0)
LET N = MATCH.F(LNTEXT,"SIMULATE",0)
IF N NE 0
  LET LNTEXT = SUBSTR.F(LNTEXT,N+8,BUFFER.V)
  LET SLNTEXT= LNTEXT
  LET N = MATCH.F(SLNTEXT,SEMICOLON,0)
  IF N NE 0
    LET SUBSTR.F(SLNTEXT,N,1)= BLANK
  ALWAYS
ALWAYS
LET N = MATCH.F(LNTEXT,"START",0)
IF N NE 0
  ADD 6 TO N
 LET LNTEXT = SUBSTR.F(LNTEXT,N,LENGTH.F(LNTEXT))
  LET N = MATCH.F(LNTEXT,SEMICOLON,0)
  IF N NE 0
    LET SUBSTR.F(LNTEXT,N,1)= BLANK
  ALWAYS
  WRITE LNTEXT AS " LET ", T * ,/ USING 22
ALWAYS
  LET N = MATCH.F(LNTEXT,"COMMUNICATE",0)
   IF N NE 0
       LET STEP = SUBSTR.F(LNTEXT,N+21,5)
    ALWAYS
LOOP
USE 22 FOR OUTPUT
```

```
FOR I = 1 TO TNOSV DO
LET BEG = BLANK
LET END = BLANK
LET VAR = BLANK
CALL CHECK.ARRAYS(STATE.VECTOR(I))
IF BEG NE BLANK
  WRITE BEG,END,VAR AS
  " FOR I = ",T *," TO ",T *," DO",/,
  " LET W(I) = ",T *,"(I)",/,
  " LOOP",/
ELSE
  IF STATE.VECTOR(I) NE BLANK
  WRITE I,STATE.VECTOR(I) AS " LET  W(",I 2,")  = ",T *,/
  ALWAYS
ALWAYS
LOOP
FOR I = 1 TO NOOF
DO
  WRITE OUT.FUNC(I) AS " LET ",T 80,/
 LOOP
FOR I = 1 TO NOAV
DO
  WRITE EXAUXY(I) AS T *,/
 LOOP
WRITE SLNTEXT AS T *,/
WRITE AS " DO ",/
FOR I = 1 TO NOINT
```

```
DO
  FOR M = 1 TO LENGTH.F(EXINTERPOL(I))
   DO
     LET CHR = SUBSTR.F(EXINTERPOL(I),M,1)
     IF CHR = EQUALSIGN OR CHR = OPBRACKET OR CHR = CLOSBRACKET
        LET SUBSTR.F(EXINTERPOL(I),M,1) = BLANK
     ALWAYS
   LOOP
WRITE EXINTERPOL(I) AS T * USING THE BUFFER
READ BEQUAL USING THE BUFFER
CALL CHECK.INTERPOLATION(POINTS(I))
LOOP
WRITE AS " CALL RUNGEKUTTA GIVEN W(*), DW(*), TIME,TNOSV",/
FOR I = 1 TO NOOF
DO
  WRITE OUT.FUNC(I) AS " LET ",T 80,/
 LOOP
 RETURN
 END

''**************************
''*   INITIALIZE           *
''*         ROUTINE        *
''**************************
 ROUTINE INITIALIZE
 DEFINE T,J,JJ,I,FLAG,N AS INTEGER VARIABLE
 FOR T = 1 TO NOCM
 DO
```

```
LET FLAG = 0
UNTIL FLAG NE 0
DO
  LET FLAG = MATCH.F(LNTEXT,"END MODEL",0)
  LET J =MATCH.F(LNTEXT,"INITIALIZE",0)
  LET JJ=MATCH.F(LNTEXT,"ZERO",0)
  IF J NE 0 AND JJ NE 0
    FOR I = 1 TO STN(T)
    DO
      WRITE MSTATE(T,I) AS " LET ",T *," = 0.0",/ USING 21
    LOOP
 ELSE
   IF J NE 0 AND JJ EQ 0
    UNTIL J = 0
    DO
    READ LNTEXT AS T 80,/
    CALL SKIPCOMMENTS
    LET J = MATCH.F(LNTEXT,EQUALSIGN,0)
    IF J NE 0
     LET FLAG2 = 0
     CALL DATA.STATEMENT(J)
     IF FLAG2 = 0          '' ASSIGNMENT STATEMENT
     LET N = MATCH.F(LNTEXT,SEMICOLON,0)
     IF N NE 0
       LET SUBSTR.F(LNTEXT,N,1) = BLANK
     ALWAYS
     WRITE LNTEXT AS " LET ", T *,/ USING 21
```

```
      ALWAYS
     ALWAYS
     LOOP
    ALWAYS
   ALWAYS
  LET J = MATCH.F(LNTEXT,"COMMUNICATE",0)
   IF J NE 0
       LET STEP = SUBSTR.F(LNTEXT,J+21,5)
    ALWAYS
     LET J = MATCH.F(LNTEXT,"SAVE",0)
      IF J NE 0
        LET I = MATCH.F(LNTEXT,"AT EVERY",0)
         IF I NE 0
          LET LNTEXT = SUBSTR.F(LNTEXT,1,I-1)
          LET LNTEXT = CONCAT.F(LNTEXT,SEMICOLON)
         ALWAYS
         CALL SAVE.OUTPUT
       ALWAYS
  READ LNTEXT AS T 80,/
  CALL SKIPCOMMENTS
     LOOP
  LOOP
IF STEP = BLANK
   LET STEP = "1.0"
ALWAYS
WRITE STEP AS " ADD ",T *," TO TIME",/ USING 22
RETURN
```

```
END

''************************************
''*     SAVE.OUTPUT                  *
''*          ROUTINE                 *
''************************************
ROUTINE SAVE.OUTPUT
  LET I = MATCH.F(LNTEXT,"SAVE",0)
  IF I NE 0
   ADD 4 TO I
   LET SAFE = 1
   CALL RETRIEVE.ROUTINE(SAVE(*))
   FOR J = 1 TO INDEX
   DO
     ADD 1 TO NOOS
     LET SSAVE(NOOS) = SAVE(J)
   LOOP
   ALWAYS
   RETURN
  END

''************************
''*                      *
''* SORT PROCEDURE       *
''*                      *
''************************
ROUTINE SORT(INTEXT,N)
 DEFINE N,I,J,W  AS INTEGER VARIABLES
```

```
DEFINE LHS,RHS,INTEXT AS 1-DIM TEXT ARRAYS
DEFINE CHR,TEMPTEXT AS TEXT VARIABLES
RESERVE INTEXT AS 80
RESERVE LHS,RHS AS 20
FOR I = 1 TO N
 DO
   LET J = MATCH.F(INTEXT(I),EQUALSIGN,0)
   IF J NE 0
    LET LHS(I) = AMPERSAND
    LET TEMPTEXT = BLANK
    LET TEMPTEXT=CONCAT.F(TEMPTEXT,SUBSTR.F(INTEXT(I),1,J-1))
    FOR W = 1 TO LENGTH.F(TEMPTEXT)
     DO
       LET CHR=SUBSTR.F(TEMPTEXT,W,1)
       IF CHR = OPBRACKET OR CHR = CLOSBRACKET
       LET CHR = AMPERSAND
       ALWAYS
       IF CHR NE BLANK
        LET LHS(I)=CONCAT.F(LHS(I),CHR)
       ALWAYS
     LOOP
     LET LHS(I)=CONCAT.F(LHS(I),AMPERSAND)
    LET RHS(I) = AMPERSAND
    LET TEMPTEXT = BLANK
  LET W = LENGTH.F(INTEXT(I))
  LET TEMPTEXT=CONCAT.F(TEMPTEXT,SUBSTR.F(INTEXT(I),J+1,W))
    FOR W = 1 TO LENGTH.F(TEMPTEXT)
```

```
      DO
        LET CHR=SUBSTR.F(TEMPTEXT,W,1)
        IF CHR = "(" OR CHR = ")" OR CHR="+" OR CHR="*" OR CHR="-"
         OR CHR="/" OR CHR=";" OR CHR=" "
        LET CHR = AMPERSAND
        ALWAYS
         LET RHS(I)=CONCAT.F(RHS(I),CHR)
      LOOP
         LET RHS(I)=CONCAT.F(RHS(I),AMPERSAND)
    ALWAYS
    LOOP
LET FLAG = 1
WHILE FLAG = 1 DO
  LET FLAG = 0
   FOR I = 2 TO N
   DO
     FOR W = 1 TO I-1 DO
      LET J= MATCH.F(RHS(W),LHS(I),0)
      IF J NE 0
       LET TEMPTEXT = INTEXT(I)
       LET INTEXT(I)=INTEXT(W)
       LET INTEXT(W)=TEMPTEXT

       LET TEMPTEXT = LHS(I)
       LET LHS(I)=LHS(W)
       LET LHS(W)=TEMPTEXT

       LET TEMPTEXT = RHS(I)
```

```
        LET RHS(I)=RHS(W)
        LET RHS(W)=TEMPTEXT
        LET FLAG = 1
       ALWAYS
    LOOP
  LOOP
 LOOP
RETURN
END

