SimGest:
A Simulation Experimentation Environment and a Program Generator for Interactive Simulation

By
Qiang Jin

A thesis submitted to the School of Graduate Studies
and Research of the University of Ottawa
in partial fulfillment of the requirements of
Master of Science in Systems Science

University of Ottawa
Ottawa, Ontario
Canada
August 1993

ISBN 0-315-95927-4

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

I hereby declare that I am the sole author of this document. I authorize the University of Ottawa to lend this document to other institutions or individuals for the purpose of scholarly research.

Qiang Jin

I further authorize the University of Ottawa to reproduce this document by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Qiang Jin

To my parents whom I love and respect.

Table of Contents

Acknowledgement

I would like to express my gratitude to my thesis supervisor, Dr. Tuncer I. Ören, for his valuable time, guidance and advice during the research and preparation of my thesis.

I would also like to extend my thanks to my friends for their encouragement and assistance in my thesis preparation.

Trademark and Copyright Acknowledgement

Abstract

A good methodology for simulation experimentation can result in high efficiency in carrying out simulation experiments. To fully support a user to carry out a simulation experiment, a supportive and user friendly simulation experimentation environment is desirable. The program generator which generates the executable programs from a model expressed in a high-level simulation programming language is another important issue in simulation experimentation.

This thesis presents a new methodology for simulation experimentation based on the model specification language Gest. A simulation experimentation environment which supports a user to carry out the simulation experiment based on the model written in Gest is implemented. A program generator which analyzes the model and generates an executable simulation program with simulation tables for the simulation experimentation environment is also discussed.

List of Figures

1. Introduction

Simulation is experimentation with dynamic models, i.e., models that have time-varying behavior. In computerized simulation, the interest is in computer-processable models which have behavior that can be indexed with time (Ören 1987, p. 4321). The continuous models discussed in this thesis are described by ordinary differential equations.

In simulation, a model usually is defined and expressed in a high-level programming language. To carry out the simulation experiments with the specifications of the experiments that determine the behavior of the model, the high-level simulation specifications must be translated into a program which can be compiled and then executed on a computer.

Simulation environments can have extensive abilities for every phase of a simulation study, such as model specification and identification, parameter fitting and tuning, specification and checking of parameters, experimental conditions and run control, simulation behavior analysis and display etc. A good simulation system should allow users to specify the models in a high-level specification language and carry out the simulation experiments interactively.

In this thesis, a simulation experimentation environment is developed for models expressed in a high-level model specification language named Gest, GEneral Systems Theory implementer (Ören 1984). In Gest, a specific model is composed of a parametric model and the associated sets of model parameter values, called also parameter sets. A given parametric model can be associated with different parameter sets. A simulation model consists of a declarative part and a generative part. In the declarative part, the static structure of a model is given by specifying the names of different types of descriptive elements of a model, such as input, state, output and auxiliary variables, constants, parameters and auxiliary parameters. To carry out the simulation experimentation with a Gest model on a computer, the program generator developed by Ye (1989) is used to translate the model into a compilable C program. In this thesis, Ye's program generator is modified as follows: the program generator creates a C program and

associated simulation tables to contain information necessary to carry out simulation runs. These tables are accessed by the executable C program to obtain the simulation experimental conditions for each run. Therefore, the corresponding table-driven C program is generated only once. In its original form (Ye 1989) for each simulation run, the C program had to be generated.

The interactive simulation experimentation environment gives a user a method to interface with the program generator to select a model and generate a corresponding executable C program. The simulation experimentation environment also generates the templates which list parameters, initial conditions and integration constants necessary to carry out the simulation runs. Furthermore, the experimentation environment uses these templates to receive appropriate information and puts them in the corresponding simulation tables so that a users can update the values of the parameters and experimental conditions. In this thesis, the simulation experimentation environment is designed and implemented on the SUN/SPARC workstation with a XView graphic user interface (Heller 1991; Quercia and O'Reilly 1990). The developed Graphic User Interface (GUI) environment provides a user with a very efficient way to define or update the values of parameters and experimental conditions interactively. After each execution of a simulation run, a user can display the simulation results in different ways through the simulation environment in order to observe the behavior of the model.

In Chapter 2, the basic functions and characteristics of simulation environments are explained. An overview of simulation program generators is presented in Chapter 3. In Chapter 4, an overview of simulation experimentation environment is discussed. The structure of Gest models and functional decompositions of the simulation environment and the program generator are discussed in detail in Chapter 5. The conclusions and recommended future work are presented in Chapter 6.

2. Simulation Environments

In this thesis, the simulation experimentation environment featured by a Graphic User Interface is designed and implemented on SUN/SPARC workstation to support a user to carry out the simulation experiments based on the given model from the MaGest system (Aytaç and Ören 1986). A simulation experiment can be efficiently carried out by a user with a supportive simulation environment. Several functions are supported by the simulation environment called SimGest. The detailed discussions are in Chapter 5.

2.1 Simulation

In computerized simulation experimentation, the main purpose is to study the behavior of models. With valid models, simulation is often dramatically more cost-effective than are real experiments, which can be expensive, dangerous or impossible because a new system is not yet available. A simulation study includes a parametric model, sets of parameter values, simulation experiments, and simulation runs. Before and after every simulation run, prerun and postrun activities exist. Similarly, before and after a simulation study, prestudy and poststudy activities exist.

A model is a representation of a system (or object, or phenomenon). The model is called an adequate one if it is appropriate for the purpose (or goal) in the mind of the model builder. A model can be viewed as a simplification or idealization of the system (Murthy and Rodin 1987, p. 17).

The modelling aspect involves computer-assisted modelling and model-based management. A computer-assisted modelling environment provides guidance to the user in the specification of models. Model-based management aims at keeping, in a model base (i.e., model file) computer-processable models expressed in a model specification language. A simulation model is described by mathematical equations along with state variable initial conditions and the values of parameters. The mathematical equations determine the behavior of a model.

A model can be specified or programmed in a modelling environments. Modelling environments can be conceived as separate entities from simulation experimentation

environments to ease and enhance efficiency of several model-based and model quality assurance activities. The acceptability and the associated scope of a model generated or updated within a modelling environment can then be certified and a list of certification criteria utilized by the modelling environment can be given explicitly. Such a model and its acceptability conditions can then be passed to a simulation environment (Ören 1992, p. 9). The simulation environment allows a user to design and build the model specification or choose a model with a specification. Then the user can select a model in the high-level simulation specification through the simulation environment to carry out the simulation experiments.

The simulation model is represented in the computer by the derivative procedure, which repeatedly evaluates all defined-variable and derivative expressions to update state-variable time histories (Korn 1989, p. 9). The high speed of computation of modern computers has improved the speed of execution of simulations. Besides, the method of carrying out simulations can also raise the speed of the simulation experiments. Parallel simulation, for example, attempts to speed up discrete-event simulations by executing portions of the simulation in parallel. This is typically achieved by removing the global simulation clock and its associated event list and replacing them with a synchronization algorithm designed to insure that events are still executed in the proper sequence (Payne 1991, p. 51). Graphical interfaces for simulation development, input, monitoring while running, and output are important for activities such as menu-driven simulation setup, flexible output selection and formats, iconic programming, etc. Graphical interfaces do much more than simply produce three-dimensional geometries visually, they help to synthesize and interpret data available to the simulation user (Pace 1991, p. 3). The advent of object-oriented computer programming languages and the introduction of object-oriented methodologies in simulations have introduced new paradigms that can help achieve the software design goals. Object-oriented practices create a model of the real world that can be directly represented in software. Objects provide the data (attributes) for representing real world entities. Specific operations (methods) describe the processing associated with the data. Object-oriented programming strives to enforce good software engineering practices by providing libraries of reusable, modular, maintainable code that can be re-assembled for new applications (Denney 1991, p. 69).

Other simulation experiments are carried out with some other software engineering tools, such as using reverse engineering in the simulation life cycle (Birta et al. 1992, p. 69). In addition, the field of artificial intelligence in simulation which is also called cognizant, intelligent, expert, rule-based, or knowledge-based simulation (or simulation systems or environments) has gained enough impetus.

Simulation experimentation environments can help users in specifying values of model parameters, initial conditions of the state variables and simulation run conditions. Some other functions can also be specified by simulation experimentation environments.

2.2 Functions Supported by Simulation Environments

Simulation environments provide several types of computer assistance supporting and guiding a user in modelling, model-base management, specification of the simulation experiments, program generation based on high level specifications, execution and monitoring of simulation runs, other types of behavior generation, such as qualitative simulation, optimization, statistical and rule-based inferencing, as well as symbolic processing of models for model analysis and transformation. Different simulation environments have different emphasis. Some simulation environments mainly support a user to specify and evaluate the models. Some simulation environments support a user to carry out simulation based on a given model. They let a user modify the values of parameters as well as experimental conditions to produce the time history of the state variables. During the simulation experiments, simulation environments let a user update the value of parameter(s) and the number of simulation runs. The user can execute another simulation with updated information. Then the simulation environment processes the results of the state variable and displays the results in several ways that the environment supports. Some simulation environments have both abilities of modelling and execution for a simulation experiment. The user may define a new model with the specification and generate the programs for the execution of simulation experiments and then observe the new model behavior through a simulation environment.

2.2.1 Specification of Experiments

The experimental conditions consist of two groups of information, experimental frames and their application to (parametric model, model parameter set) pairs. An experimental frame defines a limited set of circumstances under which a system or a model is to be observed or subjected to experimentation. It is composed of five components:

(1) Observational variables, the variables can be any descriptive variable and not necessarily the output variables of a model.

(2) Admissible input segments, i.e., sequence of values of the input variables acceptable by the model.

(3) Initial settings of the state variables.

(4) Termination conditions, which may be specified explicitly in terms of the simulation time or the descriptive variables of the model.

(5) Specifications for collection, compression and display of simulation behavior.

A simulation study often investigates the different behavior of the model by giving different values of parameters, initial conditions of the state variables and the number of simulation runs. Normally the parameters, initial conditions and the number of simulation runs are defined in a model in a high-level language. Each time if one of these is changed, one has to go back to the model and make modifications and then execute the simulation experiments. However, as the model is in a high level language, the execution of the simulation experiment after each modification often needs re-compilation of the model. Sometime only a minor syntax error during the modification can cause many complications with the compilation. In this case, one has to debug the syntax error in order to carry on the simulation experiment, which is always a time consuming matter.

The experimental conditions can be defined or modified through the simulation experimentation environment. A user doesn't have to go back to the original model to do the modifications. The user can modify the values of parameters, initial conditions and the number of simulation runs through the simulation environment. The user can also define new parameter sets or number of simulation runs regardless of whether or not it is defined in the original model. Therefore the user can immediately carry on the simulation experiments after each modification of the simulation conditions, which greatly improves

the efficiency of the simulation experimentation.

2.2.2 Program Generation

To carry out simulation experiments on a computer with a given model, one has to create the simulation program which implements the model before any execution of a simulation run. A well designed simulation experimentation environment can support the program generation. The simulation experimentation environment may also have files which define a particular situation and contain enough information to generate the outline of a simulation program. Therefore by selecting a model and some functions of program generation supported by the simulation experimentation environment, one can obtain simulation programs that can be executed on the computer for the simulation experiments.

Program generation has brought simulation techniques within the bounds of confidence and competence for a large number of engineers and managers who have, in the past, been daunted by the problems of implementing a simulation model.

In the thesis, the program generation is executed through the simulation experimentation environment when a model is selected by a user. When a model is chosen, the simulation environment employs the program generator to generate the simulation program and put the routines in the program. The simulation program is generated in C program, which is compilable and ready to be run for the simulation experiments.

2.2.3 Monitoring execution of the program

A simulation environment allows a user to view the whole process of the simulation experiment. For instance, a user can check whether a model is successfully translated into an executable programming language after selection of the model. The user can further check if the generated code is successfully compiled. When the execution of the simulation experiment is over, a visual or audible signal is given so that the user is aware of the end of the simulation execution. Therefore, the user can either select the display of the results or modify the simulation conditions for next simulation run.

2.2.4 Processing model behavior

One can investigate the behavior of a model by observing the simulation results corresponding to the different values of parameter(s) and initial conditions. The model behavior can be shown in either tabular or graphic format. The former provides the detailed simulation results. However, the graphic format of results is more meaningful so the behavior of the model can be easily observed.

After each execution of a simulation experiment, the simulation environment stores all the results in the internal data files ready to be processed for display. A user can select any way that is supported by the simulation environment to display the simulation results. The process of displaying the results in a tabular format is to process the data stored in the files and then display the results on the screen. For the graphic display, the process controlled by the simulation environment takes different actions according to the selection of the display chosen by a user through the simulation environment. If only one state variable plot is selected, the simulation environment extracts the data of the selected state variable from the data files and plots it on the screen. Within the simulation environment, a user can display all the state variables as well as just a single state variable. If the simulation involves several simulation runs, the simulation environment can display both single run results and multi-run results of a state variable. The simulation environment is also able to change both the vertical and horizontal scales of the graphic display so that a user can observe certain parts of the graphic display in detail by expanding amplitude of the vertical or horizontal axis. The simulation environment provides the user a handy method to observe the model behavior.

2.3 Other characteristics of simulation environments

Simulation environments play an important role in modern simulation experiments. Well designed simulation environments do not only provide some types of computer assistance but also create some new methodologies for simulation experiments to improve the usefulness and efficiency of simulation. Other simulation environments have emerged with development of the computer hardware and software. There are some advancements in the modeling environments:

(1) Modelling environments for several types of modelling formalisms can be used in

trajectory and in structural simulation, although several of the modelling formalisms are not yet fully supported in the current modelling environments.

(2) Modelling environments for intelligent agents are for goal-directed systems, systems with perception abilities, and systems with several types of learning abilities would be desirable to specify such systems and to experiment with them in an associated simulation environment.

(3) Knowledge-based modelling environments can benefit from inclusion of built-in quality assurance techniques and certification ability of the models with respect to criteria which should be made available to the user in a readily understandable format.

(4) Cognizant (or intelligent) modelling environments may have cognitive knowledge processing abilities. Regardless of the terminology adopted (i.e., cognizant modelling environments or intelligent modelling environments), several aspects of advanced knowledge processing abilities can be part of the knowledge processing abilities of the modelling environments. Rule based inferencing is only one of the abilities of such systems. Therefore, possibilities other than rule-based inference exist for cognizant modelling environments.

(5) Model bases and model-based management systems, availability of high level specification languages and model bases of models expressed in terms of such high level languages would be very useful in several application areas. One practical and important case is being built for the building industry.

(6) Knowledge bases and the use of database. In addition to model bases, several types of knowledge bases to hold knowledge necessary as simulation life cycle activities would be useful. Some knowledge would be independent of the application domain and may cover modelling formalisms, or even different types of scientific and engineering knowledge. Domain dependent knowledge processing ability of the environment. For any knowledge base, one should also be concerned with its quality assurance. Knowledge bases of learning systems should especially be monitored for integrity of the knowledge and for avoidance of possible deterioration of the knowledge processing ability.

(7) Verification and validation are vital functions in a modelling and simulation environment. A good modelling environment should have rule-based or algorithmic tools to enhance such activities and to provide an explicit list of the criteria used.

Lists of rules necessary to check consistency of models with respect to modelling methodologies or other criteria should be available with any certification of acceptability of the models.

For simulation environments, a mixed simulation environment is a computer system where a reasonable subset of the simulation life cycle activities and other types of model-based knowledge generation activities (such as the ones in decision support systems, rule-based expert systems, optimization systems, or statistical inference systems) can coexist in a synergistic way. Mixed simulation environments include both type 1 and type 2 simulative systems, simulative expert systems and expert simulation systems. Simulative systems are systems developed for purposes other than simulation. Inclusion of simulation ability transforms the environment to a virtual laboratory where generated or existing models can be used for simulation purposes. Type 1 simulative systems are design systems which also have simulation abilities. An aspect of the model generated for design purposes can also be used to perform some simulation experiments. In type 2 simulative systems an existing model is used with updated and often on-line data for better decisions about a system such as a business, agricultural or ecological management systems, or a national economy. Simulative expert systems are expert systems having access to a simulation system. Expert simulation systems are simulation systems which benefit from an expert system.

A comprehensive simulation environment is a computer aided modelling and simulation system (CAMASS) which extensively supports simulation life cycle activities as well as mixed simulation activities and additional quality assurance activities needed by the non-simulation knowledge processing activities. In modelling and simulation, comprehensive systems can provide several needed and complementary knowledge processing abilities. Such systems need not be fully integrated systems. Several environments and tools can coexist and work in concert.