''**********************************
''*    DATA.STATEMENT              *
''*          ROUTINE               *
''**********************************
 ROUTINE DATA.STATEMENT(N)
 DEFINE N,STAR AS INTEGER VARIABLES
 DEFINE LHSOFEX,RHSOFEX,EXPRESION,TERMVAL AS TEXT VARIABLES
  LET LHSOFEX = SUBSTR.F(LNTEXT,1,N-1)
 LET I = MATCH.F(LHSOFEX,OPBRACKET,0)
   IF I NE 0
     LET FLAG2= 1
     LET LHSOFEX = SUBSTR.F(LHSOFEX,1,I-1)
     LET RHSOFEX = SUBSTR.F(LNTEXT,N+2,LENGTH.F(LNTEXT))
     LET LNTEXT = RHSOFEX
     LET I = 1
     CALL RETRIEVE.ROUTINE(AUXY(*))
     WRITE AS " LET J = 1 ",/ USING 21
     FOR I = 1 TO INDEX
```

```
    DO
      LET STAR = MATCH.F(AUXY(I),"*",0)
      IF STAR NE 0
        LET TERMVAL = SUBSTR.F(AUXY(I),1,STAR-1)
        LET EXPRESION = SUBSTR.F(AUXY(I),STAR+1,LENGTH.F(RHSOFEX))
        WRITE TERMVAL AS " FOR I = 1 TO ",T *," DO",/ USING 21
        WRITE LHSOFEX,EXPRESION AS " LET ",T *,"(J)= ",T *,/
         USING 21
        WRITE AS " ADD 1 TO J",/ USING 21
        WRITE AS "LOOP",/ USING 21
      ELSE
        WRITE LHSOFEX,AUXY(I) AS " LET ",T *,"(J) = ",T *,/
         USING 21
        IF I NE INDEX
          WRITE AS " ADD 1 TO J",/ USING 21
        ALWAYS
      ALWAYS
    LOOP
    ALWAYS
  RETURN
  END

''************************************
''*     CHECK.ARRAYS                 *
''*          ROUTINE                 *
''************************************
ROUTINE CHECK.ARRAYS(STATEVECT)
DEFINE STATEVECT AS TEXT VARIABLES
```

```
DEFINE N,NN,NNN AS INTEGER VARIABLES
LET N = MATCH.F(STATEVECT,OPBRACKET,0)
IF N NE 0
  LET VAR = SUBSTR.F(STATEVECT,1,N-1)
  LET NN = MATCH.F(STATEVECT,TWODOTS,0)
  LET NNN =MATCH.F(STATEVECT,CLOSBRACKET,0)
  LET BEG = SUBSTR.F(STATEVECT,N+1,NN-(N+1))
  LET END = SUBSTR.F(STATEVECT,NN+2,NNN-(NN+2))
ALWAYS
RETURN
END

''*******************************
''*  CHECK INTERPOLATION        *
''*         ROUTINE             *
''*******************************
ROUTINE CHECK.INTERPOLATION(POINTS)
DEFINE FLAG,POINTS,L AS INTEGER VARIABLE
LET FLAG = 0
LET BEQUAL = CONCAT.F(BLANK,BEQUAL)
FOR L = 1 TO ARRAY2.INDEX
 DO
  IF BEQUAL = R.ARRAY(L)
     LET FLAG = 1
     READ SPACE, AFTEQUAL,PARENT USING THE BUFFER
     LET BEQUAL = CONCAT.F(BEQUAL,"(I)")
     LET PARENT = CONCAT.F(PARENT,"(I)")
     WRITE BEQUAL,POINTS,PARENT,AFTEQUAL AS
```

```
    " FOR I = 1 TO 20",/," DO",/,
    " LET ",T *,"= INTERPOLATE (",I 2,",",T *,",",T *,"(*,*))",/,
    " LOOP",/
    LEAVE
   ALWAYS
  LOOP
  IF FLAG = 0
     READ AFTEQUAL,PARENT USING THE BUFFER
     WRITE BEQUAL,POINTS,PARENT,AFTEQUAL AS
     " LET ",T *,"= INTERPOLATE (",I 2,",",T *,",",T *,"(*,*))",/
  ALWAYS
RETURN
END
```

## Appendix B

## EXEC FILE

```
&CONTROL OFF
GLOBAL TXTLIB PASCAL
-STARTO
&BEGTYPE
          WELCOME TO GEST SOFTWARE SYSTEM
        THIS SYSTEM PROVIDES FACILITIES FOR

      0   GEST TEMPLATE MODEL SPECIFICATION
      1   INTERACTIVE SPECIFICATION OF GEST MODEL
      2   CERTIFICATION OF A MODEL
      3   TRANSLATION OF THE SPECIFIED MODEL
      4   EXECUTION OF A TRANSLATED MODEL
      5   LIST MODELS IN THE MODEL BASE

    ENTER THE FACILITY YOU WANT, PLEASE
&END
&READ VARS &ANS
&IF &ANS = 0 &GOTO -TEMPLATE
&IF &ANS = 1 &GOTO -INTSP
&IF &ANS = 2 &GOTO -CERT
&IF &ANS = 3 &GOTO -TRANS
&IF &ANS = 4 &GOTO -EXEC1
&IF &ANS = 5 &GOTO -LIST
```

```
&IF &ANS = QQ &EXIT
&TYPE INVALID RESPONSE, ENTER 'QQ' TO QUIT ANYWAY
&TYPE
&GOTO -STARTO
-TEMPLATE
EXEC SIMSCRPT TEMPLATE
X FILE SIMU44
&GOTO -STARTO
-INTSP
EXEC PASCAL SEM12
LOAD SEM12(START
&IF &RETCODE EQ 0 &GOTO -SEMA
&EXIT
-SEMA
&TYPE WOULD YOU LIKE TO INSERT COMMENTS IN THE PROGRAM? (Y/N)
&READ VARS &ANS
&IF &ANS = Y CP X FILE OUT
&TYPE DO YOU WANT TO CERTIFY THE MODEL  (Y/N)
&READ VARS &ANS
&IF &ANS = Y &GOTO -CERT
&EXIT 1
-LIST
L * MODEL
&TYPE
&TYPE
&TYPE
&TYPE ENTER 'R' TO RETURN TO MAIN MENU OR ANYTHING TO STOP
```

```
&READ VARS &ANS
&IF &ANS = R &GOTO -STARTO
&EXIT 4
-CERT
&TYPE CERTIFICATION IS ABOUT TO BEGIN...
&TYPE ENTER MODEL NAME
&READ VARS &ANS
&FN = &ANS
STATE &FN GEST *
&IF &RETCODE EQ 0 &GOTO -STEP1
&TYPE &FN GEST DOES NOT EXIST
&EXIT
*
-STEP1  STATE FILE GEST01 A
&IF &RETCODE NE 0 &GOTO -STEP2
ERASE FILE GEST01 A
*
-STEP2  STATE FILE GESTERR A
&IF &RETCODE NE 0 &GOTO -COP
ERASE FILE GESTERR A
*
-COP  COPY &FN GEST A FILE GEST01 A
EXEC SIMSCRPT ( ACC
EXEC SIMCON
*
-EXEC STATE MAGEST TEXT *
&IF &RETCODE NE 0  &EXIT 999
```

```
-RERUN
EXEC SIMRUN MAGEST *
&IF &RETCODE NE 0  &GOTO -RTERR
STATE FILE GESTERR A
&IF &RETCODE NE 0  &GOTO -CERT
&TYPE -> &FN GEST NOT CERTIFIED
*
-ASK1 &TYPE ENTER 'E' TO EDIT OR 'S' TO EXIT
&READ VARS  &ANS
&IF &ANS = S  &EXIT 2
&IF &ANS = E  &GOTO -COR
&IF &ANS= . &GOTO -ASK1
&GO TO -ASK1
* RUN TIME ERROR
-RTERR  &TYPE -> RUNTIME ERROR:THE REASON MIGHT BE THE FOLLOWING:
&TYPE    -LENGTH OF RECORDS( < 73 ) OR ENTERING <CR> AS INPUT
-ASK4 &TYPE ENTER 'E' TO EDIT OR 'S' TO EXIT
&READ VARS  &ANS
&IF &ANS = S  &EXIT 999
&IF &ANS = E  &GOTO -A
&IF &ANS= . &GOTO -ASK4
&GO TO -ASK4
*
-COR  ERASE FILE GEST01 A
COPY FILE GEST02 A FILE GEST01 A
-A XEDIT FILE GEST01 A
ERASE FILE GESTERR A
```

```
*
-ASK2  &TYPE ENTER 'C' TO RERUN OR 'S' TO STOP
&READ VARS &ANS
&IF &ANS = S  &EXIT 2
&IF &ANS = C  &GOTO -RERUN
&IF &ANS =  . &GOTO -ASK2
&GOTO  -ASK2
*
-CERT &TYPE -> &FN GEST A  HAS BEEN CERTIFIED
*
-ASK3
&TYPE ENTER 'T' TO TRANSLATE OR 'S' TO STOP
&READ VARS &ANS
&IF &ANS = S  &EXIT 0
&IF &ANS = T  &GOTO -TRANS
&IF &ANS =  . &GOTO -ASK3
&GOTO  -ASK3
*
-TRANS
&IF &ANS = T &GOTO -TRANSCTN
&BEGTYPE
  ENTER MODEL NAME TO BE TRANSLATED (FT = GEST)
&END
&READ VARS &ANS
&FN = &ANS
STATE &FN GEST *
&IF &RETCODE EQ 0 &GOTO -FILETRANS
```

```
&TYPE &FN GEST DOES NOT EXIST
&EXIT
*
-FILETRANS
STATE FILE SIMU80 *
&IF &RETCODE EQ 0 ERASE FILE SIMU80
COPY &FN GEST * FILE SIMU80 A
-TRANSCTN
STATE FILE GEST02 *
&IF &RETCODE EQ 0 &GOTO -CONTN
&TYPE CERTIFICATION HAS NOT BEEN DONE
&TYPE ENTER 'C' TO CERTIFY THE PROGRAM OR ANYTHING TO QUIT
&READ VARS &ANS
&IF &ANS = C &GOTO -CERT
&EXIT
-CONTN
*ENAME FILE GEST02 A &FN GEST A(REP
*
*EXEC SIMSCRPT PROJ(COMP
-EXEC SIMSCRPT PROJ(GO
COPY FILE SIMU22 A FILE SIMU21 A(APP
COPY FILE SIMU23 A FILE SIMU20 A(APP
COPY FILE SIMU20 A FILE SIMU21 A(APP
ERASE FILE SIMU20
STATE SIM21 SIMSCRPT
&IF &RETCODE EQ 0 ERASE SIM21 SIMSCRPT
COPY TEMP SCMP B FILE SIMU21 A(APP
```

```
COPY FILE SIMU21 A SIM21 SIMSCRPT A(LR 80
&TYPE ENTER 'X' TO EXECUTE OR 'S' TO STOP
&READ VARS &ANS
&IF &ANS = X &GOTO -EXEC1
&EXIT 0
-EXEC1
&IF &ANS = 4  &GOTO -EXEC2
&IF &ANS = X &GOTO -EXEC2
&EXIT 1
-EXEC2
STATE SIM21 SIMSCRPT
&IF &RETCODE EQ 0 &GOTO -OK
&TYPE MODEL HAS NOT BEEN TRANSLATED
&TYPE YOU MAY TRY AGAIN
&TYPE
&GOTO -TRANS
-OK
STATE FILE SIMU88
&IF &RETCODE EQ 0 ERASE FILE SIMU88
EXEC SIMSCRPT SIM21(COMP
EXEC SIMSCRPT SIM21(GO
COPY FILE SIMU88 A FILE SIMU87 A(APP
COPY FILE SIMU87 A FILE SIMU86 A(APP
*ERASE FILE SIMU88
*ERASE FILE SIMU87
STATE FILE SIMU21
&IF &RETCODE EQ 0 ERASE FILE SIMU21
```

```
STATE FILE SIMU22
&IF &RETCODE EQ 0 ERASE FILE SIMU22
STATE FILE SIMU23
&IF &RETCODE EQ 0 ERASE FILE SIMU23
STATE FILE SIMU24
&IF &RETCODE EQ 0 ERASE FILE SIMU24
X FILE SIMU86
&TYPE THANK YOU FOR USING GESTSS
&TYPE ENTER 'R' TO RETURN TO MAIN MENU
&READ VARS &ANS
&IF &ANS = R &GOTO -STARTO
&EXIT 2
```

## Appendix C

## INTERACTIVE MODEL SPECIFICATION PROGRAM

```
(*
   INTERACTIVE SPECIFICATION OF GEST MODEL
                    BY
              FELIX DOGBEY
                                                  *)
PROGRAM SEM  (INPUT/,OUT,OUTPUT);
CONST
  BUFFLEN = 80;
TYPE
  BUFFER = PACKED ARRAY(.1..BUFFLEN.) OF CHAR;
  CK     =  ARRAY(.1..20.) OF BUFFER;
  INTT   =  ARRAY(.1..20.) OF INTEGER;
VAR RC : INTEGER;
   LEN,MLEN,PLEN,TBLEN,EXLEN,GFLAG,K: INTEGER;
   NUMBER : INTEGER;
   BUFF,MBUFF,PBUFF,TBBUFF,EXBUFF: BUFFER;
   CMODNAME : CK;
   PPLEN,LCMOD,INLEN: INTT;
   PARAM, INPAT : CK;
   XBUFF : CK;
   OUT  : TEXT;
PROCEDURE READINPUT(VAR BUFF : BUFFER; VAR LEN : INTEGER);
```

```
VAR I,J: INTEGER;
   BEGIN
     I := 0;
     READ(BUFF(.1.));
     IF NOT EOLN THEN
     BEGIN
       I := 1;
       WHILE ((NOT (EOLN)) AND (I < BUFFLEN)) DO
     BEGIN
       I := I + 1;
       READ(BUFF(.I.))
     END;
     LEN := I
     END
   ELSE
     LEN := 0;
(*  FOR J := 1 TO I DO
     XBUFF(.K,J.) := BUFF(.J.)
     K := K + 1                    *)

 END;
PROCEDURE CONTINUOUSMODEL (VAR BUFF : BUFFER; VAR LEN: INTEGER);
 VAR I, FLAG,S : INTEGER;
 BEGIN
     IF GFLAG = 0 THEN
     BEGIN
       WRITELN(' ENTER NAME OF COMPONENT MODEL ');
       READINPUT(BUFF,LEN)
```

```
    END;
SYSTEM('EXECSERV CLEARSCN', RC);WRITELN;WRITELN;WRITELN;
WRITE('  CONTINUOUS MODEL SPECIFICATION FOR    ');
       FOR I := 1 TO LCMOD(.K.)  DO
       WRITE(CMODNAME(.K,I.));
       WRITELN(' BEGINS.....');
WRITELN;WRITELN;WRITELN;
FLAG:= 0;
   IF LCMOD(.K.) > 0 THEN
   BEGIN
       WRITE(OUT,' CONTINUOUS MODEL');
       FOR I := 1 TO LCMOD(.K.)  DO
       WRITE(OUT,CMODNAME(.K,I.));
       WRITELN(OUT,';');
       WRITELN(OUT,'STATIC STRUCTURE')
    END;
WRITELN(' ENTER INPUT VARIABLES');
READINPUT(BUFF,LEN);
(*  STORE ALL INPUTS *)
  FOR S := 1 TO LEN DO
    INPAT(.K,S.) := BUFF(.S.);
    INLEN(.K.) := LEN;
   IF LEN > 0 THEN
   BEGIN
       WRITE(OUT,' INPUT ');
       FOR I := 1 TO LEN DO
```

```
        WRITE(OUT,BUFF(.I.));
        WRITELN(OUT,';');
    END;
WRITELN(' ENTER STATE VARIABLES');
READINPUT(BUFF,LEN);
   IF LEN > 0 THEN
   BEGIN
       WRITE(OUT,' STATE ');
       FOR I := 1 TO LEN DO
       WRITE(OUT,BUFF(.I.));
       WRITELN(OUT,';');
    END;
WRITELN(' ENTER OUTPUT VARIABLES');
READINPUT(BUFF,LEN);
   IF LEN > 0 THEN
   BEGIN
       WRITE(OUT,' OUTPUT ');
       FOR I := 1 TO LEN DO
       WRITE(OUT,BUFF(.I.));
       WRITELN(OUT,';');
    END;
WRITELN(' ENTER AUXILIARY VARIABLES');
READINPUT(BUFF,LEN);
   IF LEN > 0 THEN
   BEGIN
       WRITE(OUT,' AUXILIARY VARIABLES ');
```

```
      FOR I := 1 TO LEN DO
         WRITE(OUT,BUFF(.I.));
         WRITELN(OUT,';');
    END;

WRITELN(' ENTER PARAMETER VARIABLES');
READINPUT(PBUFF,PLEN);
(*  STORE ALL PARAMETERS *)
  FOR S := 1 TO PLEN DO
    PARAM(.K,S.) := PBUFF(.S.);
  PPLEN(.K.) := PLEN;
   IF PLEN > 0 THEN
   BEGIN
       WRITE(OUT,' PARAMETERS ');
       FOR I := 1 TO PLEN DO
       WRITE(OUT,PBUFF(.I.));
       WRITELN(OUT,';');
    END;

WRITELN(' ENTER TABULAR FUNCTIONS ');
READINPUT(TBBUFF,TBLEN);
   IF TBLEN > 0 THEN
   BEGIN
       WRITE(OUT,' TABULAR FUNCTIONS ');
       FOR I := 1 TO TBLEN DO
       WRITE(OUT,TBBUFF(.I.));
       WRITELN(OUT,';');
    END;
```

```
WRITELN(' ENTER INTERPOLATIONS ');
READINPUT(BUFF,LEN);
   IF LEN > 0 THEN
   BEGIN
       WRITE(OUT,' INTERPOLATIONS .');
       FOR I := 1 TO LEN DO
       WRITE(OUT,BUFF(.I.));
       WRITELN(OUT,';');
    END;
WRITELN(OUT,' END STATIC STRUCTURE');
WRITELN(OUT,' DYNAMIC STRUCTURE');
WRITELN(OUT,' DERIVATIVES');
WRITELN(' ENTER DERIVATIVES');
READINPUT(BUFF,LEN);
   WHILE LEN > 0 DO
   BEGIN
       FOR I := 1 TO LEN DO
       WRITE(OUT,BUFF(.I.));
       WRITELN(OUT,';');
       READINPUT(BUFF,LEN)
    END;
WRITELN(OUT,' END DERIVATIVES');
WRITELN(' ENTER OUTPUT FUNCTION');
READINPUT(BUFF,LEN);
if len > 0 then
 begin
   FLAG := 1;
```