Availability of modelling and simulation tools as well as CASE (Computer Aided Software Engineering) tools in comprehensive environments will increase their efficiency. Some of the non-conventional tools include program understanding tools. CAST (Computer Aided Systems Theory) tools are special tools for simulation

environments and can provide the power of systems theory in simulation environments. CAST and systems theory provide valuable knowledge that can and should be used as a basis for advanced tools for modelling and simulation environments.

Integrated simulation environments is the concept that has been an important milestone towards the realization of integrative simulation environments.

Integrative simulation environments are closed systems; new tools or environments cannot easily be added. New architectures should be open to allow integration of new components.

In different engineering fields such as computer, electrical, or mechanical engineering, already several advanced design environments have simulation abilities. Future design environments should continue to have simulation abilities where simulation programs can automatically be generated from some aspects of the design. Simulation ability of such systems may allow designers to experiment with designs while expert component of the system may provide advice for desirable modifications.

System engineering coupled with the contemporary computational power is making a new debut as CBSE (Computer based systems engineering). A powerful CBSE environment does not yet exist. However, it is hoped that soon such an environment with abilities to provide a virtual laboratory to test the designs before implementation will be created.

The increased use of CASE tools and environments creates an interface problem which can be resolved by using a repository-based architecture simulation environments. For this integrative environment, repository-based simulation is a promising possibility (Ören 1992 p. 8-12).

3. Overview of Simulation Program Generators

A program generator is a program which accepts a simulation model and produces the equivalent computer program, either in a simulation programming language or in a general high-level language (Mathewson 1974, p. 181).

The simulation programming language (SPL) provides high-level concepts to help a user articulate the unique features of the models. Each of the simulation programming languages can be used to produce programs which are very different in structure and syntax and yet, when run, provide identical results by executing routines which make the necessary changes to the data structures representing the state of the components of the model (Davis 1979, p. 181). The simulation programming language has a skeletal control structure which dictates the questions the user should be aware of the situation, and the order in which those questions should be posed. The user should have some feel for the syntax and structure of the simulation programming language, even though the user may be using the program generator to avoid getting involved in the details.

The translated source programs for dynamic continuous simulation systems combine written simulation-run, integration, and output routines and some library routines such as function generation (Korn 1989, p. 14). The user then just needs to supply a main program defining the experiment and compile, link and run the complete program for the simulation experiment.

Discrete system simulation is concerned with the representation on a computer, of time-consuming activities which take place simultaneously or in parallel in the real world. The program generator is to construct character strings which are syntactically correct statements in the simulation programming language and into which have been inserted the names of the entities and queues which are unique to the particular model.

Program generators give a solution to the simulation experiment based on the model in the simulation programming language, yet a problem still remains. The problem is the time needed for program generation and regeneration after each program change. New numerical parameter values are accepted at once. But simulation programming language

processing, compilation, and linking impose delays every time the program for the model or for the experimental protocol is modified. These repeated delays interrupt an experimenter's train of thought. They are especially annoying if the compiler returns an error message after a delay. Furthermore, the user must check the input/output consistency. For different design objectives, the user has to repeat the process several times.

A specialized program generating system, able to deal with the data structures, would allow a much more flexible approach to the design process. Simulation program generation gives the advantage that ninety or more percent of the simulation program will be generated in a standard, syntactically correct form and guaranteed to run correctly (Davies 1979, p. 187).

4. Overview of simulation experimentation environments

Simulation is an indispensable tool for evaluating decision alternatives in complex design, control or analysis problems (Birta 1992, p. 69). Simulation experimentation environments provide assistance in supporting a simulation user to carry out simulation experiments. A desirable simulation experimentation environment should support a user from the first stage where models are defined to the final stage where all the simulation results are presented.

Simulation experimentation environments support a user in the modelling stage. A user may program models in a high level simulation language or a user may just use some dialogue windows within the environments to define models. In this case, a user doesn't even have to know the simulation language because the interactive dialogue windows provide enough information that the user is able to do modelling in an English-oriented conversation. During modelling some warnings or error messages are given if there is any improper input by the user.

MaGest88, a modelling and simulation environment, has been implemented on a SUN workstation. The MaGest system uses its knowledge to provide the user dynamic templates to fill in and to dynamically tailor the templates as new knowledge is added to the system during the course of the specification of a model. By using the system, a user has the ability to provide the specifications. Documentation of the models is not only done in a consistent way but also in a computer processable way. The occurrence of some types of errors is completely eliminated.

Models or systems under study can also be represented by the graphical formalism within a simulation environment. The formalism which gives rise to a network structure, provides an effective means of not only describing the problem but of revealing features of the problem that are difficult to appreciate from the program code. The network graph representation is constructed by the appropriate interconnection of a predefined set of node types. The program code for a given problem and the associated network graph representation have a direct correspondence that is made available through the simulation

environments. Like network graphs, the annotation templates provide an alternate representation for the program code. The templates provide a tabular, rather than graphical display which is designed to reveal specific structural and organizational aspects of the program. A statement template lists all occurrences of a selected statement type together with relevant information extracted from the statement arguments. Program templates are generated to show program-wide cross-reference information in a convenient form. These templates let a user view a particular aspect of the program at a higher level. Simulation environments support a user to select the statement type in three ways: network graph representation, a line numbered listing of the program code and a menu (Birta et al. 1992, p. 77-78).

Furthermore with a visual modelling tool embedded in the simulation experimentation environment, a user can define models without knowing which language is used. The visual modelling paradigm allows the user to both represent and study the behavior of a system graphically. One of the tools for doing this is available. "The basic idea of it is that the user inputs a model just as the user would describe it to a colleague: by drawing a picture. The purely graphical modelling paradigm is powerful and allows many complex situations to be easily represented without burdening the user with all the usual responsibilities and pitfalls of textual modelling." (Funka-Lea et al. 1991, p. 39).

Simulation experimentation environments should allow a user to access the model after modelling and interrupt the simulation experiment if it is necessary when the simulation experiments are running. Simulation experimentation environments have knowledge of the dynamic structure of the model which is necessary to generate model behavior, output variables and functions, input scheduling to represent to the model, values of model parameters, collection, reduction, and display specifications for simulation data, initial conditions, terminal conditions of a simulation run, and the number and the relationships of the runs (Wendt 1993). Query systems implemented in the simulation experimentation environments provide comprehensive assistance in the formulation of the queries, their execution, and analysis and interpretation of the results. Ideal simulation experimentation environments give a user the power to carry out simulation experiments in terms of flexibility and convenience. It is not compulsory for a user to know details about simulation experimentation environments. The simulation environments should fully

support the user in carrying out simulation experiments and documentation.

Within the environments the data dictionary can be used to provide the information for dynamic analysis with minimal additional work on the part of the user (Warren 1991, p. 295). When the expert-system is embedded in the environments, it can reason and evaluate the simulation results. Simulation environments first program models with a description of the system dynamics information. Then the environments let a user define simulation run parameter from the input screen. After simulation the environments keep the results of the most recent simulation runs. Meanwhile the context independent help, such as tutorials, definitions and formulae etc. are available from the environments. To evaluate the simulation results, a user can select simulation run parameter help, such as definitions and recommendations based on the most recent simulation run results. The system can interpret the results of most simulation runs and give feedback to the user with the supported expert-system in the environments.

A Graphic User Interface (GUI) can also set up a good simulation environment fulfilling many duties that are demanded for simulation experimentation. In this thesis, the simulation experimentation environment is designed and implemented with a GUI. Many features of the GUI provided for the simulation environment have enabled the simulation environment to be presented in a unique way. More detail discussion and presentation are in the following Chapter.

Some previous researches of simulation experimentation environment have been done. The work of Birta and So (1990) presents a database support environment for simulation experiments (NEMS). NEMS: A database support environment for numerical experimentation provides a general management function both for the setting up of experiments and for the data that is associated with the experiments. The management function which it provides is independent both of the specific nature of the experiments being carried out and of the language used to create the modules that participate in these experiments. The fundamental task of the NEMS is to provide a database support environment for manipulating the objects in terms of the values associated with their various attributes. A user can examine, summarize or tabulate the data of the simulation

experiments with the database support environment. The NEMS provides a productivity tool for researchers, software developers and system designers.

The work of do-Régo (1992) provides an environment (SimAd) for experimental design through which a user can do experimental design. There are two experimental design matrices available in the system: factorial design and fractional factorial design. Factorial designs can provide valuable assistance in understanding a complicated simulation model. Fractional factorial designs provide a way to get good estimates of only the main effects and perhaps two way interactions at a fraction of the computational effort. SimAd lets a user do experimental designs interactively by providing a factorial design matrix or a fractional factorial design matrix. After the design, the modules in the system find decisions for parameters and execute the simulation experiments. Once the execution of the simulation experiment is over, SimAd analyzes the results of the simulation experiment and then gives the user advice about the performance through the user interface in the environment. The user can therefore redesign the matrix to continue the simulation experiments. SimAd presents a simulation environment through which a user can efficiently design the matrix and check the effects on the model.

In this thesis, the system called SimGest is presented. SimGest provides a user friendly interactive simulation experimentation environment. A user can carry out simulation experiments using a Gest model within SimGest.

5. Gest System and SimGest

This chapter discusses functional decomposition of the Gest system, specification of simulation studies in Gest, the Gest analyzer and generators of programs and tables, computer-aided specifications, structure of the generated C program, computer-aided specifications and the display of the results.

5.1 Functional decomposition of the Gest system

Gest (General Systems Theory implementer) denotes a system theory-based modelling and specification language (Ören 1984) as well as the modelling and simulation environment to support it. The Gest environment, also called Gest System, consists of two parts: MaGest and SimGest (see Figure 5.1). The MaGest (Modelling Advisor for Gest) is a modelling environment and assists a user to define models expressed in the Gest specification language. MaGest is being developed as a separate project. SimGest is the simulation experimentation environment for Gest.

SimGest consists of three parts: an analyzer and a set of generators, a computer-aided specification module and display routines. The analyzer and generators consist of Gest analyzer, program generator and generator of simulation tables. The computer-aided specification module interactively updates or modifies the specification of parameters, experimental conditions and the simulation study. The display routines display the behavior of the simulation study.

Simulation tables are created by the generator of simulation tables for parameters, experimental conditions and simulation study. The parameter tables contain information about the values of parameters. The experimental condition tables hold information about the initial values of state variables and simulation global variables; i.e., initial, terminal and communication times and step size of the computation. The simulation study table involves simulation run conditions etc.

The simulation program (in C) is the generated source program for a selected model written in the Gest specification language. Its compilation produces an executable simulation program. It is executed to carry out the simulation experiments. These results

are processed by the display routines within the SimGest system to display the behavior of the model.

5.2 Specification of simulation studies in Gest

As seen in Figure 5.1, the Gest specification consists of two parts: the Gest model and an optional part. The Gest model has two parts: the static structure and dynamic structure. The static structure declares the elements of a model, such as input, state, output, and auxiliary variables, and constants, parameters, auxiliary parameters, etc. The dynamic structure defines the specification of the derivatives of state variables and computation of the necessary auxiliary variables. The dynamic structure also includes the specification of the output variables.

The optional part contains: parameter set(s), experimental conditions and the specification of simulation run(s). Each parameter set defines the values of the parameter(s) within the model. The simulation experimental conditions define the initial values of the state variables, time variables (such as initial, terminal and communication times). The simulation run condition specifies the combination of the parameter set, and the experimental conditions to be used in a simulation study. The optional part can be defined either in the original model written in the Gest specification language or defined by the user through the interactive simulation experimentation environment.

Figure 5.2 shows a typical Gest model of a continuous model in the Gest specification language. At the very beginning of the Gest model, the name of the model (cannonball) and the type of the simulation model (ContinuousModel) are defined. In the Gest model the static structure defines the state variables (x, xdot, y and ydot), constant (g), parameters (r, v0, and theta), and auxiliary parameters (ap1 and ap2). The dynamic structure defines four differential equations.

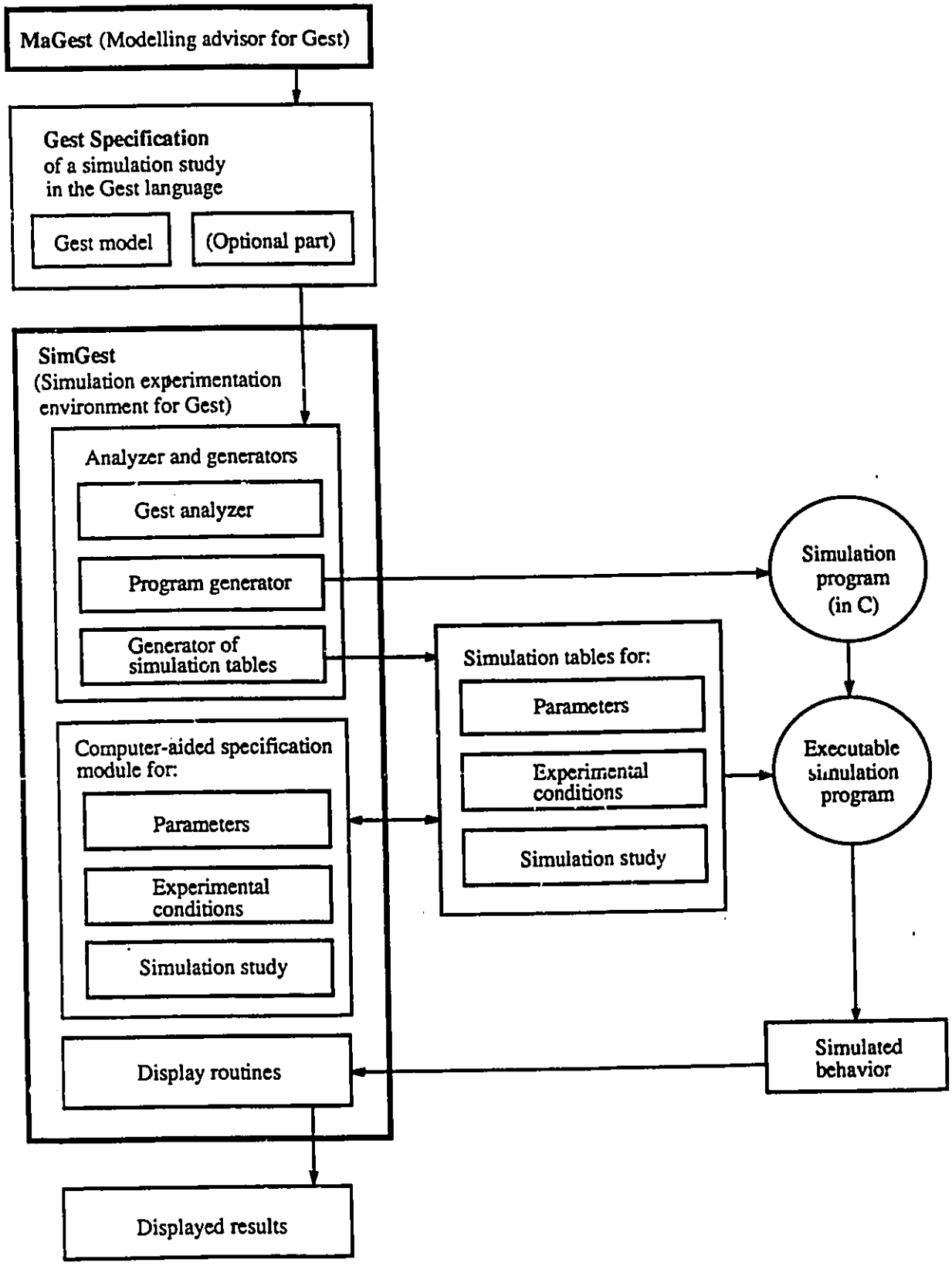

Figure 5.1 Overall Architecture of the Gest System and a Functional Decomposition of SimGest