```
    WRITELN(OUT,' OUTPUT FUNCTION')
  end;
    WHILE LEN > 0 DO
    BEGIN
        FOR I := 1 TO LEN DO
        WRITE(OUT,BUFF(.I.));
        WRITELN(OUT,';');
        READINPUT(BUFF,LEN)
     END;
     IF FLAG = 1 THEN
     WRITELN(OUT,' END OUTPUT FUNCTION');
     WRITELN(OUT,' END DYNAMIC STRUCTURE')
END;
PROCEDURE PARAMETERSET;
VAR J : INTEGER;
    I,W,P : INTEGER;
BEGIN
        WRITELN(OUT);
        WRITE(OUT,' PARAMETER SET  FOR ');
        FOR I := 1 TO MLEN DO
        WRITE(OUT,MBUFF(.I.));
        WRITELN(OUT,';');
SYSTEM('EXECSERV CLEARSCN', RC);WRITELN;WRITELN;
        WRITE(' PARAMETER SET SPECIFICATION FOR ');
        FOR I := 1 TO MLEN DO
        WRITE(MBUFF(.I.));
        WRITELN(';');WRITELN;WRITELN;
```

```
FOR P := 1 TO NUMBER DO
BEGIN
 WRITELN(OUT);
 WRITE(OUT,'MODEL ');
 FOR W := 1 TO LCMOD(.P.) DO
   WRITE(' ',CMODNAME(.P,W.));
   WRITELN;WRITELN;
 FOR W := 1 TO 25 DO
   WRITE(OUT,CMODNAME(.P,W.));
   WRITELN(OUT,';');
IF PPLEN(.P.) > 0 THEN
   BEGIN
       WRITE(' ');
       PPLEN(.P.) := PPLEN(.P.) + 1;
       FOR I := 1 TO PPLEN(.P.)  DO
       BEGIN
         IF (PARAM(.P,I.) = ',') OR (I = PPLEN(.P.)) THEN
           BEGIN
             WRITELN(' =  (ENTER THE EXPRESSION)');
             WRITE(OUT,' = ');
             READINPUT(BUFF,LEN);
             FOR J := 1 TO LEN DO
             WRITE(OUT,BUFF(.J.));
             WRITELN(OUT,';')
           END
         ELSE
           IF PARAM(.P,I.) <> ' ' THEN
```

```
            BEGIN
              WRITE(PARAM(.P,I.));
              WRITE(OUT,PARAM(.P,I.))
            END
        END
    END;
    IF TBLEN > 0 THEN
    BEGIN
     TBLEN :=TBLEN + 1;
      FOR I := 1 TO TBLEN DO
      BEGIN
       IF (TBBUFF(.I.) = ',') OR (I = TBLEN) THEN
          BEGIN
            WRITE(' DEFINE ');
            WRITE(OUT,' TABULAR FUNCTION');
            FOR J := 1 TO TBLEN DO
             BEGIN
              WRITE(TBBUFF(.J.));
              WRITE(OUT,TBBUFF(.J.))
             END;
      WRITELN(' TABULAR FUNCTION');
      READINPUT(BUFF,LEN);
WHILE LEN > 0 DO
BEGIN
      FOR J := 1 TO LEN DO
      WRITE(OUT,BUFF(.J.));
      WRITELN(OUT,';');
```

```
        READINPUT(BUFF,LEN)
      END;
      WRITELN(OUT,' END TABULAR FUNCTION')
      END
            ELSE
              IF TBBUFF(.I.) <> ' ' THEN
                 BEGIN
                   WRITE(TBBUFF(.I.));
                   WRITE(OUT,TBBUFF(.I.))
                 END
          END
          END
    END
END;
PROCEDURE EXPERIMENTALFRAME;
VAR J,S,I: INTEGER;
BEGIN
    WRITE(OUT,' FRAME 1 FOR ');
         FOR I := 1 TO MLEN DO
         WRITE(OUT,MBUFF(.I.));
         WRITELN(OUT);
    WRITE(' EXPERIMENTAL FRAME SPECIFICATION FOR ');
         FOR I := 1 TO MLEN DO
         WRITE(MBUFF(.I.)); WRITELN;
    WRITELN(OUT,' GLOBAL');
    WRITELN(' ENTER TIME UNIT');
    READINPUT(BUFF,LEN);
```

```
IF LEN > 0 THEN
  BEGIN
   WRITE(OUT,' TIME UNIT IS ');
   FOR I := 1 TO LEN DO
     WRITE(OUT,BUFF(.I.));
   WRITELN(OUT,';')
  END;

WRITELN(' ENTER SIMULATION STARTING TIME');
READINPUT(BUFF,LEN);
IF LEN > 0 THEN
  BEGIN
   WRITE(OUT,' START TIME IS ');
   FOR I := 1 TO LEN DO
     WRITE(OUT,BUFF(.I.));
   WRITELN(OUT,';')
  END;

WRITELN(' ENTER FINAL TIME');
READINPUT(BUFF,LEN);
IF LEN > 0 THEN
  BEGIN
   WRITE(OUT,' SIMULATE UNTIL TIME > ');
   FOR I := 1 TO LEN DO
     WRITE(OUT,BUFF(.I.));
   WRITELN(OUT,';')
  END;
  WRITELN(OUT,' INTEGRATE BY RUNGE-KUTTA ');
```

```
    WRITELN(' ENTER COMMUNICATION TIME INTERVAL');
    READINPUT(BUFF,LEN);
    IF LEN > 0 THEN
      BEGIN
       WRITE(OUT,' COMMUNICATE AT EVERY ');
        FOR I := 1 TO LEN DO
          WRITE(OUT,BUFF(.I.));
        WRITELN(OUT,';')
      END;
      WRITELN(OUT,' END GLOBAL');

  FOR I := 1 TO NUMBER DO
  BEGIN
   WRITE(OUT,'MODEL ');
    FOR J := 1 TO LCMOD(.I.) DO
      WRITE(OUT,CMODNAME(.I,J.));
    WRITELN(OUT);
   WRITELN(' ENTER INITIALISE AND SAVE STATEMENTS FOR  ');
    FOR J := 1 TO LCMOD(.I.) DO
      WRITE(CMODNAME(.I,J.));
    WRITELN;
READINPUT(BUFF,LEN);
   WHILE LEN > 0 DO
   BEGIN
       FOR J := 1 TO LEN DO
       WRITE(OUT,BUFF(.J.));
       WRITELN(OUT,';');
```

```
        READINPUT(BUFF,LEN)
      END;
       WRITE(OUT,' END MODEL ');
       FOR J := 1 TO LCMOD(.I.) DO
         WRITE(OUT,CMODNAME(.I,J.));WRITELN(OUT);
   END;
     WRITELN(OUT,' END FRAME 1');
     WRITE(OUT,' RUN 1 TO OBSERVE MODEL ');
     FOR J := 1 TO MLEN DO
       WRITE(OUT,MBUFF(.J.)); WRITELN(OUT);
      WRITELN(OUT,'   WITH PARAMETER SET 1 IN FRAME 1;');
      WRITELN(OUT,'   WITH POST RUN');
      WRITELN(OUT,'        OUTPUT MODULE 1 ON PRINTER;');
      WRITELN(OUT,'   END POST RUN;');
      WRITELN(OUT,' END RUN 1'); WRITELN(OUT);
      WRITELN(OUT,' OUTPUT MODULE 1');
      WRITELN(' PREPARE THE OUTPUT MODULE STATEMENTS');
  READINPUT(BUFF,LEN);
     WHILE LEN > 0 DO
     BEGIN
         FOR J := 1 TO LEN DO
         WRITE(OUT,BUFF(.J.));
         WRITELN(OUT,';');
         READINPUT(BUFF,LEN)
      END;
      WRITELN(OUT,' END OUTPUT MODULE 1');

END;
```

```
PROCEDURE COUPLEDMODEL;
 VAR I, J, S, P, W,LEN, FLAG : INTEGER;
 BEGIN
     FLAG := 0;
     SYSTEM('EXECSERV CLEARSCN',RC);WRITELN;WRITELN;WRITELN;
     WRITELN('  COUPLED MODEL SPECIFICATION BEGINS....');
     WRITELN;WRITELN(' ENTER COUPLED MODEL NAME');
     READINPUT(MBUFF,MLEN);
     IF MLEN > 0 THEN
     BEGIN
        WRITE(OUT,' PROGRAM ');
        FOR I := 1 TO MLEN DO
        WRITE(OUT,MBUFF(.I.));
        WRITELN(OUT,';')
     END;
     WRITELN(' ENTER EXTERNAL INPUT(S) IF ANY');
     READINPUT(EXBUFF,EXLEN);
     IF EXLEN > 0 THEN
     BEGIN
         WRITELN(OUT,' EXTERNAL');
         FLAG := 1;
         WRITE(OUT,'   INPUT');
         FOR I := 1 TO EXLEN DO
         WRITE(OUT,EXBUFF(.I.));
         WRITELN(OUT,';')
     END;
```

```
WRITELN(' ENTER EXTERNAL OUTPUTS  IF ANY');
READINPUT(BUFF,LEN);
IF LEN > 0 THEN
BEGIN
    WRITE(OUT,'   OUTPUT');
    FOR I := 1 TO LEN DO
    WRITE(OUT,BUFF(.I.));
    WRITELN(OUT,';')
END;
IF FLAG = 1 THEN
WRITELN(OUT,' END EXTERNAL;');
WRITELN(' ENTER THE NUMBER OF COMPONENT MODELS');
READLN(NUMBER);
WRITE(OUT,' COMPONENT MODELS ');
FOR J := 1 TO NUMBER DO
  BEGIN
    K := J;
    WRITELN(' ENTER NAME OF COMPONENT MODEL ',K);
    READINPUT(BUFF,LEN);
    FOR S := 1 TO LEN DO
     BEGIN
      CMODNAME(.K,S.) := BUFF(.S.);
      WRITE(OUT,CMODNAME(.K,S.))
     END;
   IF J < NUMBER THEN
     WRITE(OUT,',');
      LCMOD(.K.) := LEN;
```

```
    END;
    WRITELN(OUT);
  FOR J := 1 TO NUMBER DO
  BEGIN
      K := J;
     GFLAG := 1;
CONTINUOUSMODEL(BUFF,LEN);
 WRITE(OUT,' END COMPONENT MODEL ');
 FOR S := 1 TO LCMOD(.K.) DO
   WRITE(OUT,CMODNAME(.K,S.));
   WRITELN(OUT)
END;
IF EXLEN > 0 THEN
 BEGIN
 WRITELN(' EQUIVALENCING');
 WRITE(' ');
 FOR S := 1 TO MLEN DO
   WRITE(MBUFF(.S.)); WRITE('.');
 FOR S := 1 TO EXLEN DO
   WRITE(EXBUFF(.S.)); WRITELN(' = ');

      READINPUT(BUFF,LEN);
  IF LEN > 0 THEN
  BEGIN
      WRITELN(OUT,'EQUIVALENCING');
      WRITELN(OUT,'  INPUTS');
 WRITE(OUT,' ');
 FOR S := 1 TO EXLEN DO
```

```
    WRITE(OUT,EXBUFF(.S.)); WRITE(OUT,' = ');

        FOR I := 1 TO LEN DO
        WRITE(OUT,BUFF(.I.));
        WRITELN(OUT,';');
        WRITELN(OUT,' END EQUIVALECING');WRITELN;
    WRITE(' COUPLING FOR ');
    FOR S := 1 TO MLEN DO
      WRITE(MBUFF(.S.)); WRITELN;WRITELN;

(* ZZZZ  COUPLING SPECIFICATION    *)

   FOR P := 1 TO NUMBER DO
   BEGIN
   IF INLEN(.P.) > 0 THEN
      BEGIN
          INLEN(.P.) := INLEN(.P.) + 1;
          FOR I := 1 TO INLEN(.P.)  DO
          BEGIN
           IF (INPAT(.P,I.) = ',') OR (I = INLEN(.P.)) THEN
             BEGIN
             WRITE(' ');
             FOR W := 1 TO LCMOD(.P.) DO
             WRITE(CMODNAME(.P,W.));WRITE('.');
               WRITELN(' <-----   (ENTER THE SOURCE)');
                READINPUT(BUFF,LEN);
               WRITE(OUT,' ');
               FOR W := 1 TO LCMOD(.P.) DO
               WRITE(OUT,CMODNAME(.P,W.));WRITE(OUT,'.');
```

```
            FOR J := 1 TO LEN DO
              WRITE(OUT,BUFF(.J.));
              WRITELN(OUT,';')
            END
          ELSE
            IF INPAT(.P,I.) <> ' ' THEN
              BEGIN
                WRITE(INPAT(.P,I.));
                WRITE(OUT,INPAT(.P,I.))
              END
            END
          END;
        WRITE(OUT,' END COUPLING FOR ');
  FOR S := 1 TO MLEN DO
    WRITE(OUT,MBUFF(.S.)); WRITELN(OUT,';');WRITELN(OUT)
    END;
    WRITE(OUT,' END MODEL');
  FOR S := 1 TO MLEN DO
    WRITE(OUT,MBUFF(.S.)); WRITELN(OUT,';');WRITELN(OUT)
 END;
PARAMETERSET;
WRITELN(OUT,' END PARAMETER SET 1');
EXPERIMENTALFRAME;
WRITE(OUT,' END PROGRAM ');
FOR I := 1 TO MLEN DO
   WRITE(OUT,MBUFF(.I.));
```

```
 WRITELN(OUT)
END;
PROCEDURE INIT;
VAR N : INTEGER;
BEGIN
      SYSTEM('EXECSERV CLEARSCN', RC); WRITELN;WRITELN;WRITELN;
      WRITELN('  THIS FACILITY PROVIDES FOR THE SPECIFICATION ');
      WRITELN('              OF THE FOLLOWING :');WRITELN;
      WRITELN('    1      COMPONENT MODELS ');
      WRITELN('    2      COUPLED MODEL SPECIFICATION');
      WRITELN('    3      PARAMETER SET SPECIFICATION');
      WRITELN;WRITELN;
      WRITELN('  ENTER THE FACILITY YOU WANT, PLEASE');READLN;
      READ(N);
      CASE N OF
           1: CONTINUOUSMODEL(BUFF,LEN);
           2: COUPLEDMODEL;
           3: PARAMETERSET;
      OTHERWISE
      SYSTEM('EXECSERV CLEARSCN', RC);
      WRITELN(' I AM SORRY I CANT HELP YOU, TRY AGAIN');
     END;
    END;
     BEGIN
      REWRITE(OUT);
      GFLAG := 0;
      K:=1;
```

```
    INIT;
    WRITELN('      END OF A SESSION.........');WRITELN;
    CLOSE(OUT);
END.
```

## Appendix D

## TEMPLATE GENERATOR PROGRAM

```
PREAMBLE
    NORMALLY MODE IS REAL
    DEFINE VAR AS TEXT VARIABLE
END ''PREAMBLE
MAIN
    USE 44 FOR OUTPUT
    WRITE AS "PROGRAM ",/
    WRITE AS " ENTER ",/ USING 6
    WRITE AS "    CPM        FOR COUPLED MODEL",/ USING 6
    WRITE AS "    CNT        FOR CONTINUOUS MODEL",/ USING 6
    WRITE AS "    MEM        FOR MEMORYLESS",/ USING 6
    WRITE AS "    DIS        FOR DISCRETE MODEL",/ USING 6
    WRITE AS " ",/ USING 6
    READ VAR
    IF VAR = "CPM"
       CALL COUPLED
    ELSE IF VAR = "CNT"
            CALL CONTINUOUS
            CALL PARAMETERSET
         ELSE IF VAR = "MEM"
                 CALL MEMO
              ELSE IF VAR = "DIS"
```

```
                    CALL DISCRETE
                    ELSE WRITE AS "INVALID RESPONSE",/ USING 6
                ALWAYS
            ALWAYS
        ALWAYS
    ALWAYS
END ''MAIN
ROUTINE CONTINUOUS
    WRITE AS "CONTINUOUS MODEL",/
    WRITE AS " STATIC STRUCTURE",/
    WRITE AS "   INPUT ",/
    WRITE AS "   STATE ",/
    WRITE AS "   OUTPUT",/
    WRITE AS "   AUXILIARY VARIABLES ",/
    WRITE AS "   AUXILIARY PARAMETERS",/
    WRITE AS "   END AUXILIARY PARAMETERS;",/
    WRITE AS "   PARAMETERS ",/
    WRITE AS "   TABULAR FUNCTIONS ",/
    WRITE AS "   INTERPOLATIONS ",/
    WRITE AS "   END INTERPOLATIONS;",/
    WRITE AS " END STATIC STRUCTURE",/
    WRITE AS " DYNAMIC STRUCTURE",/
    WRITE AS "    DERIVATIVES ",/
    WRITE AS "    END DERIVATIVES;",/
    WRITE AS "    OUTPUT FUNCTION ",/
    WRITE AS "    END OUTPUT FUNCTION;",/
    WRITE AS " END DYNAMIC STRUCTURE;",/
```

```
    WRITE AS " END MODEL ........",/
END ''CONTINUOUS
ROUTINE PARAMETERSET
    WRITE AS " PARAMETER SET 1 FOR  ......",/
    WRITE AS " END PARAMETER SET 1;",/
    WRITE AS " FRAME 1 FOR ......",/
    WRITE AS "   GLOBAL ",/
    WRITE AS "      TIME UNIT IS ......",/
    WRITE AS "      SIMULATE UNTIL  ......",/
    WRITE AS "      START TIME .....",/
    WRITE AS "      INTEGRATE BY RUNGE_KUTTA;",/
    WRITE AS "      COMMUNICATE AT EVERY ....",/
    WRITE AS "   END GLOBAL;",/
    WRITE AS "   MODEL .....",/
    WRITE AS "       INITIALISE STATE ",/
    WRITE AS "       SAVE    ",/
    WRITE AS "       INTERPOLATION LINEAR",/
    WRITE AS "   END MODEL  ....: ",/
    WRITE AS " END FRAME 1;",/
    WRITE AS " RUN 1",/
    WRITE AS " END RUN 1;",/
    WRITE AS " OUTPUT MODULE 1",/
    WRITE AS " END OUTPUT MODULE 1;",/
    WRITE AS "END PROGRAM ....",/
 END ''PARAMETERSET
ROUTINE MEMO
    WRITE AS " MEMORYLESS MODEL ",/
```