The parameter set(s) of the optional part are shown in Figure 5.3. There are three parameter sets in this example. The integer number following the keyword ParameterSet distinguishes each parameter set. Each parameter set defines the values of the parameters for the set. The specification of simulation run uses the values of the sets by defining which parameter set is to be used in the structure of the simulation run conditions.

```
ContinuousModel cannonball
   StaticStructure
      States  x,xdot,y,ydot
      Outputs x,y
      Constants g
         g    = 9.81
      EndConstants
      Parameters r,v0,theta
      AuxiliaryParameters ap1,ap2
         ap1  = v0*cos(theta)
         ap2  = v0*sin(theta)
       EndAuxiliaryParameters
    EndStaticStructure

   DynamicStructure
      Derivatives
         x'    = xdot
         xdot' = -r*sqrt(xdot*xdot+ydot*ydot)*xdot
         y'    = ydot
         ydot' = -r*sqrt(xdot*xdot+ydot*ydot)*ydot-g
      EndDerivatives
   EndDynamicStructure
EndModel cannonball
```

Figure 5.2 Structure of Gest model of a Typical Example
Continuous Cannonball Model in Gest

Figure 5.4 shows the structure of the experiment set. Each experiment set in the example defines the simulation times and initial conditions of the state variables of the model defined in the dynamic structure. As seen in the figure, the experiment set contains two parts: global variables (Global) and initial conditions of state variables (InitialState). The global variables include time variables, the integration method and step size. The initial

conditions define the initial values of state variables. Similar to the parameter set, the number following the keyword Experiment is used to identify each experiment set.

```
ParameterSet 1
   Model cannonball
      Parameter
         r     = 0.000075
         v0    = 900
         theta = 0.1745
      EndParameter
   EndModel cannonball
EndParameterSet 1

ParameterSet 2
   Model cannonball
      Parameter
         r     = 0.000075
         v0    = 900
         theta = 0.707
      EndParameter
   EndModel cannonball
EndParameterSet 2

ParameterSet 3
   Model cannonball
      Parameter
         r     = 0.000075
         v0    = 900
         theta = 1.4
      EndParameter
   EndModel cannonball
EndParameterSet 3
```

Figure 5.3 Structure of the Parameter Set of the Cannonball Model

Having defined parameter set(s) and experiment set(s), the specification of simulation run defines the simulation run conditions. Figure 5.5 illustrates the structure of the simulation run. In this example, there are three simulation runs. Each simulation run defines the simulation run conditions: combination of parameter set (WithParameterSet) and experimental conditions (InExperiment). Since the parameter set contains the values of the parameter(s) and experiment set holds the initial conditions and simulation times, therefore, the simulation run structure actually defines each simulation run conditions

with the values of the parameter(s) and experimental conditions to be used in the simulation studies. For instance, the first simulation run (Run 1) specifies the values of parameter(s) defined within parameter set (WithParameterSet 1), and the simulation experimental conditions defined in experiment set (InExperiment 1). Hence, by choosing a different simulation run, the simulation experiments can be carried out with different simulation run conditions.

```
Experiment 1
   Global
      TimeUnitIs second
      IntegrateBy RungeKutta
      StepSize = 0.1
      Communicate AtEvery 1 second
      SimulateUntil Time = 30
   EndGlobal

   Model cannonball
      InitialState
         x    = 0
         xdot = ap1
         y    = 0
         ydot = ap2
      EndInitialState
      SaveOutput x,y
   EndModel cannonball
EndExperiment 1
```

Figure 5.4 Structure of the Experiment Set of the Cannonball Model

Most often the optional part of a model will not be given. In this case, a user has to define the parameter set(s), experimental conditions and the specification of simulation run(s) in order to carry out the simulation experiment. The interactive simulation experimentation environment provides a user with a facility to define the optional part as is discussed in the later section 5.4.

```
Run 1
   ToObserve cannonball
      WithParameterSet 1
      InExperiment 1
EndRun 1

Run 2
   ToObserve cannonball
      WithParameterSet 2
      InExperiment 1
EndRun 2

Run 3
   ToObserve cannonball
      WithParameterSet 3
      InExperiment 1
EndRun 3
```

Figure 5.5 Structure of the Simulation Run of the Cannonball Model

5.3 Gest analyzer and program and table generators

As seen in Figure 5.1, The analyzer and generators consist of the Gest analyzer, the program generator and the generator of simulation tables.

In order to carry out the simulation experiment using a given model in Gest, the model must be translated into an executable program expressed in a programming language that can be run on a computer. The Gest analyzer analyzes a Gest model as an input of the SimGest and obtains the information about the specification of simulation conditions. Based on the information, the C program generator translates the Gest model and generates the corresponding simulation program in the C program language. At same time the generator of simulation tables generates simulation tables which contain necessary data for parameters, experimental conditions and simulation study. These tables are accessible and used by the executable simulation program later on during the simulation experiment in order to obtain the simulation conditions. The executable program is table-driven, therefore, by changing the contents of the simulation tables, the execution of simulation run can be carried out immediately without re-generating and re-

compiling the simulation program. The generated simulation tables are also accessed by the computer-aided specification module. Hence, the simulation conditions can be updated with the assistance of the SimGest system.

5.4 Simulation tables

The simulation tables are generated to store the information of the simulation conditions. The simulation conditions specified in the optional part of a model in Gest can be divided into: (1) Values of parameter(s), (2) Initial conditions of state variables, (3) Simulation times, and (4) Simulation run conditions. Therefore, the generator of simulation tables generates corresponding four types of internal tables to support table-driven simulation program in C. For a certain type of tables, the information in each set of the optional part is stored in the tables indexed by the corresponding set number.

Figure 5.6 shows detailed relationships of the simulation tables with the executable simulation program and computer-aided specification within the SimGest. The information of parameter(s) is contained in the generated tables parameter(s). The values of the parameter(s) defined in the parameter set N are stored in the table parameter(s)_N. Similarly, the information of initial conditions of state variables is put into the tables initial_conditions. The initial conditions specified in the experiment set M are in the table initial_conditions_M. This is also true for the information of simulation times. Tables global hold the simulation times. The information of specification of simulation run is written in the tables simulation_run. Each table of simulation_run contains the pointers to other tables containing the information of parameters, initial conditions and simulation times. Table simulation_run_P contains the information of the set Run P.

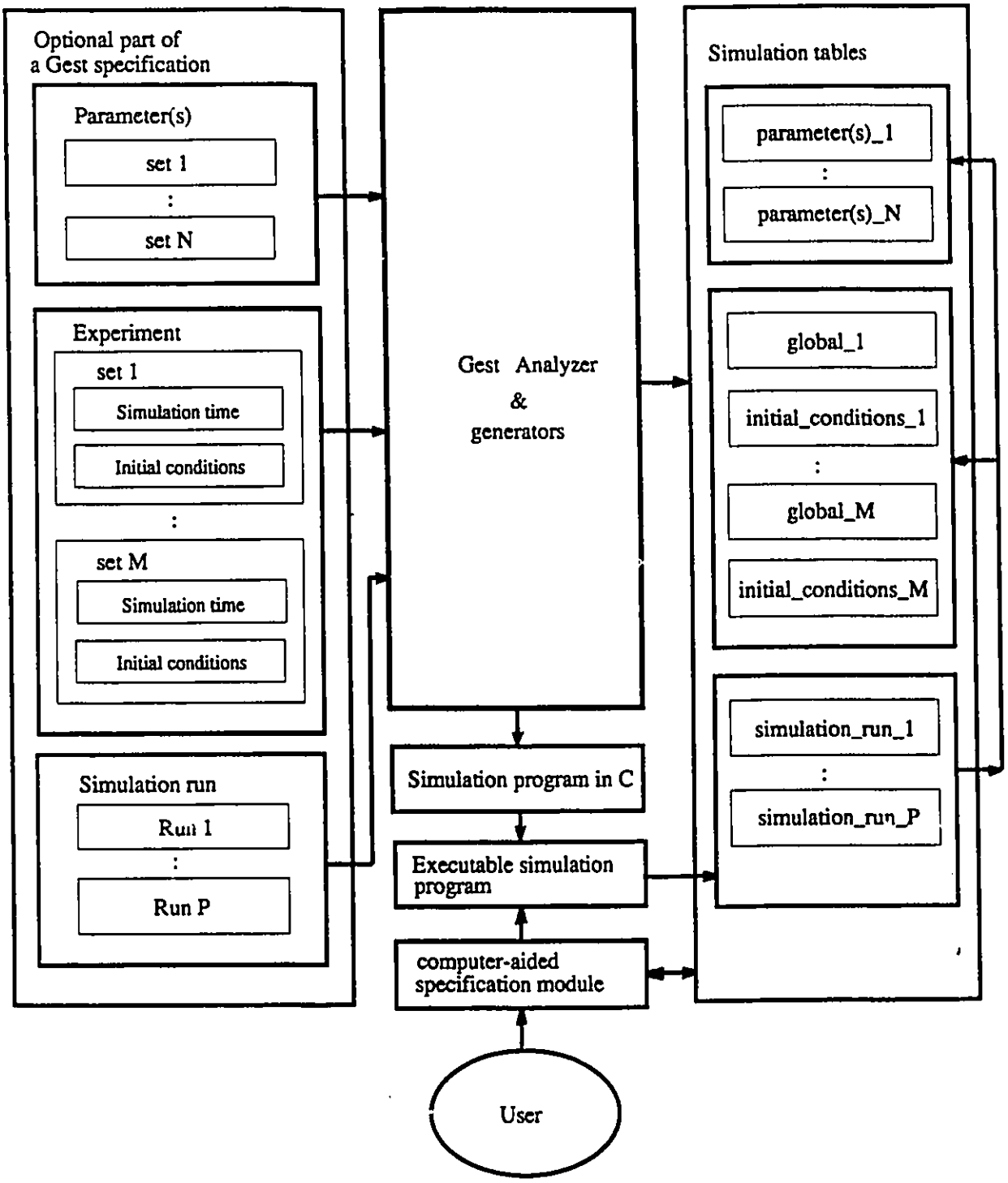

Figure 5.6 Relationships between a Model in Gest and the Simulation Tables

As seen in the Figure 5.6, when the execution of simulation run is carried out, the executable simulation program first opens the tables of the simulation_run to obtain the pointers to the tables for: parameter(s), simulation times and initial conditions. The executable simulation program then uses these pointers to each table to acquire all the information for the simulation experiment. The values of the parameters are obtained by reading the tables parameter(s). The initial conditions of state variables are obtained by reading the contents of the tables initial_conditions. The information of simulation times is known by reading the tables global. Therefore, the executable simulation program gets all the information about the model necessary for the simulation experiments.

As mentioned before, these simulation tables can also be updated by a user by the computer-aided specification within the SimGest. If the optional part of a model in Gest is not given, the simulation environment generates all the tables to hold the information specified by a user within the SimGest.

5.5 Structure of generated C program

A C program can be composed of several modules because of the characteristics of modularity. Therefore, the generated C program from the model in Gest is formed with more than one modules. Some of the modules are pre-created and stored in the files under user selected directories. These modules include some regular routines to support the C program when the simulation run is carried out. The C program is able to use the routines by including them in the header file. Whereas some of the modules are generated when the model is analyzed because these modules are different from one model to another. After analysis of the model in Gest, the program generator generates the complete program by putting all the header files containing the routines and created modules together to form the C program which accesses the created simulation tables for the simulation conditions when the program is run.

To provide better understanding of the generated C program, Figure 5.7 shows an overview of the structure of the generated C program.

```
#include <system header file>                    /* See Figure 5.8            */
#include "generated header file"                /* See Figure 5.8 (last line) */
global declaration part                         /* See Figure 5.9            */
main()
{
routines();                                     /* use functions of routines */
}

routines()                                      /* to be called by main to
                                                   perform functions         */
{
    functions of routines;
}
```

Figure 5.7 Overview of the Structure of the C Program Generated by the Program Generator

All the header files are both system supported header files (with sign < >) and SimGest created and stored header file (with sign " "). The system supported header file contains general purpose C functions. The SimGest created header file has specific functions that the user requires for special purpose. The variables, parameters and constants of a Gest model are declared in the declaration part. All the routines are defined in main and are called when the program is executed.

Figure 5.8 shows the header file of the generated C program. The header file is automatically included in the C program when it is generated. The file "LIB/method.c" contains the supporting routines for the simulation experiments and is generated by the SimGest system and is defined in the directory LIB.

```
#include  <stdio.h>                    /* generated by the C system */
#include  <ctype.h>                    /* generated by the C system */
#include  <string.h>                   /* generated by the C system */
#include  <math.h>                     /* generated by the C system */
#include  "LIB/method.c"               /* generated by SimGest      */
```

Figure 5.8 Header File of the Generated C Program

Figure 5.9 is a declaration part in the generated C program for the model of the cannonball. In general, the declaration contains the variable(s), constant(s), parameter(s), and auxiliary parameter(s) that are defined in the static structure of a given Gest model.

```
#define   m  4                       /* number of state variables */
#define   S$ "cannonball"            /* name of the model         */
#define   T$ "second"                /* unit of simulation time   */
static    float   ti, tf, tc, h0;    /* experimental conditions   */
static    float   r;                 /* parameter in the model    */
static    float   v0;                /* parameter in the model    */
static    float   theta;             /* parameter in the model    */
static    float   g;                 /* constant in the model     */
static    float   ap1;               /* auxiliary parameter       */
static    float   ap2;               /* auxiliary parameter       */
static    char    *var_name[m] =     /* variables in the model    */
            {
              "x",                   /* state variable x          */
              "xdot",                /* state varibale xdot       */
              "y",                   /* state variable y          */
              "ydot",                /* state variabel ydot       */
            };
```

Figure 5.9 Declaration of the Generated C Program for the Model Cannonball

As we have noted previously, the header file and declaration part of the generated C program is the translation of the Gest model of a simulation model expressed in Gest specification language. Therefore, the declaration part is always generated regardless of whether or not the optional part is defined in the model. However, the declaration part varies from one model to another because the number and the name of the parameter(s), constant(s), auxiliary parameter(s), and state variables are dependent on individual model.

When the optional part is given in the model, each part in the optional part is translated and stored in the simulation tables. The information of the parameter(s), initial conditions of state variables and specification of simulation run(s) are stored in the tables named parameter(s), initial_conditions and global, and simulation_run, respectively. Having the information of the optional part in the simulation tables, the program generator generates a routine as a part of the generated simulation program to access all the required simulation tables and obtain all the necessary information when the simulation experiments are carried out. Therefore, by calling the routine, all the information about the simulation conditions can be obtained for the simulation experiments.

If the optional part is not defined in the model, the computer-aided specification module allows a user to define the values of parameter(s), experimental conditions and simulation study. After a user defines all these simulation conditions, the simulation environment within the SimGest creates the simulation tables to store the information which is also accessed by the routine in the generated C program. Therefore, even if the optional part is not defined in the model, this routine is still generated in order to obtain the simulation conditions defined by a user through the simulation experimentation environment.

Figure 5.10 illustrates the algorithm of the main part of the generated C program. As seen in the figure, the header file, global declaration and main part are three major parts. Within the global declaration part, in addition to the declaration of the parameters, state variables, auxiliary parameters and constants etc. (line 6 to 14), the file pointers to the files containing the simulation tables are also declared (line 15) in order to get the contents of the simulation tables. The buffers are declared to store the names of the files (line 16). The variable run number is declared to hold the number of simulation run (line 17).

```
1    header file
2        system header file
3        generated header file
4    end header file

5    global declaration part
6        number of state variables
7        name of model
8        unit of simulation time
9        variables for computation
10       variables of simulation times
11       name of parameters
12       name of constants
13       name of auxiliary parameters
14       name of state variables
15       file pointers
16       buffers for the name of files containing results for displaying
17       variable of run number defined in Gest specification
18   end global declaration

19   main
20       declaration part
21           file pointer
22           constants in the model
23       end declaration

24       if the file containing number of run doesn't exist
25         then print error message
26       end if
27       get the number
28       close the file

29       loop until run index reaches the run number
30           make name of the file to store simulation results of table format
31           make name of the file to store simulation results for plotting
32           if opening of  the files is not successful
33             then print error message
34           end if
35           call routine input to access simulation tables
36           call routine rk4 to do Runge Kutta computation
37           close the file
38       end loop
39   end main
```

Figure 5.10 Algorithm of the Generated C Program: Header File, Global Declaration and the Main Part

The main part (from line 19) first gets the total number of simulation run (line 24 to 28). Each simulation run condition associated with corresponding information of initial conditions of state variables, values of parameters and simulation times is stored in the simulation run table. The loop (from line 29) indexed by the number of simulation run is to obtain all the associated simulation information by calling routine input (line 35) for each run. When the information is obtained, routine rk4 (line 36) is called to do Runge Kutta computation and store the results in the specified files, which are processed by the display results part in the SimGest to study the behavior of the model. The actual generated C code for Figure 5.10 is listed in Appendix 1.

The algorithm of the generated routine input called by the main part to access the simulation tables is shown in Figure 5.11. When the routine input is called by the main part, it makes a name of file containing the pointers to the tables of parameters, initial conditions and simulation times (line 13). Then it opens the file to get all the pointers (line 17 to 19). Having the pointers to the tables, the routine input opens the table of parameters to get the values of parameters (line 24 to 26). With the obtained values of parameters, the values of auxiliary parameters are given (line 27 to 28). Similarly, the routine obtains the simulation times (line 30 to 34) and initial conditions (line 35 to 38). The initial conditions of state variables associated with auxiliary parameters are given (line 39 to 40). Appendix 2 is the actual generated C code of the routine input.

```
1     routine input
2       declaration part
3         file pointer to simulation table
4         name of file containing simulation run conditions
5         name of file containing parameters
6         name of file containing simulation times
7         name of file containing initial conditions
8         buffer for the title of display
9         integer variable
10      end declaration

11      make title for display
12      put the title into buffer
13      make name of file containing simulation run conditions
14      if the file doesn't exist
15        then print error message
16      end if
17      get the file pointer to file parameter(s)
18      get the file pointer to file initial conditions
19      get the file pointer to file simulation times
20      close the file
21      if the file containing the parameters doesn't exist
22        then print error message
23      end if
24      get name and values of first parameter
25      get name and values of second parameter
26      get name and values of third parameter
27      obtain value of first auxiliary parameter
28      obtain value of second auxiliary parameter
29      close the file
30      if the file containing the simulation times doesn't exist
31        then print error message
32      end if
33      get the simulation times
34      close the file
35      if the file containing the initial conditions doesn't exist
36        then print error message
37      end if
38      get all initial conditions
39      assign first auxiliary parameter to a state variable as initial value
40      assign second auxiliary parameter to a state variable as initial value
41       close the file
42    end routine
```

Figure 5.11 Algorithm of the Generated Routine to Access the Simulation Tables

In the main part of generated C program, the routine rk4 is called to do Runge Kutta computation. Figure 5.12 is the algorithm of the routine rk4. The routine initializes all the variables used in the computation (line 6) and uses the step size based on communication time (line 7) to do Runge Kutta computation (line 8). Then it does loop until the time reaches the simulation terminal time (line 9). Within the loop, for each time, the routine derive is called to do numerical computation (line 10) and put the results for tabular display (line 11) and plotting (line 12) in specified places. After finishing all the computation, the routine loads all the data to files (line 14 to 15). The actual generated C code for the routine is listed in Appendix 3.

```
1     routine rk4
2        declaration part
3           Runge Kutta computation workplace
4           boundary of calculating values
5        end declaration

6        initialization
7        define the step size based on communication time
8        Runge Kutta computation
9        while loop until time reaches the terminal time
10          call routine derive for the values of each state variable
11          get data for table format display
12          get data for results plotting
13       end while
14       create data files for table listing and curve plotting
15       load data to files
16    end routine rk4

17    routine derive
18       declaration part
19          variable time
20       end declaration

21       time value
22       value of first state variable
23       value of second state variable
24       value of third state variable
25       value of fourth state variable
26    end routine
```

Figure 5.12 Algorithm of the Library Routine for Runge Kutta Computation

5.6 Computer-aided specifications

In the simulation experimentation, the values of the parameters, experimental conditions and simulation study for a model are frequently modified so that the model behavior with different values of parameter(s), experimental conditions and simulation run conditions can be studied. In some other cases, the values of parameters, experimental conditions and simulation run conditions for simulation experiments may not be given in the original model. Until all of these are defined, the simulation experiment can not be carried out.

The developed computer-aided specification module within the SimGest enables a user to interactively define and update simulation conditions. In SimGest, whether or not the simulation conditions are defined, the computer-aided specification module lets a user modify or define the values of the parameters, experimental conditions and simulation run conditions without necessarily modifying a model in Gest specification language. Therefore, there is no need to re-generate the simulation program and re-compile the generated simulation program. The simulation experiments can be immediately carried out.

5.6.1 Specification of parameters

One can study the behavior of a model by carrying out simulation experiment with the updated values of parameters in the model and observing the effect of the values of the parameters on the model. For a model expressed in Gest specification language, the specification of parameters is specified in the parameter set of the optional part. Frequently, the specification of parameters is not given in a model or needs to be updated in order to observe the behavior of the model.

Regardless of the specification of parameters is defined in the model, the computer-aided specification module for parameters in the SimGest not only permits a user to modify the specification of parameters and but also to define the parameter set. This capability allows the simulation experiment to be carried out with different specification of parameters. The computer-aided specification module for parameters lets a user easily display and modify the parameters for a selected parameter set. It also allows a user to

define values of parameters for a parameter set.

Figure 5.13 shows a developed interface window for specification of parameters in SimGest. When the window is selected by a user within the SimGest, the computer-aided specification module for parameters puts the names and values of parameters for a specified parameter set on each field by accessing the simulation tables of parameters. If any value of parameter is changed or updated by a user, the specification module for parameters replaces the values of parameters in the table of parameters with the updated data. Therefore, a user can easily update the values of parameters.

If no parameter set is defined in the model, the specification module for parameters provides same pop-up window but with blank in the value field to let a user define the values of parameters. Once the values of parameters are defined by a user, the specification module creates simulation parameter table indexed by the number of parameter set specified by a user and stores the values of the parameters in the table. If there is no parameter in the model, the specification module for parameters gives a warning message when a user attempts to display or modify the values of parameters. Figure 5.14 illustrates the algorithm of displaying the information of parameters on the interface window as a part of the computer-aided specification module for parameters.

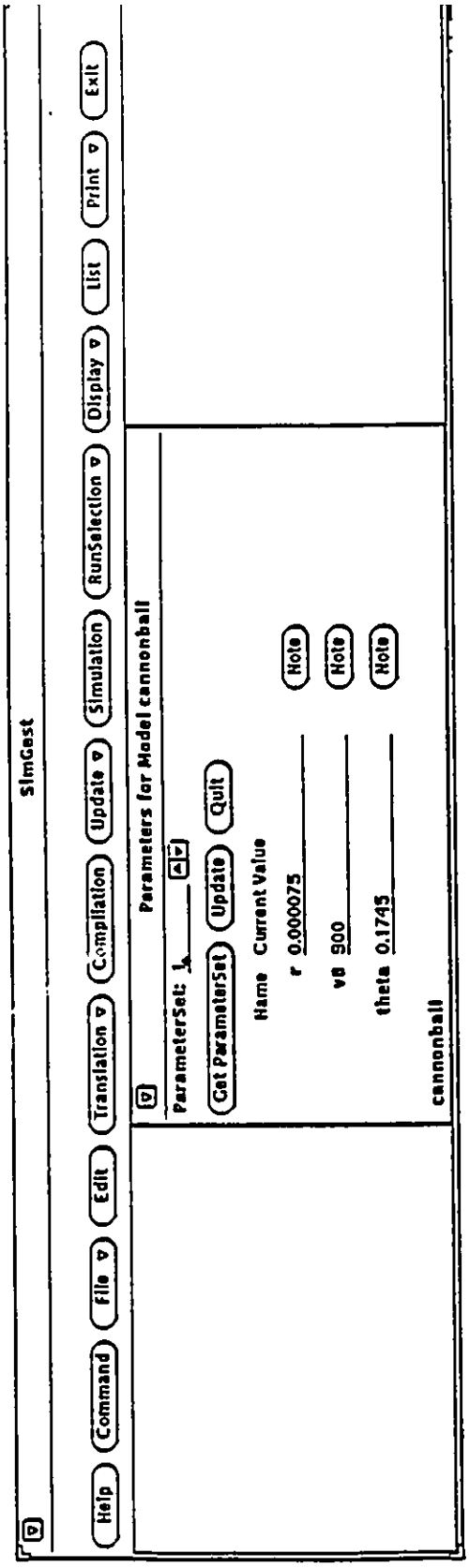

Figure 5.13 Interface Window for Specification of Parameters

```
1     routine display_parameters
2       declaration
3         variable for number of parameters
4         variable for number of parameter set
5         flag for parameters
6         buffer for title of window
7       end declaration
8       obtain title of window for a model
9       if no parameter exists in the model
10        then warning message
11        else if parameter in parameter set is defined
12                then while (index of parameter number <= total parameter )
13                       call routine fill_parameter to obtain values of
                           parameters in the simulation tables
14                     end while
15                     set flag for parameters true
16                     call routine def_para create panels for parameters on the
                         window
17                     call routine list_defined_par to put values of parameters
                         on the window
18              else set flag for parameters false
19                     call routine def_para to crate panels for parameters on
                         the window
20                     call routine list_blank_par to put blank for parameters
21            end if
22            make width of the window
23            make height of the window
24            show the window
25      end if
26    end routine

27    routine fill_parameter
28      declaration
29        variable for number of parameter set
30        file pointer
31        buffers
32      end declaration
33      get pointer to the parameter simulation tables corresponding to the
          number of parameter set
34      open the table
35      loop until index equals the total number of parameters
36        get name and value of a parameter
37        put the name in the buffer
38        put the value in the buffer
39      end loop
40      close the table
41    end routine
```

Figure 5.14 Algorithm of Displaying Parameters as a part of Computer-aided Specification Module for Parameters (Continued)

```
42    routine def_para
43       declaration
44          flag for parameter set
45          variable for number of parameter
46          window parameter
47       end declaration
48       if flag for parameter set is true
49         then set the number of parameter set on the window
50       end if
51       if panel of the window not created
52        then while (index for parameter number < total parameter)
53                  create panel
54              end while
55       end if
56       set flag indicating panels created
57    end routine


58    routine list_defined_par
59       declaration
60          variables for number of parameter and parameter set
61       end declaration

62       while (index number of parameter < total parameter)
63          put name of parameter on the window
64          put value of parameter on the window
65       end while
66    end routine


67    routine list_blank_par
68       declaration
69          variable for number of parameter
70       end declaration

71       while (index number of parameter < total parameter)
72          put name of parameter on the window
73          put blank for value of parameter on the window
74       end while
75    end routine
```

Figure 5.14 Algorithm of Displaying Parameters as a part of Computer-aided Specification Module for Parameters

When the specification module for parameters calls the routine to display the values of parameters, the routine immediately checks whether the model has any parameter (line 9). If there is at least one parameter in the model, the routine then check if the value of parameter is defined in parameter set (line 11). If yes, the routine goes to loop (line 12)

until it hits the total number of parameters. For each loop, the routine fill_parameter is called in order to obtain the values of all parameters (line 13). After the flag for defined parameter is set true (line 15), the routine def_para is called to create panels on the window to display the parameters (line 16). Then the routine list_defined_par is called to put the obtained values of parameters onto the window (line 17). If there is no parameter defined in the model, the flag for the defined parameter is set false (line 18). The routine def_para is still called (line 19) to create panels on the window in order to create a window with blank on the parameter value field (line 20), with which a user can define values of parameters.

When the routine fill_parameter (line 27) is called, the number of parameter set is passed to it. Therefore, the routine fill_parameter gets pointer to table which contains the values of parameters corresponding to the number of parameter set (line 33). By doing loop, the values of parameters for the parameter set are obtained (line 35 to 39).

The routine def_para (line 42) is called to create panels for the window displaying the parameters. It also checks whether the panels have been created (line 51). If panels have not been created, it creates panels (line 52 to 54) and set flag indicating that the creation of panels is done in order to avoid re-creation of panels (line 56).

The routine list_defined_par (line 58) is called to display the defined parameters on the window. The routine list_blank_par (line 67) is called to display a window on which a user can define values of parameters.

The actual C programs for the Figure 5.14 is listed in Appendix 4. The algorithm of updating values of parameters in the specification module for parameter is similar to the algorithm discussed above. For the parameter display, the specification module for parameters takes the values of parameters from the simulation tables for parameters and display them on the window. Whereas, for the updating parameters, the specification module for parameters gets the values defined by a user on the window and store them in the simulation parameter tables.

5.6.2 Specification of experimental conditions

In addition to updating the values of the parameters to study the behavior of a model, it is also desirable to observe the behavior of a model with different experimental conditions. It can be realized by modifying initial conditions of the state variables as well as simulation times. By carrying out the simulation experiment with the different initial conditions of state variables and simulation times, one can learn the behavior of a model with respect to the experimental conditions.

The experimental conditions in a model in Gest language include two parts, initial conditions of state variables and simulation times. The simulation times include communication time, calculating step size, simulation starting and terminating time. Experiment set in a model in Gest defines the experimental conditions. The computer-aided specification module for experimental conditions supports a user to define or modify initial conditions of state variables and simulation times. Similar to the parameters, the computer-aided specification module for experimental conditions employs a interface window as well. Figure 5.15 shows the window supporting the specification of experimental conditions for the model cannonball.

As seen on the window, there are four parts: initial conditions, constants list, auxiliary parameter list and time. With the window, a user can view all the information about experimental conditions as well as the constants and auxiliary parameters which may have been associated with the experimental conditions. A user can change any value of initial conditions and simulation times. When any value on the window is changed or updated, the computer-aided specification module for experimental conditions gets the value and updates the corresponding simulation tables. Therefore, when a simulation experiment is carried out, the updated information of experimental conditions will be used for the simulation run. Figure 5.16 illustrates the algorithm of displaying the information of initial conditions on the window as a part of the computer-aided specification module for experimental conditions in the SimGest.

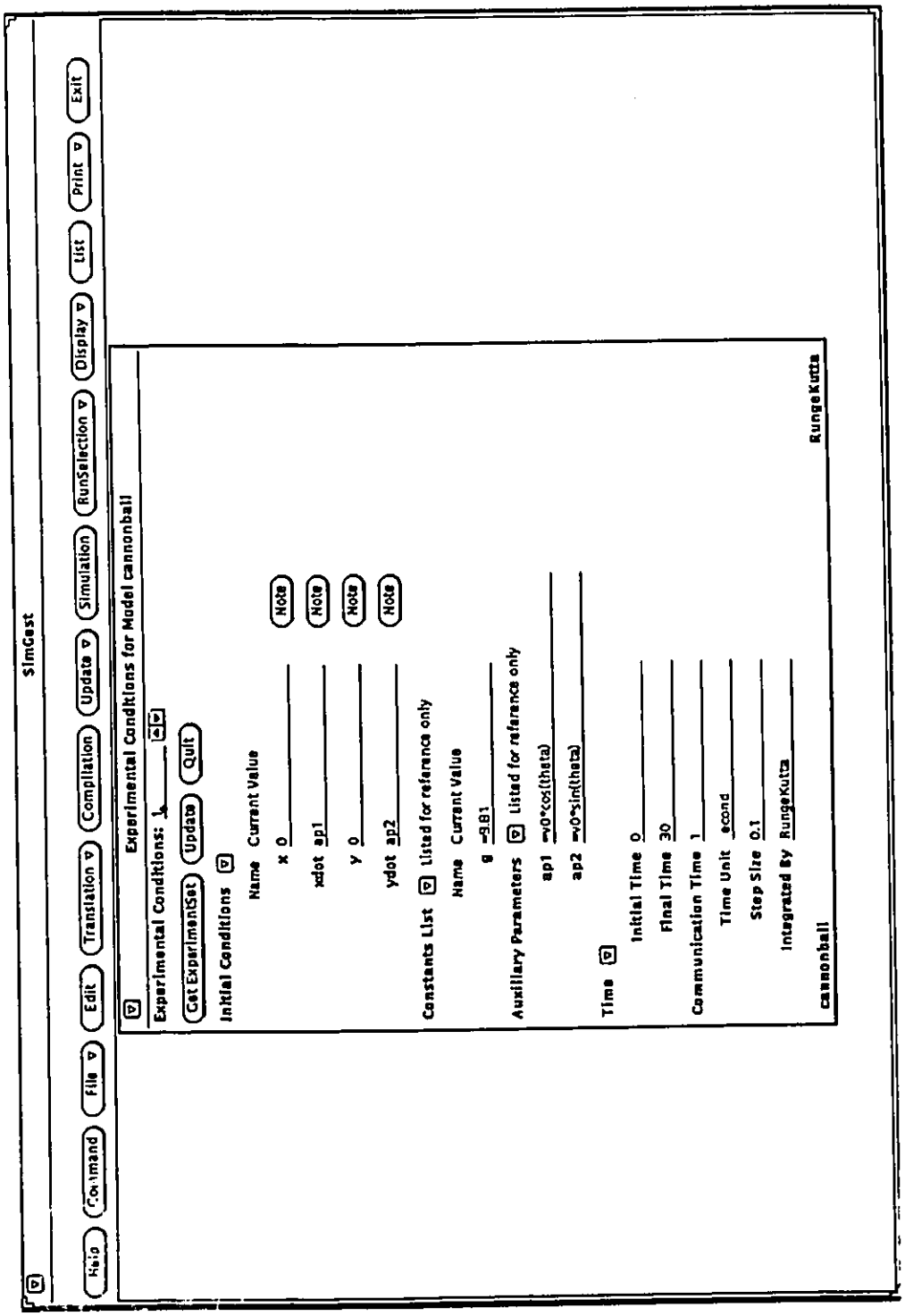

Figure 5.15 Interface Window for Specification of Experimental Conditions

```
1     routine display_initial_conditions
2       if the selection of displaying initial condition is true
          then call routine var_create to create panels for initial conditions
3               if the initial condition is defined in the model
4                 then call routine initialization to initialize
5                       call routine get_var_info to get initial conditions
6                 else call routine get_undefined_var to put blank for initial
                       conditions
7               end if
8         else close the initial conditions display on the window
9       end if
10      make width of the window
11      make height of the window
12    end routine

13    routine var_create
14      declaration
15        variable for number of state variables
16      end declaration
17      while (index of state variables < total state variable in the model)
18        create panel for variable on window
19      end while
20    end routine

21    routine initialization
22      declaration
23        file pointer
24        buffer for file name
25        variable for number of state variable
26      end declaration
27      get file name for simulation table for initial conditions
28      call routines to obtain initial conditions
29      set flag indicating initialization is done
30    end routine

31    routine get_var_info
32      declaration
33        variable for number of state variable
34        variable for number of experiment set
35      end declaration
36      get number of experiment set
37      while (index of number of state variable < total state variable)
38        put name of state variable on window
39        put obtained initial conditions on window
40      end while
41    end routine
```