```
    WRITE AS "   STATIC STRUCTURE ",/
    WRITE AS "     INPUT ",/
    WRITE AS "     OUTPUT ",/
    WRITE AS "     PARAMETERS ",/
    WRITE AS "     AUXILIARY VARIABLES",/
    WRITE AS "     AUXILIARY PARAMETERS",/
    WRITE AS "     END AUXILIARY PARAMETERS;",/
    WRITE AS "   END STATIC STRUCTURE;",/
    WRITE AS "   DYNAMIC STRUCTURE",/
    WRITE AS "   END DYNAMIC STRUCTURE;",/
    WRITE AS " END MODEL ....",/
END ''MEMO
ROUTINE DISCRETE
    WRITE AS " DISCRETE MODEL ",/
    WRITE AS "   STATIC STRUCTURE ",/
    WRITE AS "     INPUT ",/
    WRITE AS "     STATE ",/
    WRITE AS "     OUTPUT ",/
    WRITE AS "     PARAMETERS ",/
    WRITE AS "     AUXILIARY VARIABLES",/
    WRITE AS "     AUXILIARY PARAMETERS",/
    WRITE AS "     END AUXILIARY PARAMETERS;",/
    WRITE AS "   END STATIC STRUCTURE;",/
    WRITE AS "   DYNAMIC STRUCTURE",/
    WRITE AS "     STATE TRANSITION",/
    WRITE AS "     END STATE TRANSITION;",/
    WRITE AS "   END DYNAMIC STRUCTURE;",/
```

```
      WRITE AS " END MODEL ....",/
   END ''DISCRETE
   ROUTINE COUPLED
      WRITE AS "COUPLED MODEL ",/
      WRITE AS "  EXTERNAL ",/
      WRITE AS "     INPUT ",/
      WRITE AS "     OUTPUT ",/
      WRITE AS "  END EXTERNAL;",/
      WRITE AS "  COMPONENT MODELS ",/
      CALL CONTINUOUS
      WRITE AS " EQUIVALENCING ",/
      WRITE AS "   INPUTS ",/
      WRITE AS " END EQUIVALENCING;",/
      WRITE AS " COUPLING FOR ...",/
      WRITE AS " END COUPLING FOR ",/
      WRITE AS "END MODEL ....",/
      CALL PARAMETERSET
END ''COUPLED
```

## Appendix E

## DEER POPULATION MODEL

```
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
                                        START TIME = 1880;
        SIMULATE UNTIL TIME = 1970;
        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
    PRINT DP, PP, DKPP VERSUS TIME;
```

```
END OUTPUT MODULE 1;

END PROGRAM STUDY_OF_DEER_POPULATION;
```

## Appendix F

## TRANSLATED DEER POPULATION MODEL

```
PREAMBLE
NORMALLY, MODE IS REAL
DEFINE W, DW AS 1-DIM ARRAY
DEFINE H AS A VARIABLE
DEFINE TNOSV2 AS INTEGER VARIABLE
DEFINE INTERPOLATE AS FUNCTION
DEFINE  DP AS VARIABLES
DEFINE  PP,  DKPP AS VARIABLES
DEFINE  DNI,  DPR,  DD AS VARIABLES
DEFINE  AREA,  NIR AS VARIABLES
DEFINE  PPT,  DKPPT AS 2-DIM ARRAY
END ''PREAMBLE
''****************************
''*   MAIN.PORTION          **
''*      OF PROGRAM         **
''****************************
MAIN
RESERVE W, DW AS   1
LET TNOSV =  1
LET TNOSV2 = TNOSV
LET H = .07
LET     AREA = 800000.0
```

```
LET     NIR  = 0.2
RESERVE PPT AS  2 BY 2
READ PPT USING 24
RESERVE DKPPT AS  6 BY 2
READ DKPPT USING 24
LET           DP = 4000.
USE 86 FOR OUTPUT
PRINT  1 LINES AS FOLLOWS
   STUDY OF DEER POPULATION
LET TIME = 1880
LET  W( 1)  =  DP
UNTIL TIME = 1970
DO
LET  PP= INTERPOLATE ( 2,TIME,PPT(*,*))
LET  DKPP= INTERPOLATE ( 6,DD,DKPPT(*,*))
CALL RUNGEKUTTA GIVEN W(*), DW(*), TIME,TNOSV
ADD 1.0 TO TIME
WRITE TIME, DP, PP, DKPP AS D(8,2),S 3, 3 D(12,6),/ USING 88
LOOP
END ''MAIN
''****************************
''*     DERIVATIVE          **
''*      FUNCTION           **
''****************************
ROUTINE DERIV(W, DW,TME)
DEFINE W, DW AS 1-DIM ARRAY
DEFINE L,B,K AS INTEGER VARIABLES
```

```
DEFINE TME AS VARIABLE
RESERVE W, DW AS  1
LET  DP = W( 1)
          DNI  = NIR*DP
          DPR  = PP*DKPP
 W( 1) = DNI - DPR
          DD   = DP/AREA
FOR K = 1 TO TNOSV2
DO
 LET DW(K) = W(K)
LOOP
END ''DERIV
```

## Appendix G

## RESULTS OF DEER POPULATION MODEL

STUDY OF DEER POPULATION

| TIME | DP | PP | DKPP |
|---|---|---|---|
| 1881.00 | 4056.394531 | 300.000000 | 0. |
| 1882.00 | 4050.124023 | 300.000000 | 3.000705 |
| 1883.00 | 4043.767822 | 300.000000 | 3.000626 |
| 1884.00 | 4037.323730 | 300.000000 | 3.000546 |
| 1885.00 | 4030.790283 | 300.000000 | 3.000466 |
| 1886.00 | 4024.166504 | 300.000000 | 3.000384 |
| 1887.00 | 4017.451172 | 300.000000 | 3.000301 |
| 1888.00 | 4010.642578 | 300.000000 | 3.000217 |
| 1889.00 | 4003.739746 | 300.000000 | 3.000133 |
| 1890.00 | 3996.741943 | 300.000000 | 3.000047 |
| 1891.00 | 4052.773437 | 300.000000 | .014988 |
| 1892.00 | 4046.452881 | 300.000000 | 3.000659 |
| 1893.00 | 4040.045898 | 300.000000 | 3.000580 |
| 1894.00 | 4033.550049 | 300.000000 | 3.000500 |
| 1895.00 | 4026.964355 | 300.000000 | 3.000419 |
| 1896.00 | 4020.287598 | 300.000000 | 3.000337 |
| 1897.00 | 4013.518799 | 300.000000 | 3.000253 |
| 1898.00 | 4006.655762 | 300.000000 | 3.000169 |
| 1899.00 | 3999.698486 | 300.000000 | 3.000083 |
| 1900.00 | 4055.771484 | 300.000000 | .014999 |

| | | | |
|---|---|---|---|
| 1901.00 | 4049.492187 | 300.000000 | 3.000697 |
| 1902.00 | 4043.127197 | 300.000000 | 3.000618 |
| 1903.00 | 4036.674072 | 300.000000 | 3.000539 |
| 1904.00 | 4030.131836 | 300.000000 | 3.000458 |
| 1905.00 | 4023.499023 | 300.000000 | 3.000376 |
| 1906.00 | 4016.774170 | 300.000000 | 3.000294 |
| 1907.00 | 4009.956543 | 300.000000 | 3.000209 |
| 1908.00 | 4003.044678 | 300.000000 | 3.000124 |
| 1909.00 | 3996.036865 | 300.000000 | 3.000037 |
| 1910.00 | 4052.058105 | 300.000000 | .014985 |
| 1911.00 | 4045.727539 | 300.000000 | 3.000650 |
| 1912.00 | 4039.310547 | 300.000000 | 3.000571 |
| 1913.00 | 4032.804687 | 300.000000 | 3.000491 |
| 1914.00 | 4026.208496 | 300.000000 | 3.000409 |
| 1915.00 | 4019.521240 | 300.000000 | 3.000327 |
| 1916.00 | 4012.741699 | 300.000000 | 3.000243 |
| 1917.00 | 4005.868164 | 300.000000 | 3.000158 |
| 1918.00 | 3998.899658 | 300.000000 | 3.000072 |
| 1919.00 | 4054.961426 | 300.000000 | .014996 |
| 1920.00 | 4048.670898 | 300.000000 | 3.000687 |
| 1921.00 | 4042.294189 | 300.000000 | 3.000607 |
| 1922.00 | 4035.829590 | 300.000000 | 3.000528 |
| 1923.00 | 4029.275635 | 300.000000 | 3.000447 |
| 1924.00 | 4022.630859 | 300.000000 | 3.000365 |
| 1925.00 | 4015.894287 | 300.000000 | 3.000282 |
| 1926.00 | 4009.064453 | 300.000000 | 3.000198 |
| 1927.00 | 4002.140137 | 300.000000 | 3.000113 |

| | | | |
|---|---|---|---|
| 1928.00 | 3995.120117 | 300.000000 | 3.000027 |
| 1929.00 | 4051.128906 | 300.000000 | .014982 |
| 1930.00 | 4044.785645 | 300.000000 | 3.000639 |
| 1931.00 | 4038.355469 | 300.000000 | 3.000559 |
| 1932.00 | 4031.836426 | 300.000000 | 3.000479 |
| 1933.00 | 4025.227051 | 300.000000 | 3.000398 |
| 1934.00 | 4018.526367 | 300.000000 | 3.000315 |
| 1935.00 | 4011.732666 | 300.000000 | 3.000231 |
| 1936.00 | 4004.845459 | 300.000000 | 3.000146 |
| 1937.00 | 3997.862793 | 300.000000 | 3.000060 |
| 1938.00 | 4053.909912 | 300.000000 | .014992 |
| 1939.00 | 4047.604736 | 300.000000 | 3.000673 |
| 1940.00 | 4041.213379 | 300.000000 | 3.000594 |
| 1941.00 | 4034.733643 | 300.000000 | 3.000515 |
| 1942.00 | 4028.164307 | 300.000000 | 3.000434 |
| 1943.00 | 4021.504150 | 300.000000 | 3.000352 |
| 1944.00 | 4014.751953 | 300.000000 | 3.000268 |
| 1945.00 | 4007.906250 | 300.000000 | 3.000184 |
| 1946.00 | 4000.965820 | 300.000000 | 3.000098 |
| 1947.00 | 3993.929687 | 300.000000 | 3.000011 |
| 1948.00 | 4049.921875 | 300.000000 | .014977 |
| 1949.00 | 4043.561768 | 300.000000 | 3.000624 |
| 1950.00 | 4037.114746 | 300.000000 | 3.000544 |
| 1951.00 | 4030.578613 | 300.000000 | 3.000463 |
| 1952.00 | 4023.951660 | 300.000000 | 3.000381 |
| 1953.00 | 4017.233398 | 300.000000 | 3.000299 |
| 1954.00 | 4010.421875 | 300.000000 | 3.000215 |

| | | | |
|---|---|---|---|
| 1955.00 | 4003.516113 | 300.000000 | 3.000130 |
| 1956.00 | 3996.515381 | 300.000000 | 3.000044 |
| 1957.00 | 4052.543701 | 300.000000 | .014987 |
| 1958.00 | 4046.219971 | 300.000000 | 3.000656 |
| 1959.00 | 4039.809814 | 300.000000 | 3.000577 |
| 1960.00 | 4033.310547 | 300.000000 | 3.000497 |
| 1961.00 | 4090.174561 | 0. | 3.000416 |
| 1962.00 | 4084.371338 | 300.000000 | 3.001126 |
| 1963.00 | 4078.488770 | 300.000000 | 3.001054 |
| 1964.00 | 4072.524902 | 300.000000 | 3.000980 |
| 1965.00 | 4066.478516 | 300.000000 | 3.000906 |
| 1966.00 | 4060.348389 | 300.000000 | 3.000831 |
| 1967.00 | 4054.133545 | 300.000000 | 3.000753 |
| 1968.00 | 4047.832764 | 300.000000 | 3.000676 |
| 1969.00 | 4041.444824 | 300.000000 | 3.000597 |
| 1970.00 | 4034.968262 | 300.000000 | 3.000518 |

## Appendix H

## MOTOR MODEL

```
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  = 1.0 * 10 ** 10;
  END MODEL PID_CONTROLLER;
  MODEL MOTOR
    KM  = 6.2 * 10 ** (-3);
    R   = 5.3;
    J   = 7.5 * 10 ** (-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;
THDOT=  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;
```

## Appendix I

## TRANSLATED MOTOR MODEL

```
PREAMBLE
NORMALLY, MODE IS REAL
DEFINE W, DW AS 1-DIM ARRAY
DEFINE H AS A VARIABLE
DEFINE TNOSV2 AS INTEGER VARIABLE
DEFINE INTERPOLATE AS FUNCTION
DEFINE  YREF AS VARIABLES
DEFINE  Y AS VARIABLES
DEFINE  I,  X AS VARIABLES
DEFINE  U AS VARIABLES
DEFINE  E,  P,  D AS VARIABLES
DEFINE  G,  GD,  TD,  TI AS VARIABLES
DEFINE  TH,  THDOT AS VARIABLES
DEFINE  ME AS VARIABLES
DEFINE  KM,  R,  J,  CT AS VARIABLES
END ''PREAMBLE
''****************************
''*   MAIN.PORTION        **
''*     OF PROGRAM        **
''****************************
MAIN
RESERVE W, DW AS   4
```

```
LET TNOSV =  4
LET TNOSV2 = TNOSV
LET H = .07
LET  YREF= 1.7500
LET      G   = 1.0
LET        GD  = 1.0
LET        TD  = 1.0
LET        TI  = 1.0 * 10 ** 10
LET         KM  = 6.2 * 10 ** (-3)
LET         R   = 5.3
LET         J   = 7.5 * 10 ** (-7)
LET         CT  = 0.033
LET  I = 0.0
LET  X = 0.0
LET TH   =  0.0
LET THDOT=  0.0
USE 86 FOR OUTPUT
PRINT  1 LINES AS FOLLOWS
          STUDY OF MOTOR_CONTROLLER
LET  W( 1)  =  I
LET  W( 2)  =  X
LET  W( 3)  =  TH
LET  W( 4)  =  THDOT
LET         P = G*E
LET       Y = CT*TH
LET         D = -GD*(Y - X)
LET         U = P+I+D
```

```
UNTIL TIME > 1.0
DO
CALL RUNGEKUTTA GIVEN W(*), DW(*), TIME,TNOSV
LET          P = G*E
LET        Y = CT*TH
LET          D = -GD*(Y - X)
LET          U = P+I+D
ADD 0.01  TO TIME
WRITE TIME, U, Y AS D(8,2),S 3, 2 D(12,6),/ USING 88
LOOP
END ''MAIN
''*****************************
''*    DERIVATIVE          **
''*      FUNCTION          **
''*****************************
ROUTINE DERIV(W, DW,TME)
DEFINE W, DW AS 1-DIM ARRAY
DEFINE L,B,K AS INTEGER VARIABLES
DEFINE TME AS VARIABLE
RESERVE W, DW AS   4
LET  I = W( 1)
LET  X = W( 2)
LET  TH = W( 3)
LET  THDOT = W( 4)
          E = YREF - Y
 W( 1) = -GD/TD*(X - Y)
 W( 2) = E/TI
```