Figure 5.16 Algorithm of Displaying Initial Conditions as a Part of Computer-aided Specification Module for Experimental Conditions (Continued)

```
42    routine get_undefined_var
43        declaration
44            variable for number of state variable
45        end declaration
46        while (index of number of state variable < total state variable)
47            put name of state variable on window
48            put blank for initial conditions
49        end while
50    end routine
```

Figure 5.16 Algorithm of Displaying Initial Conditions as a Part of Computer aided Specification Module for Experimental Conditions

When a user selects a window to display initial conditions in order to view or modify the experimental conditions, the computer-aided specification for experimental conditions calls the routine display_initial_conditions (line 1) to perform its functions. After checking the display for initial conditions of state variables is selected, the routine then calls the routine var_create to create panels to display the initial conditions on the window (line 2). Then the routine display_initial_conditions checks if the initial conditions are defined in a model (line 3) so that it can get names of state variables (line 4) and obtain all the information from the simulation tables for initial conditions and put them on the window (line 5). If the initial conditions are not defined in a model, the routine lets a user define the initial conditions by providing a same window with the state variables but values (line 6). With that 'blank' window, a user can define the initial conditions by filling up the field for the values of state variables.

Routine var_create (line 13) is called to create panels for each state variable on the window (line 17 to 19) to show the values on the window. Routine initialization (line 21) is to get the name of state variables from simulation tables for initial conditions (line 27) and then uses other supporting routines to obtain the initial values of the state variables (line 28). Then it sets flag indicating that initialization is done (line 29). Routine get_var_info (line 31) is called to put the obtained initial conditions on the window (line 37 to 40). Whereas routine get_undefined_var (line 42) is called to give a blank window to a user (line 46 to 49).

The actual C code for the algorithm of displaying initial condition for the computer-aided specification module for experimental conditions is listed in Appendix 5. The algorithms of displaying constants, auxiliary parameters and simulation times on the window are similar to that of initial conditions.

When the modification of experimental conditions is selected, the computer-aided specification module for experimental conditions calls another routine to perform as the routine display_initial_conditions does but puts the information on the window defined by a user to corresponding simulation tables. These tables are accessed by the executable simulation program. Therefore, the simulation experiments can be carried out with the newly updated or defined experimental conditions.

5.6.3 Specification and execution of simulation studies

Having the values of the parameters and experimental conditions in the model, one can carry out the simulation experiments. However, in order to study the behavior of a model effected by parameters or experimental conditions one may want to carry out the simulation experiments with different values of the parameters and experimental conditions. In a model in Gest, the simulation study is defined in the Run set in which the defined combination of parameter sets and experimental conditions are specified as a simulation run condition. If one wants to carry out the simulation experiment with different values of parameters and experimental conditions one has to define all the Run sets in a model. That may make a model program too long and take longer to generate and compile the simulation program.

The SimGest supports a user to define or modify the specification and execution of simulation studies. The computer-aided specification and execution of simulation studies lets a user select any available combination of parameter set and experimental conditions, in which the values of parameters, the initial conditions and simulation times are defined. Figure 5.17 is the window provided by the SimGest to support the specification and execution studies. With the window, a user can easily set the specification and execution studies and carry out the simulation experiments based on the defined simulation run conditions. Figure 5.18 demonstrates the algorithm of how the specification and

execution of simulation studies works.

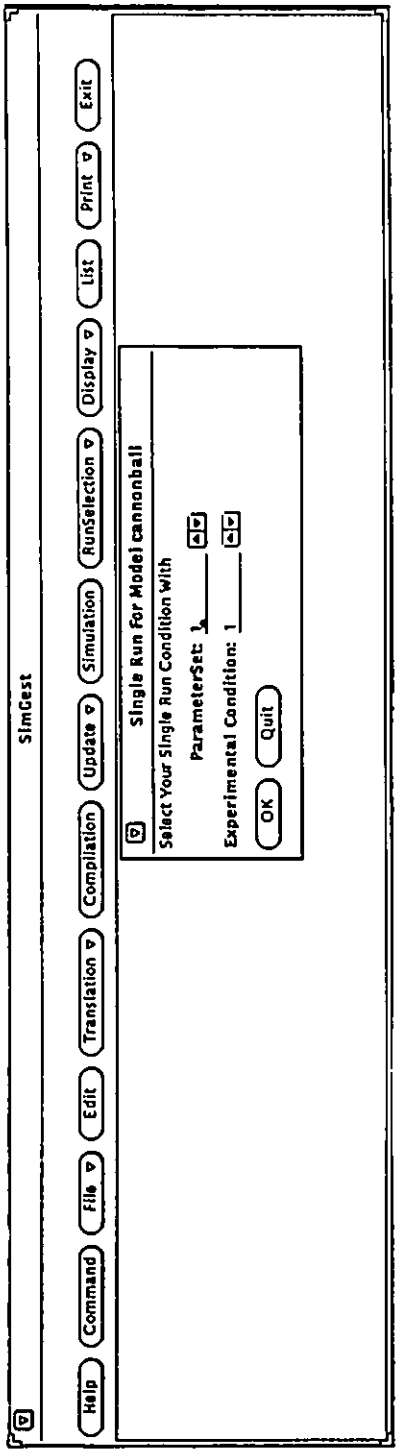

Figure 5.17 Interface Window for Specification and Execution of Simulation Studies

When a user choose the number of parameter set (line 9), the routine simulation_study gets the defined parameter set number by making the name of table for parameter set as a index (line 10). If any parameter is defined in the model (line 11), the routine puts the name of the table into the table for run condition (line 12). Then the routine obtains the experiment set number selected by a user (line 14) and make name of the table as index of the experiment set (line 15) and put it into the same table that holds the index of parameter set for run condition (line 16).

The executable simulation program obtains the indexes in the table for simulation run condition when the experiment simulation is executed. Having the indexes to the tables, the executable simulation program call the routine to access the tables individually to obtain the values of parameters, initial conditions and simulation times. Therefore, the executable simulation program gets all the simulation conditions for the execution of simulation study. Appendix 6 is the C code for the specification and execution of simulation studies in Gest.

```
1     routine simulation_study
2        declaration
3           file pointer
4           variable
5           buffers
6        end declaration

7        make name for simulation table for run condition
8        open the table to store the name
9        get the user selected number of parameter set
10       make name for simulation table for parameters
11       if parameter exists in the model
12         then put the name of the table into the table for run condition
13       end if
14       get the user selected number of experiment set
15       make name for simulation table for experiments
16       put the name into the table for run condition
17       close table
18    end routine
```

Figure 5.18 Algorithm of Specification and Execution of Simulation Studies

5.7 Displaying results

The study of the behavior of a model is to observe the results generated by simulation experiments. Each simulation experiment creates the simulation results. One can get different results with different simulation conditions. By displaying and observing the simulation results, one can learn the behavior of a model. The SimGest supports a user to observe the behavior of a model. by providing several ways of displaying the simulation results.

The simulation experiments are carried out by executable simulation program which accesses the simulation tables containing all the information about the simulation experimental conditions, values of parameters and specifications of the simulation study. After each execution of a simulation run, the simulation program generates the simulated behavior. The simulated behavior is reachable by the display routines in SimGest. If a user selects the display function, the display routines access the simulated behavior and display the results in the way that a user selects.

In this thesis, the SimGest provides a user a facility to display the results after each simulation experiment. There are two ways of displaying the results that can be selected by a user: the data format and the graphic format. The data format displays the results in a tabular format with the values of state variables versus time. The graphic format plots the results of state variables versus time. The data format gives a user relatively accurate results of the simulation experiments. The graphic format provides a user vivid view of the behavior of a model.

For the graphic display, a user can select different displays of simulation results by the display routines. The displaying results in the SimGest lets a user display a single run result or multi-run results. The single run graphic display plots the graph of the results simulated with only one run condition in a model. The multi-run graphic display shows a graph with different combinations of the run conditions defined in a model.

For single run display, a user can choose the display of all the state variables as well as the display of a single variable corresponding to the Run number. Figure 5.19 is the

window employed by the displaying results in the SimGest. A user can select run number for a single run display.

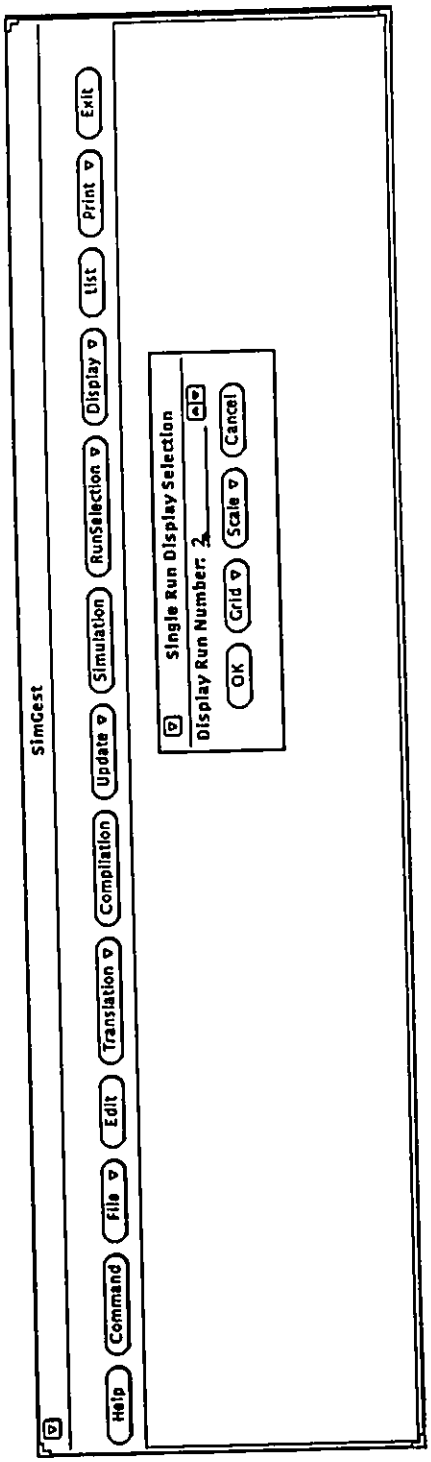

Figure 5.19 Interface Window for Selecting Run Number for Single Run

If the display for all state variable is selected, the display routines process the simulated behavior and displays all the state variables according to the `Display Run Number` chosen by a user through the window. Different display run number corresponds to different simulated behavior for the variables. Therefore, by selecting different `Display Run Number` for single display, the different simulated behavior can be displayed. Figure 5.20 shows the graphic display for all the state variables for the run number 2 for the model cannonball.

All the state variables displayed provide an overview of the behavior of a model. A user can easily observe the behavior of all variables at same time with that feature. Sometime a user may be interested in only one single variable of a model. The SimGest allows a user to display one variable graphic display if it is selected. A user can select any one of the state variables. Figure 5.21 shows the graphic display for the selected state variable y for Run 2.

Although the single state variable graphic display gives a clear view of a selected variable behavior, one may need to observe a certain area of the graphical display in order to learn the behavior of a model within that area. The displaying results provides a user a way to change the scale of the vertical axis or horizontal axis of the graph so that one can see the details in the graph that may not possibly be seen in a normal display.

For the graph in the Figure 5.21, one can enlarge it in vertical area by changing the vertical scale. The window in Figure 5.22 lets a user select the amplitude of the graphic display so that a user can look into some details of the graph.

Having the `Y-axis Start` and `End`, the simulation environment displays the vertically scaled graphic display in Figure 5.23 for the graph shown in Figure 5.21.

A vertically scaled display with a grid gives a user a clear view of the behavior of the variables in the specified area. In addition to changing the scale of the vertical amplitude, the displaying results also allows a user to change the scale of the horizontal scale of the graphic display so that a user can observe the selected variable behavior related to the simulation time factor. The window in Figure 5.24 lets a user change the horizontal scale.

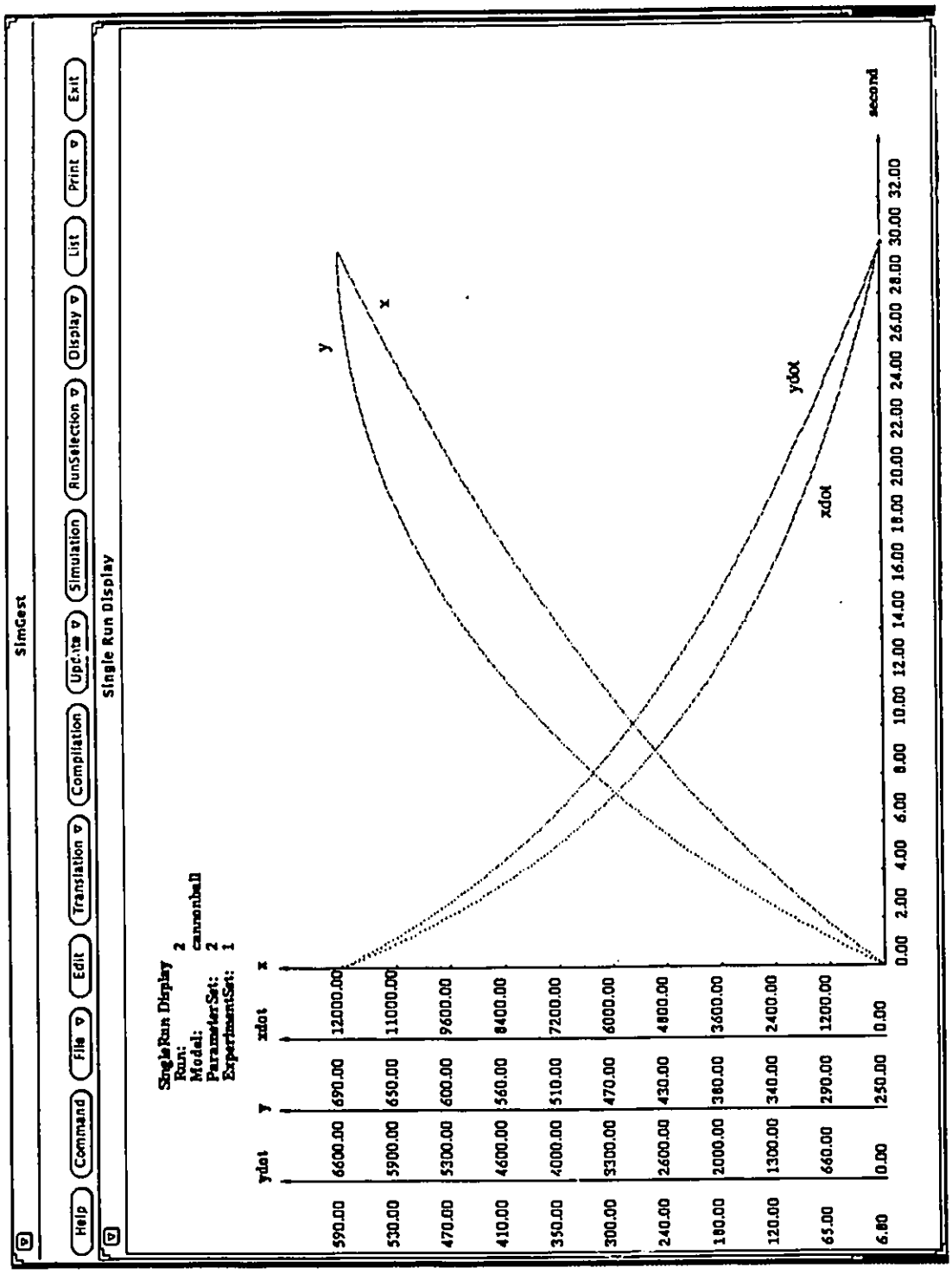

Figure 5.20 Graphic Display of All State Variables in the Model Cannonball

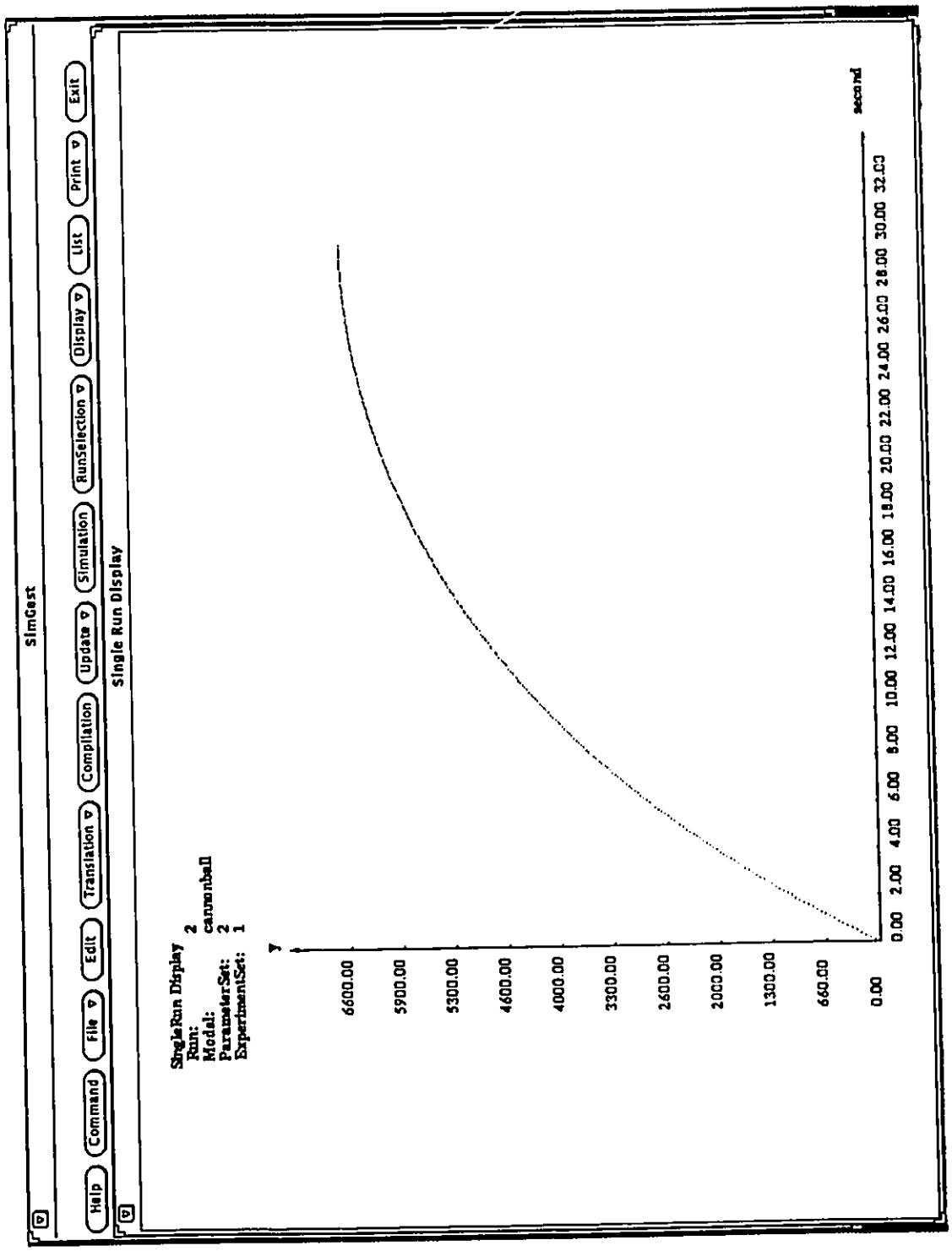

Figure 5.21 Graphic Display of a Selected Variable in the Model Cannonball

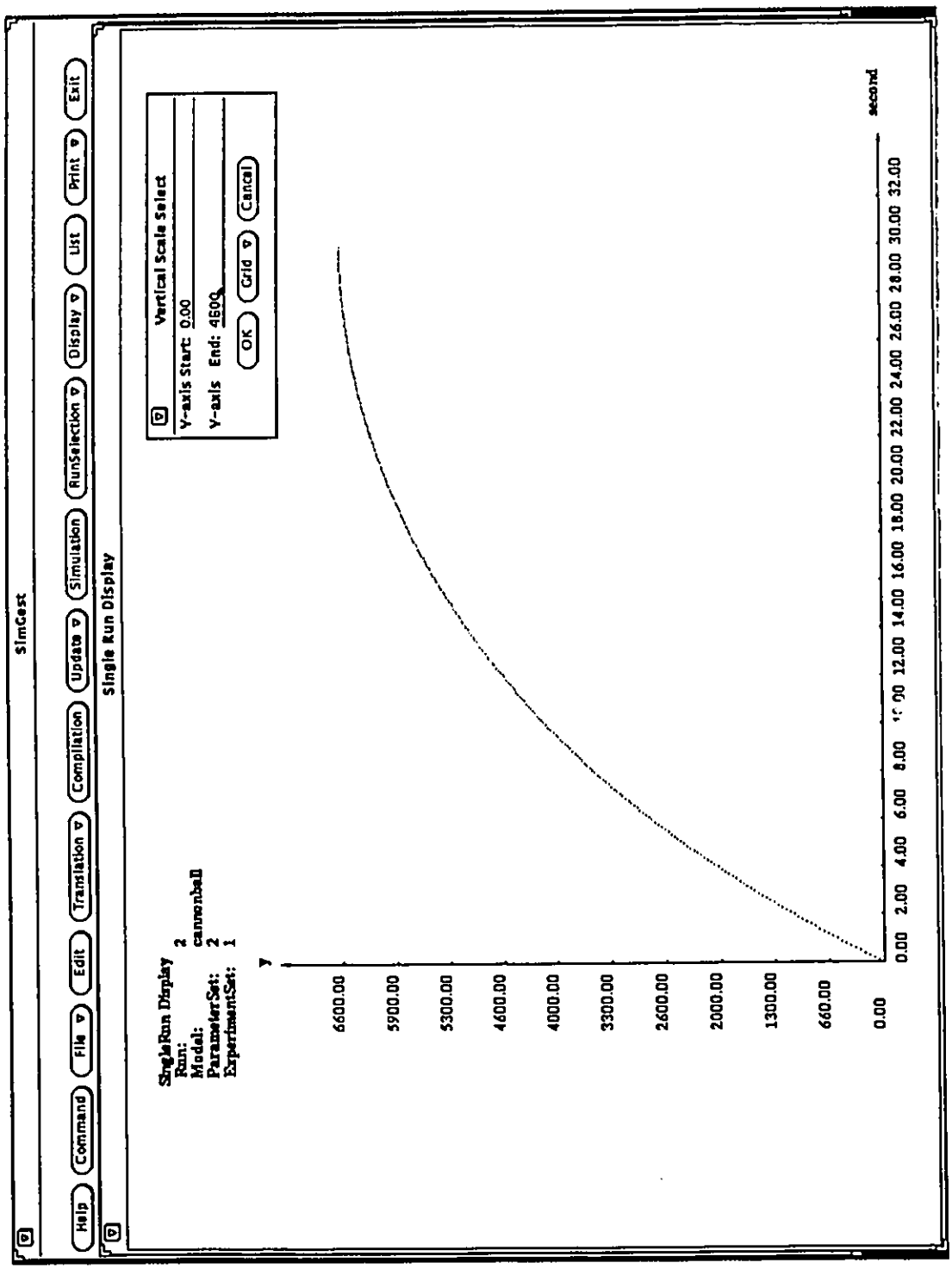

Figure 5.22 Interface Window for Selecting Vertical Scale for Graphic Display

If the Time Start and Time End are selected between the time interval for the graph, the horizontally expanded graph is displayed.

Figure 5.25 shows the horizontally expanded graphical display within the selected time interval [4.00, 20.00] for the graph in Figure 5.21

The single run display shows the behavior of a model with the selected run conditions. By selecting different display run numbers, one can observe different behavior of a model with different simulation run conditions. However, it is often desirable to display the results of different run conditions on the same graph so that one can observe the behavior of a model or compare the behavior with different run conditions. The window supported by the SimGest shown in Figure 5.26 gives a user access to selecting the multi-run graphic display of the simulated behavior

If the multi-run number is given by a user for the selected state variable (in the example the state variable is y and combinations are 2,1,3), the display routines process the simulated behavior and show the graphic display of the state variable results corresponding to the run number selected by a user. The selected combinational number, however, must be within the multi-run number that the model possesses, otherwise a warning signal is given. Figure 5.27 shows the multi-run graphic display of the selected state variable y with different run number for the model cannonball.

On the display, three trajectories of the state variable Y are shown with corresponding run conditions. One can learn the behavior of a model based on each run condition by comparing behaviors of different simulation run conditions on the same graph. Supported by the SimGest, the different results displaying functions provide a user a useful tool to observe and learn the behavior of a model after the execution of a simulation run.

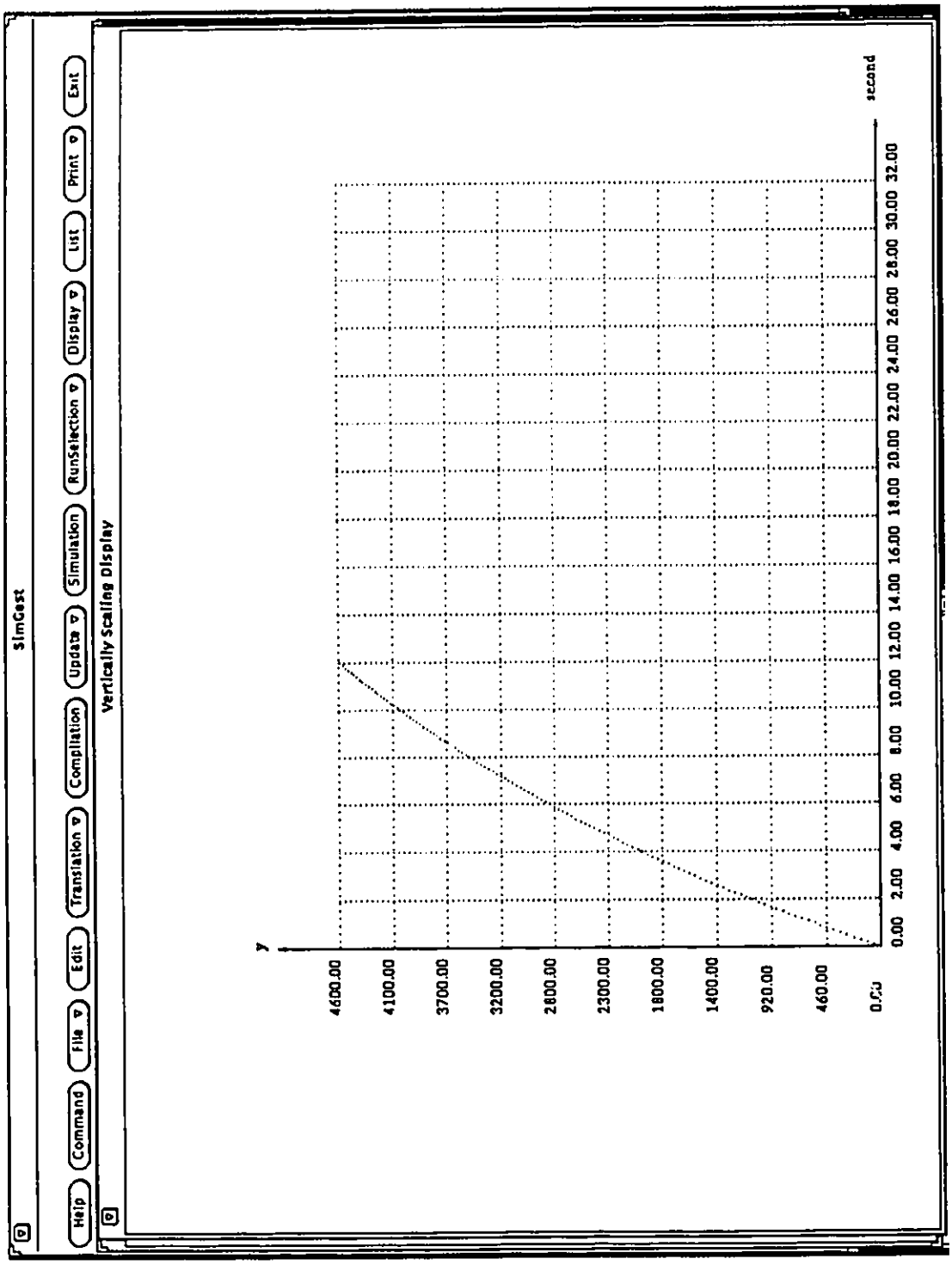

Figure 5.23 Vertically Scaled Graphic Display of a Selected Variable in the Model Cannonball