```
W( 3) = THDOT
        I   = (U-KM*THDOT)/R
       ME     = KM*I
W( 4) = ME/J
FOR K = 1 TO TNOSV2
DO
 LET DW(K) = W(K)
LOOP
END ''DERIV
```

## Appendix J

## RESULTS OF TRANSLATED MOTOR MODEL

STUDY OF MOTOR_CONTROLLER

| TIME | U | Y |
|---|---|---|
| .01 | 1.750000 | 0. |
| .02 | 1.760730 | .145991 |
| .03 | 1.110237 | .574540 |
| .04 | .097679 | 1.060658 |
| .05 | -.824410 | 1.414354 |
| .06 | -1.313672 | 1.516124 |
| .07 | -1.236596 | 1.358830 |
| .08 | -.694420 | 1.035640 |
| .09 | .045830 | .691647 |
| .10 | .674840 | .461578 |
| .11 | .965138 | .419368 |
| .12 | .848096 | .557463 |
| .13 | .420189 | .800023 |
| .14 | -.114940 | 1.040689 |
| .15 | -.538236 | 1.187286 |
| .16 | -.701299 | 1.195418 |
| .17 | -.574432 | 1.079480 |
| .18 | -.243030 | .899687 |
| .19 | .139775 | .733157 |
| .20 | .420481 | .642162 |

| | | |
|---|---|---|
| .21 | .504211 | .652088 |
| .22 | .383732 | .746357 |
| .23 | .131138 | .878100 |
| .24 | -.139858 | .991986 |
| .25 | -.322865 | 1.046542 |
| .26 | -.358739 | 1.028323 |
| .27 | -.252318 | .953532 |
| .28 | -.062466 | .858044 |
| .29 | .127358 | .781147 |
| .30 | .244264 | .750016 |
| .31 | .252584 | .770818 |
| .32 | .162860 | .828993 |
| .33 | .021948 | .897476 |
| .34 | -.109566 | .948664 |
| .35 | -.182368 | .965089 |
| .36 | -.175930 | .944842 |
| .37 | -.102747 | .900335 |
| .38 | .000634 | .851730 |
| .39 | .090694 | .818208 |
| .40 | .134537 | .810735 |
| .41 | .121172 | .828884 |
| .42 | .062968 | .862446 |
| .43 | -.012051 | .896582 |
| .44 | -.072953 | .918113 |
| .45 | -.098135 | .920363 |
| .46 | -.082445 | .904908 |
| .47 | -.037079 | .879923 |

| | | |
|---|---|---|
| .48 | .016787 | .856208 |
| .49 | .057394 | .842708 |
| .50 | .070825 | .843317 |
| .51 | .055361 | .856013 |
| .52 | .020596 | .874395 |
| .53 | -.017679 | .890682 |
| .54 | -.044322 | .898887 |
| .55 | -.050574 | .896881 |
| .56 | -.036614 | .886725 |
| .57 | -.010358 | .873351 |
| .58 | .016555 | .862302 |
| .59 | .033706 | .857526 |
| .60 | .035746 | .860064 |
| .61 | .023802 | .868015 |
| .62 | .004231 | .877643 |
| .63 | -.014489 | .885037 |
| .64 | -.025275 | .887641 |
| .65 | -.024991 | .885061 |
| .66 | -.015145 | .878944 |
| .67 | -.000728 | .872086 |
| .68 | .012146 | .867213 |
| .69 | .018728 | .865947 |
| .70 | .017287 | .868317 |
| .71 | .009391 | .872953 |
| .72 | -.001111 | .877788 |
| .73 | -.009857 | .880940 |
| .74 | -.013711 | .881413 |

| | | |
|---|---|---|
| .75 | -.011811 | .879362 |
| .76 | -.005615 | .875895 |
| .77 | .001955 | .872523 |
| .78 | .007818 | .870528 |
| .79 | .009938 | .870502 |
| .80 | .007974 | .872206 |
| .81 | .003198 | .874767 |
| .82 | -.002201 | .877094 |
| .83 | -.006072 | .878321 |
| .84 | -.007120 | .878117 |
| .85 | -.005300 | .876742 |
| .86 | -.001675 | .874871 |
| .87 | .002136 | .873286 |
| .88 | .004646 | .872559 |
| .89 | .005055 | .872860 |
| .90 | .003473 | .873945 |
| .91 | .000758 | .875297 |
| .92 | -.001904 | .876364 |
| .93 | -.003497 | .876771 |
| .94 | -.003544 | .876446 |
| .95 | -.002223 | .875606 |
| .96 | -.000215 | .874640 |
| .97 | .001623 | .873933 |
| .98 | .002606 | .873725 |
| .99 | .002466 | .874032 |
| 1.00 | .001397 | .874672 |
| 1.01 | -.000072 | .875356 |

## REFERENCES


1. ANNINO, J.S. and E.C. RUSSELL (1979), "The Ten Most Frequent Causes of Simulation Analysis Failure - and How to Avoid Them!", Simulation Today, No.60, Simulation, Vol.32, No.6, pp. 137-140.

2. BALCI, OSMAN, (1983) "Requirements for Model Development Environments", Technical Report CS83022-R, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24021, October 1983

3. BERRY, R.E., 1982 Programming Language Translation, John Wiley & Sons, Halsted Press, Chichester

4. BIRTA, L.G., OREN, T.I., and KETTENIS, D.,"A Robust procedure for discontinuity handling in Continuous System Simulation", Technical Report , Computer Science Dept., University of Ottawa, Ottawa, Ontario, Canada.

5. CARVER M.B., 1978, "Efficient Integration over Discontinuities in Ordinary differential equation simulations," Mathematics and Computers in Simulation XX(1978) 190 - 196, North-Holland Publishing Company, Amsterdam

6. CELLIER, F.E. (1979) "Combined continuous/discrete system simulation by the use of digital computers: techniques and tools", Ph.D thesis, Swiss Federal Institute of Technology, Zurich, 1979.

7. CROSBIE, R. E and J. L. HAY (1974) Digital Techniques for the Simulation of Discontinuities, In Proceedings of Summer Computer Simulation Conference, Houston, Texas, July 1974, pp 87-94

8. C.A.C.I (1982) SIMSCRIPT II.5 User's Manual IBM S/370 Series Release 9.1

9. DAVIES, N.R. (1979). Interactive Simulation Program Generation. In (B.P. Zeigler et al. eds.) "Methodology in Systems Modelling and Simulation", NorthHolland, Amsterdam, pp. 179-200

10. DINES BJORNER & CLIFF B.JONES, 1984 Formal Specification & Software Development, Prentice-Hall, Inc., Englewood Cliffs, New Jersey

11. FUTÓ, I., and GERGELY, T. (1983), "A logical approach to Simulation", In: Adequate Modelling of Systems, H. Wedde (Ed.), Springer-Verlag, Berlin, W. Germany, pp. 25-48

12. GASS, S.I. (1983), "Decision-Aiding Models: Validation, Assessment, and related Issues for Policy Analysis," Operations Research, Vol.31, No.4, pp.603-631.

13. GEAR, C.W. and OSTERBY, O (1984) "Solving ordinary differential equations with discontinuities",ACM Transactions on Mathematical Software, Vol.10, 1984, pp.23-24

14. GREENSPAN, D. (1973). Discrete Models. Addison-Wesley, Reading, MA,165p.

15. GREENSPAN, D. (1980). Arithmetic Applied Mathematics. Pergamon Press, Elmsford,NY, 165p.

16. HAY, J.L. "Numerical Integration Methods and discontinuities", Paper presented at the United Kingdom Simulation Council Conference at Wolverham Polytechnic on Sept 24, 1974

17. HAY, J.L., CROSBIE, R.E and CHAPLIN, R.I. (1974) "Integration routines for Systems with Discontinuities", Computer Journal, Vol. 17, No.3, Aug. 1974, pp 275-278

18. HALIN, H.J. (1979) "Integration across discontinuities in ordinary differential equations using power series", Simulation, February 1979, Vol 32, pp 33-45

19. HOWE, R.M. (1978) A new method for handling discontinuous nonlinear functions in Digital Simulation of Dynamic Systems, Summer Computer Simulation Conference, July 24-26, 1978, California

20. KLIR, G.J. (1978). Computer-Aided System Modelling, In "Systems Theory in Ecology",(E.Halfon, ed.), Academic Press, New York.

21. NANCE, R.E. (1981), "Model Representation in Discrete Event Simulation: The Conical Methodology," Technical Report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, VA.

22. NANCE ROBERTS, D. ANDERSON, DEAL, R.M., GARET M.S., SHAFFER W.A., 1983 Introduction to Computer Simulation: The System Dynamics Approach, Addison-Wesley Publishing Company, Reading, Massachusetts

23. NANCE, R.E. and O. BALCI (1984), "The Objectives and Requirements of Model Management," In: M. Singh (editor-in-chief), Encyclopedia of Systems and Control, Pergamon Press, Oxford, to appear.

24. OREN, T.I. (1979) Concepts for Advanced Computer-Assisted Modelling, In (B.P Zeigler et al., eds) "Methodology in Systems Modelling and Simulation", NorthHolland, Amsterdam, pp. 29-55

25. OREN, T.I. (1981) Concepts and Criteria to Assess Acceptability of Simulation Studies: A Frame of Reference , Communications of the ACM, Vol.24, No.4, pp.180-189.

26. OREN, T.I.,(1982a), "Quality Assurance of System Design and Model Management for Complex problems," International Working Conference on Model Realism, 1982 April 20 - 23, Bad Honnef near Bonn, Germany

27. OREN, T.I. (1982b) "Definition Manual of GEST 81", Technical Report TR 82-05, Computer Science Dept., University of Ottawa, Ottawa, Ontario, Canada.

28. OREN T.I. (1983) Simulation Methodology : A Top-Down View, Technical Report TR-83-09, Computer Science Dept, University of Ottawa, Ottawa, Ontario,Canada

29. OREN T.I. (1984) Symbolic Processing of Simulation Models : A Taxonomy, Technical Report TR-84-01, prepared for The Encyclopedia of Systems and Control, M. Shingh (Editor-in-chief),Pergamon Press, England

30. OREN T.I. (1984) Simulation and Model-oriented Languages : A Taxonomy, Technical Report TR-84-03, prepared for The Encyclopedia of Systems and Control, M. Shingh (Editor-in-chief),Pergamon Press, England

31. OREN, T.I. and COLLIE, B. (1980) Design of SEMA : A Software System for Computer-Assisted Modelling and Simulation of SEquential MAchines, In (T.I. Oren et al., eds.) Proceedings of 1980 Winter Simulation Conference pp. 113-123

32. OREN T.I and ZEIGLER, B.P. (1979) Concepts for Advanced Simulation Methodologies, Simulation, 32, No.3, pp. 69-82

33. ROTH, P.F., S.I. GASS, and A.J. LEMOINE (1978),"Some considerations for Improving Federal Modeling," Proc. Winter Simulation Conference, Miami Beach, FL, pp. 213-217.

34. STREZOVA, Z. (1980),"Complex Systems Modelling and Analysis: a Methodology and a Software System, Proceedings of the IMACS European Simulation Meeting on Discrete Simulation and Related Fields, Keszthely, Hungary, 10-12 September, 1980, In: A. Javor (editor), Discrete Simulation and Related Fields, pp 105-113

35. SUBRAHMANIAN and CANNON (1981), "A generator program for models of discrete-event systems, Simulation, March 1981, pp. 93-101

36. U.S. General Accounting Office (1976), "Report to the Congress: Ways to improve Management of Federally Funded Computerized Models," LCD-75-111, U.S. General Accounting Office, Washington, D.C.

37. WASSERMAN, A.I. and S. GUTZ (1982),"The Future of Programming," Communications of the ACM, Vol. 25, No. 3, pp. 196-206.

38. WILLIAM M. WAITE & GERHARD GOOS, 1984 Compiler Construction, Springer-Verlag New York Inc.

39. WYMORE, A.W. (1967). A Mathematical Theory of Systems Engineering -- The Elements, Wiley, New York, 353p.

40. WYMORE, A.W. (1976). Systems Engineering Methodology for Interdisciplinary Teams, Wiley-Interscience, New York, 431p.

41. YIN-CHANG LIU, COLEMAN B. BROSILOW (1983),"MOdular Integration Methods for Simulation of Large Scale Dynamic Systems", Paper presented at AIChE Diamond Jubilee Meeting, Washington D.C., Nov.1 1983.

42. ZEIGLER, B.P. (1976). Theory of Modelling and Simulation, John Wiley, New York