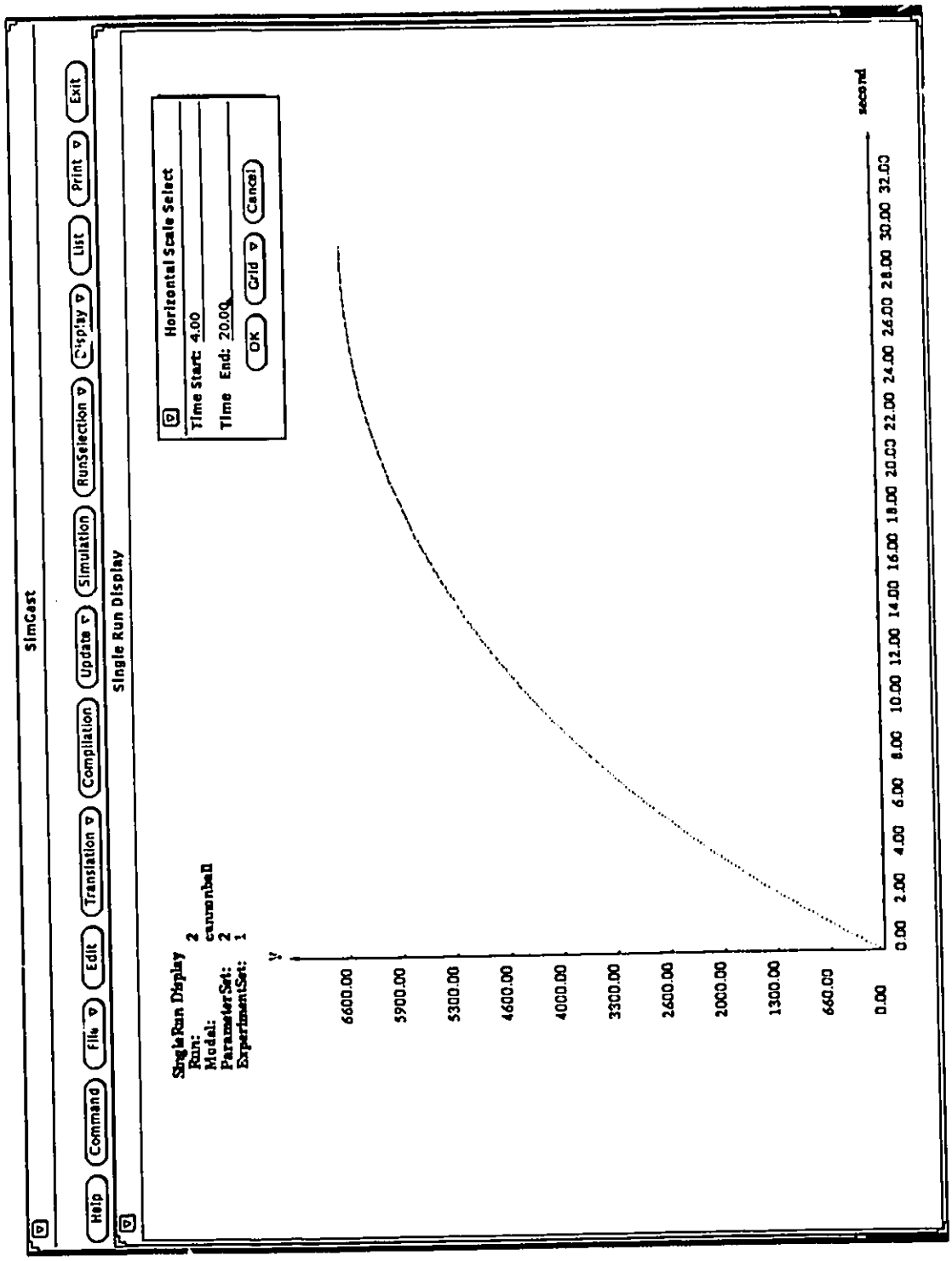

Figure 5.24 Interface Window for Selecting Horizontal Scale for Graphic Display

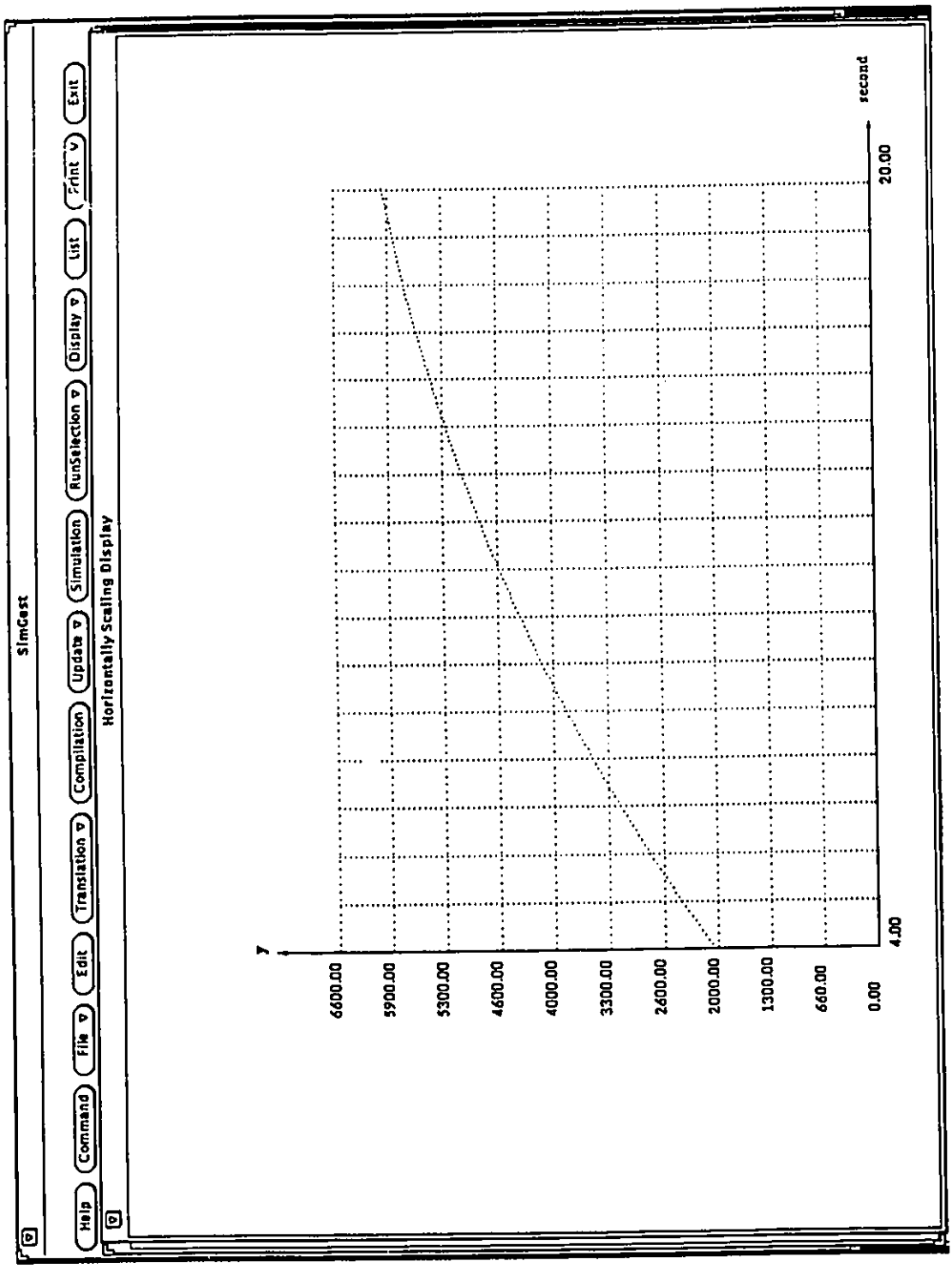

Figure 5.25 Horizontally Scaled Graphic Display of a Selected Variable in the Model Cannonball

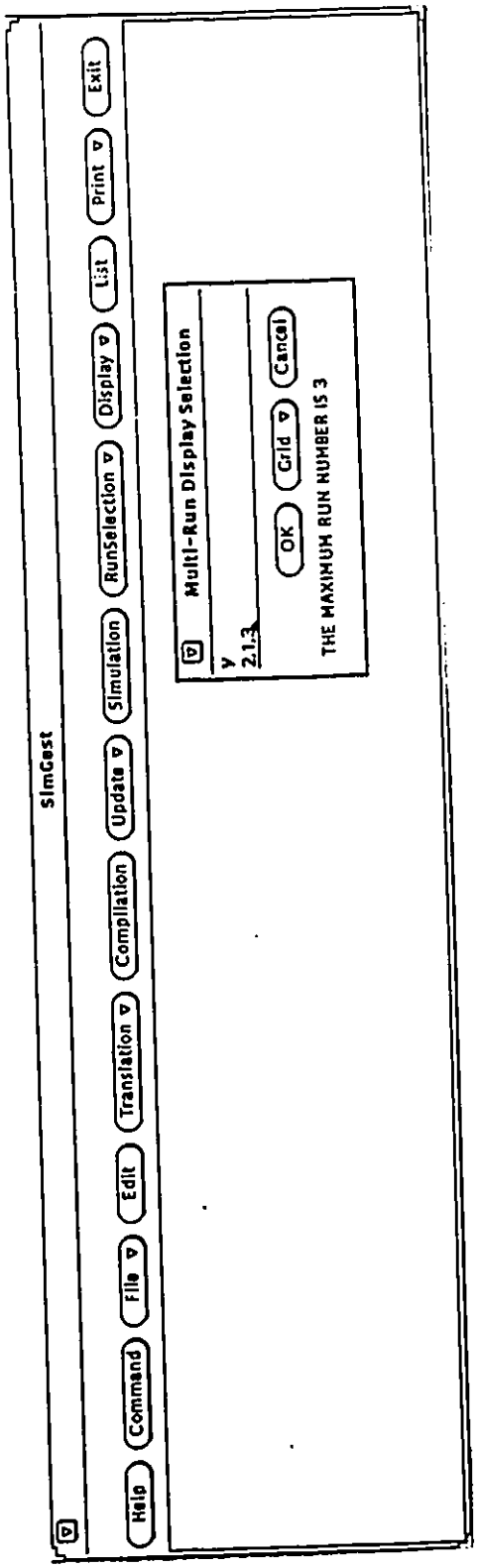

Figure 5.26 Interface Window for Selecting Run Number for Multi-run Display

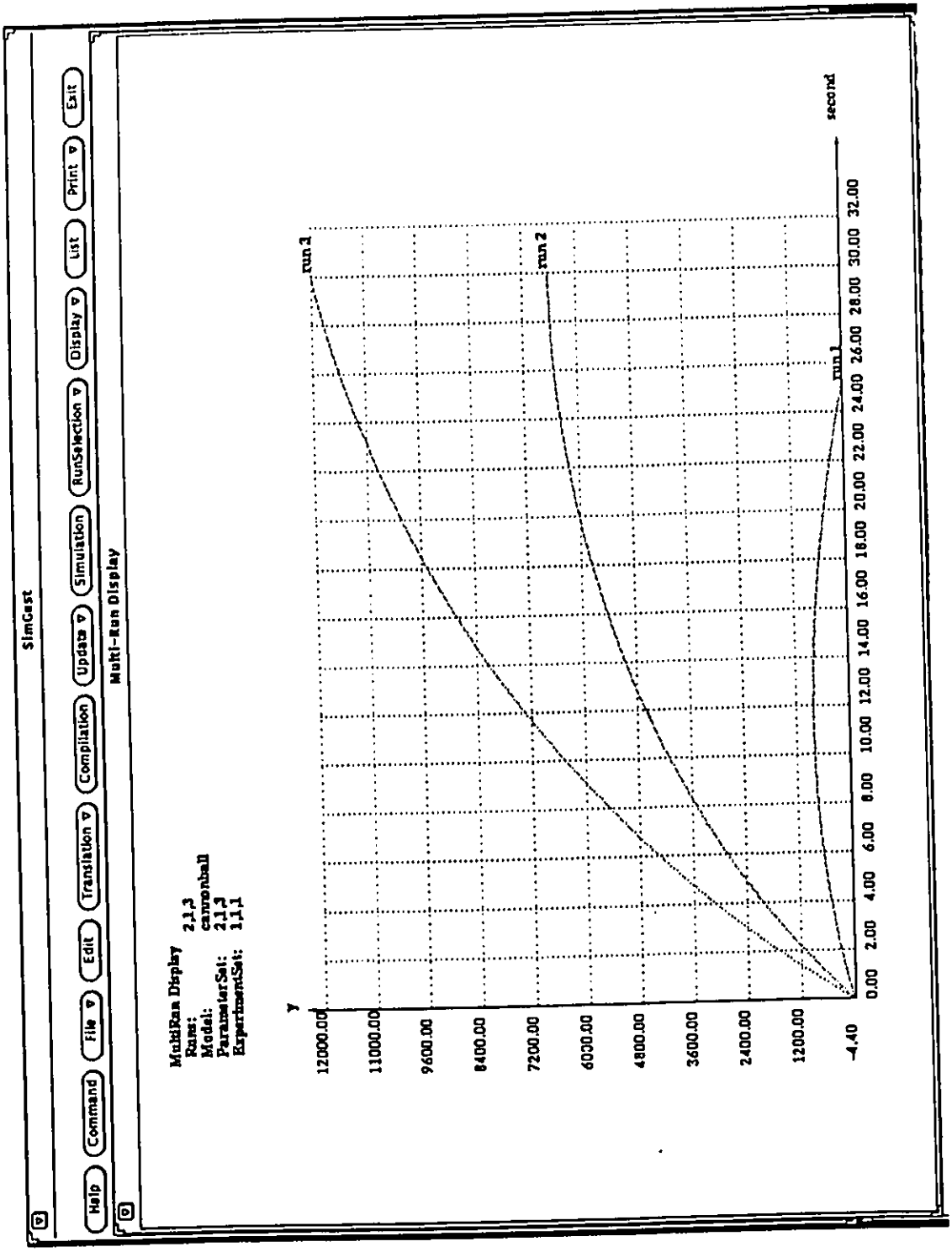

Figure 5.27 Graphic Display of a Selected Variable With Different Run Number in the Model Cannonball

6. Conclusions and future work

6.1 Achievements

In this thesis, a system called SimGest is developed and implemented on a SUN workstation. The SimGest consists of three parts: analyzer and generators, computer-aided specification module and display routines.

The analyzer and generators receive a model expressed in the Gest specification language and generate a simulation program in C. It is based on the program generator developed by Ye (1989). However, the program generator is modified in order to realize a table-driven simulation experimentation environment. Also, the analyzer and generators which consists of a Gest analyzer, program generator and generator of simulation tables are modified to be able to analyze any given continuous model and translate it into a C program. If the optional part which defines the values of the parameters and experimental conditions is specified in the input, the program generator provides this information for the simulation environment. However if the optional part is not defined, the program generator informs the simulation environment to prompts the user to define the necessary values of parameters, experimental conditions and simulation run conditions for the simulation experiment.

The computer-aided specification module for parameters, experimental conditions and the simulation study is supported by a graphic user interface that enables a user to study the behavior of a model efficiently. A user can either define or update the values of parameters, experimental conditions and simulation study interactively to carry out simulation experiment without time consuming re-generation and re-compilation of the C program. There are three interactive windows developed for a user to define or update the simulation conditions: one for parameters by which the values of parameters can be defined or updated; one for simulation experimental conditions by which the initial conditions and simulation time can be defined or updated; and third one for the simulation study to define or update the simulation run conditions .

The display routines get the simulated results and process the results in order to display

the behavior of the model under study. A user can display the results of the simulation run in several ways. One or more state variables can be displayed simultaneously. The information identifying the specific run conditions is also displayed on the same graph. In a multi-run simulation, composed of several single runs a selected state variable can be plotted on the same graph. The user can choose any run(s) referenced by run number for which to display a state variable. Vertically and horizontally scalable display of the simulation results provides additional useful features.

The SimGest system allows a user to monitor the whole process of simulation experiment, beginning from the selection of a model in the list to the stage of displaying and results of the simulation experiment. The simulation environment of the SimGest informs a user whether or not the generation and compilation are successful. The simulation environment also gives a message to a user telling the user that necessary information for carrying out the simulation run has to be defined if the parameters, experimental conditions or simulation run conditions have not been defined before the execution of simulation run.

In addition, the simulation experimentation environment provides some other functions which can be easily selected by a user by positioning the cursor on the menu option or on the button and clicking the mouse. The help information can be accessed at any time to give the user guidelines of how to use the SimGest system. A user can edit any file by using the implemented editor window in the simulation environment without exiting the system. The command window allows the user to issue on-line UNIX operating system commands to the host computer. The printing services let the user print any selected model in Gest and the generated simulation program in C as well as the simulation results in both graphic and tabular format.

6.2 Future work

The program generator developed in this thesis translates any continuous model in Gest into a C program. It is hoped that future development will make the program generator able to handle all types of the models in Gest, such as discrete models and coupled models etc., to carry out simulation experiments. In this thesis, the C programming

language was used to create the program generator. As a third generation programming language supported by many routines in the library, the C programming language is flexible and powerful. However, the object-oriented programming language C++ is more flexible and powerful. It is desirable, at least, to generate the C++ programs after the translation of the model in Gest for the execution of simulation experiment.

The function of interactively definable and modifiable single simulation run conditions supported by the simulation experimentation environment lets the user carry out simulation study by using different simulation run conditions. Multi-run simulation conditions are also an important part of a simulation study. It is desirable that the simulation experimentation environment allow the user to interactively define the multi-run simulation conditions if the multi-run conditions are not defined in the model. Therefore, the user can carry out a multi-run simulation experiment even after the translation of any model with the help of the simulation experimentation environment.

In a continuous model, ordinary differential equations are used to describe the behavior of the model. Different methods of performing numerical integration for the equations in the model affects the accuracy of the results and time required to determine the solution. To interactively define the integration method, similar to the way the user does for parameter(s) and experimental conditions will be another useful feature to add to the SimGest system.

Graphical display of the simulation results with the option of scaling lets the user easily observe the model behavior. Updating the SimGest system to run in a color environment would enhance the graphical display function. Different simulation runs and state variables could be displayed with different colors so that the user could easily distinguish individual curves on the same graph.

References:

Aytaç, Z.K. and Ören, T.I. (1986). *MAGEST: A Model-Based Advisor and Certifier for Gest Programs.* In: Modelling and Simulation Methodology in the Artificial Intelligence Era, M.S Elzas, T.I. Ören and B.P. Zeigler (eds.), North-Holland, Amsterdam, pp. 299-307.

Birta, L.G., Abou-Rabia, O. Ören, T.I., King, D.G. and Wendt, N.R. (1992). *Reverse Engineering in the Simulation Lifecycle.* Systems Analysis Modelling Simulation, 9:1, 69-84.

Birta, L.G. and So, M. (1990). *NEMS: A Database Support Environment for Numerical Experimentation.* Simulation, 45:4 (April), 189-200.

Davies, N.R. (1979). *Interactive Simulation Program Generation.* In: Methodology in Systems Modelling and Simulation, B.P. Zeigler, M.S. Elzas, G.J. Klir, T.I. Ören (eds.), North-Holland, pp. 179-200.

Denney, V.P. (1991). *Object-Oriented, Discrete-Event Simulation Using Smalltalk With Mixed-Paradigm Models.* In: Proceedings of the 1991 Summer Computer Simulation Conference, Hyatt Regency, Baltimore, Maryland, July 22-24, 1991, pp. 69-74.

do Régo, Y.D. (1992). *Outil logiciel pour le design et l'analyse d'expériences par les méthodes d'analyse factorielle et fractionnelle factorielle.* Master's Thesis, Ottawa-Carleton Institute of Computer Science, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada.

Funka-Lea, C.A., Kontogiorgos, T.D., Morris, R.J.T. and Rubin, L.D. (1991). *Experience with Enhanced Visual Modelling: Q+ with Expressions and Subnetworks.* In: Proceedings of the 1991 Summer Computer Simulation Conference, Hyatt Regency, Baltimore, Maryland, July 22-24, 1991, pp. 39-43.

Heller, D. (1991). *XView Programming Manual, Volume Seven,* O'Reilly & Associates, Inc., Sebastopol, CA, U.S.A.

Korn, G.A. (1989). *Interactive Dynamic Simulation,* McGraw-Hill, U.S.A.

Mathewson, S.C. (1974). *Simulation Program Generator.* Simulation, 23:6 (Dec.), 181-189.

Murthy, D.N.P. and Rodin, E.Y. (1987). *A Comparative Evaluation of Books on Mathematical Modelling.* Mathematical Modelling, 9:1, 17-28.

Ören, T.I. (1984). *GEST - A Modelling and Simulation Language Based on System Theoretic Concepts.* In: Simulation and Model-Based Methodologies: An Integrative View, T. I. Ören et al. (eds.), NATO ASI Series Vol. F10, Springer-Verlag, Berlin, Heidelberg, pp. 281-335.

Ören, T.I. (1987). *Simulation Methodogy: Top-Down Approach.* In: Systems and Control Encyclopedia, M.G. Singh (ed.), Pergamon Press, Oxford, England, pp. 4319-4323.

Ören, T.I. (1992). *Simulation Environments: Challenges for Advancement.* In: Proceedings of the 2nd Beijing International Conference on System Simulation and Scientific Computing, October 20-23, 1992, pp. 8-12.

Pace, D.K. (1991). *Technical and Management Factors That Can Enhance Simulation Development and Effective Use.* In: Proceedings of the 1991 Summer Computer Simulation Conference, Hyatt Regency, Baltimore, Maryland, July 22-24, 1991, pp. 3-8.

Payne, J.E. (1991). *Enhancing the Utility of Distributed Protocol Simulators.* In: Proceedings of the 1991 Summer Computer Simulation Conference, Hyatt Regency, Baltimore, Maryland, July 22-24, 1991, pp. 51-56.

Quercia, V. and O'Reilly, T. (1990). *X Window System User's Guide, Volume Three,* O'Reilly & Associates, Inc., Sebastopol, CA, U.S.A.

Warren, J.R. (1991). *A Prototype Case/Simulation System.* In: Proceedings of the 1991 Summer Computer Simulation Conference, Hyatt Regency, Baltimore, Maryland, July 22-24, 1991, pp. 295-300.

Wendt, N.R. (1993). *Application of Program Understanding and Rule-Based Quality Assurance to Slam II Simulation Programs.* Master's Thesis, Ottawa-Carleton Institute of Computer Science, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada.

Ye, Y. (1989). *A Method for Automatic Program Generation from a Specification.* In: Canadian Conference on Electrical and Computer Engineering, Montreal, Quebec, Canada, September 17-20, 1991, pp. 1078-1081.

Appendices

Appendex 1 Declaration and Main Part of the Generated C Program

```
#include <stdio.h>                          /* header files */
#include <ctype.h>                              /* system header file */
#include <string.h>                             /* system header file */
#include <math.h>                               /* system header file */
#include "LIB/method.c"                         /* system header file */
                                                /* generated header file */

#define m  4                                /* global declaration */
#define n  31                                   /* number of state variables */
#define S$  "cannonball"                        /* pointers of plotting the results */
#define T$  "second"                            /* name of model */
static float yy[n][m];                          /* unit of simulation time */
static float yi[m];                             /* variable for computation */
static float tt[n];                             /* variable for computation */
static float ti,tf,tc,h0;                       /* variable for computation */
static float r;                                 /* variables of simulation times */
static float v0;                                /* name of parameter */
static float theta;                             /* name of parameter */
static float g;                                 /* name of parameter */
static float ap1;                               /* name of constants */
static float ap2;                               /* name of auxiliary parameter */
static char *var_name[m] = {                    /* name of auxillary parameter */
                    "x",                        /* name of state variable */
                    "xdot",                     /* name of state variable */
                    "y",                        /* name of state variable */
                    "ydot",                     /* name of state variable */
                   };
FILE  *fplst;                                   /* pointer to file containing simulation
                                                   tables */
FILE  *fpplt;                                   /* pointer to file containing data for
                                                   results displaying */
static char   fnlst[40],fnplt[40];              /* buffers for the name of files containing
                                                   results for displaying */
static int nrun;                                /* variable of run number defined in Gest
                                                   specification */
static int rn;                                  /* new run processing number */
static char ascrn[5];                           /* buffer for ascii code of the run number */
```

```
main()                                          /* main of the generated program */
{                                                  /* declararion */
  FILE *sourp;                                        /* file pointer */
  g=32.2 ;                                            /* constants in the Gest model */
                                                   /* if file containing the number of run doesn't
                                                     exist */
  if ((sourp=fopen("numrun.indat","r"))==NULL)
    printf("Cannot open  numrun.indat\n");            /* print error message */
  fscanf(sourp,"%d",&nrun);                        /* get run number */
  fclose(sourp);                                   /* close the file */
  for (rn=1;rn<=nrun;rn++)                         /* loop until run index m reaches the run
                                                     number */
    {
     intetoa(&rn,ascrn);                              /* make name of file to store simulation
                                                        results of table format */
     strcpy(fnlst,"data");
     strcat(fnlst,ascrn);
     strcpy(fnplt,fnlst);                             /* make name of file to store simulation
                                                        results for plotting */
     strcat(fnlst,".lst");
     strcat(fnplt,".plt");
     if ((fplst=fopen(fnlst,"w"))==NULL)              /* if opening files is not successful */
       printf("Cannot open  %s\n",fnlst);               /* print error message */
     input(rn);                                       /* call routine input to access
                                                        simulation tables */
     rk4(tt,yi,yy);                                   /* call routine rk4 to do Runge Kutta
                                                        computation */
     fclose(fplst);                                   /* close the file */
    }
}                                               /* end main */
```

Appendix 2 Generated Routine Called by the Main to Access Simulation Tables

```
input(rn)
int rn;
{
  FILE  *ptbl;
  char  fnr[50];

  char  fnps[50];

  char  fnex[50];

  char  fnini[50];

  char   name[80];
  int    i;

  strcpy(name,"    RUN  ");
  strcat(name,ascrn);
  fprintf(fplst,"\n\n %s\n",name);
  strcpy(fnr,"run.indat");

  strcat(fnr,ascrn);
  if ((ptbl=fopen(fnr,"r"))==NULL)
    printf("Cannot open  %s\n",fnr);
  fscanf(ptbl,"%s %s %s",fnps,fnex,fnini);

  fclose(ptbl);
  if ((ptbl=fopen(fnps,"r"))==NULL)

    printf("Cannot open  %s\n",fnps);
  fscanf(ptbl,"%s %f",name,&r);
  fscanf(ptbl,"%s %f",name,&v0);

  fscanf(ptbl,"%s %f",name,&theta);
  ap1=v0*cos(theta);
  ap2=v0*sin(theta);

  fclose(ftbl);


  if ((ptbl=fopen(fnex,"r"))==NULL)

    printf("Cannot open  %s\n",fnex);
```

```
/* routine to access simulation tables */

  /* declaration */
      /* file pointer to simulation table */
      /* name of file name containing
        simulation run conditions */
      /* name of file containing parameters
        */
      /* name of file containing simulation
        times */
      /* name of file containing initial
        conditions */
      /* buffer for the title of display */
      /* integer variable */

  /* make title for display */

  /* put the title into buffer */
  /* make name of file containing simullation
    run conditions */

  /* if the file doesn't exist */
      /* print error message */
  /* get files pointers to parameters, initial
    conditions and simulation times */
  /* close the file */
  /* if the file containing parameters doesn't
    exist */
      /* print error message */
  /* get name and values of first parameter */
  /* get name and values of second
    parameter */
  /* get name and value of third parameter */
  /* obtain value of first auxiliary parameter */
  /* obtain value of second auxiliary
    parameter */
  /* close the file */


  /* if the file containing simulation times
    doesn't exist */
      /* print error message */
  /* get the simulation times */
```

```
 fscanf(ptbl,"%s %f %s %f %s %f %s %f",name,&ti,name,&tf,name,&tc,name,&h0);
 fclose(ptbl);                                  /* close the file */
 if ((ptbl=fopen(fnini,"r"))==NULL)             /* if the file containing the initial conditions
                                                   doesn't exist */
   printf("Cannot open  %s\n",fnini);               /* print error message */
 for (i=0;i<=(m-1);++i)                         /* get all intial conditions */
    fscanf(ptbl,"%s %f",name,&yi[i]);
yi[1] = ap1;                                    /* assign first auxiliary parameter to a state
                                                   variable as initial value */
yi[3] = ap2;                                    /* assign second auxiliary parameter to a
                                                   state vaiable as initial value */
 fclose(ptbl);                                  /* close the file */
}                                               /* end routine */
```

Appendix 3 Linked Library Routine for Runge Kutta Computation

```
rk4(tt,yi,yy)                          /* Runge Kutta computation routine */
  float  tt[n];
  float  yi[m];
  float  yy[n][m];

{                                      /* declaration */
  float  y1[m];
  float  y2[m];                        /* Runge Kutta computation workplace */
  float  y3[m];
  float  y4[m];
  float  f1[m];
  float  f2[m];
  float  f3[m];
  float  f4[m];
  float  t1,t2,t3,t4;
  float  zz[1000][m];
  float  tz[1000];
  float  ttc,td,ttd;
  float  h,rd,rh;
  int    jc,jd,kh,kd,nd;
  FILE   *rp;
  int    ir,jr;
  float maxy[m],miny[m];               /* boundary of calculating values */
  int   bhr[m], blr[m];

  for (ir=0;ir<m;ir++)    y1[ir]    = yi[ir];     /* initialization */
  for (ir=0;ir<m;ir++)    yy[0][ir] = y1[ir];
  for (ir=0;ir<m;ir++)    zz[0][ir] = y1[ir];
  t1 = ti ;
  tt[0] = t1;
  tz[0] = t1;

                                       /* define the step size based on
                                          communication time */
  rh = tc/h0;
  ir = 0;
  while (rh >= 10.0)
   {
    rh = rh/10.0;
    ir++;
   }
```

```
if  (rh >= 5.0)          kh = 5;
else if ( rh >= 2.0 )    kh = 2;
     else                kh = 1;
while (ir > 0)
 {
  kh = kh*10;
  ir--;
 }
rd = tc*500/(tf-ti);
ir = 0;
while (rd >= 10.0)
 {
  rd = rd/10.0;
  ir++;
 }
if ( rd >= 5.0 )         kd = 5;
else if ( rd >= 2.0 )    kd = 2;
     else                kd = 1;
while (ir > 0)
 {
  kd = kd*10;
  ir--;
 }
if (kd <= kh)
 { h = tc/kh;
   td= tc/kd;
 }
else
 h = td  = tc/kd;
ttc = tc;
ttd   = td  ;
jc = 1;
jd = 1;
```

```
while (t1 < tf )

  {
    deriv(&t1,y1,f1) ;

    for(ir=0;ir<m;ir++)
      y2[ir] = f1[ir]*h/2.0 + y1[ir];
    t2 = t1 + h/2.0 ;
    deriv(&t2,y2,f2);
    for(ir=0;ir<m;ir++)
```

```
/* Runge Kutta computation */
/* while loop until time reaches the terminal
   time */

    /* call routine deriv for the values of
       each state variable */
```

```
    y3[ir] = f2[ir]*h/2.0 + y1[ir];
  t3 = t2;
  deriv(&t3,y3,f3);
  for(ir=0;ir<m;ir++)
    y4[ir] =  f3[ir]*h + y1[ir];
  t4  = t3 + h/2.0 ;
  deriv(&t4,y4,f4);
  for(ir=0;ir<m;ir++)
    {
     y1[ir] = y1[ir] + h*(f1[ir] + 2.0*f2[ir] + 2.0*f3[ir] + f4[ir])/6.0;
     if (y1[ir] > maxy[ir])
        maxy[ir] = y1[ir];
     if (y1[ir] < miny[ir])
        miny[ir] = y1[ir];
    }
  t1 + = h;
  if (fabs(t1 - ttc) < 0.5*h)                    /* get data for table format display */
    {
     tt[jc] = t1 = ttc;
     for (ir=0;ir<m;ir++)
        yy[jc][ir] = y1[ir];
     ttc += tc;
     jc  +=  1;
     if (ttc > tf)
        ttc = tf;
    }


  if (fabs(t1 - ttd   ) < 0.5*h)                 /* get data for results plotting */
    {
     tz[jd] = t1 = ttd   ;
     for (ir=0;ir<m;ir++)
        zz[jd][ir] = y1[ir];
     ttd += td   ;
     jd += 1;
     if (ttd    > tf)
        ttd    = tf;
    }
  if (y1[2] <= 0.005) break;
}

for(ir=0;ir<m;ir++)
 {
  bhr[ir] = maxy[ir] + 1;
  blr[ir] = miny[ir] - 1;
 }
```

```
for(ir=0;ir<m;ir++)
  {
    if (bhr[ir]>bh[ir])
       bh[ir]=bhr[ir];
    if (blr[ir]<bl[ir])
       bl[ir]=blr[ir];
  }
nd = jd;
                                          /* create data files for table listing and curve
                                             plotting */
fprintf(fplst,"\n\n\t\tSimulation Results of %s\n\n\n",S$);
fprintf(fplst,"   Time");
for (ir = 0;ir < m;ir++)
   fprintf(fplst,"\t%8s",var_name[ir]);
fprintf(fplst,"\n\n");
for (jr = 0;jr < jc;jr++)                 /* load data to files */
  {
    fprintf(fplst,"%8.4f",tt[jr]);
    for (ir = 0;ir < m;ir++)
      fprintf(fplst,"\t%8.4f",yy[jr][ir]);
    fprintf(fplst,"\n");
  }


 rp = fopen(fnplt,"w");
 fprintf(rp,"%s\n%s\n",S$,T$);
 fprintf(rp,"%d\n%d\n%d\n%8.4f\n",m,nd,kd,td);
 for (ir = 0;ir < m;ir++)
    fprintf(rp,"%s\n",var_name[ir]);
 for (ir=0;ir<m;ir++)
    {
     getb(bhr,blr,upb[ir],downb[ir],m);
     fprintf(rp,"%s\n%s\n",downb[ir],upb[ir]);
     for (jr=0;jr<nd;jr++)
        fprintf(rp,"%8.4f\n",zz[jr][ir]);
    }
  fclose(rp);
}                                         /* end rk4 */


deriv(pt,xx,dx)                          /* routine for values of variables for each time */
 float *pt;
 float xx[m];
 float dx[m];.
 {
 float t;                                  /* declaration of variable time */
```

```
t = (*pt);                                          /* time value */
      dx[0]=xx[1] ;                                 /* value of first state variable */
                                                    /* value of second state variable */
      dx[1]=-r*sqrt(xx[1]*xx[1]+xx[3]*xx[3])*xx[1] ;
      dx[2]=xx[3] ;                                 /* value of third state variable */
                                                    /* value of fourth state variable */
      dx[3]=-r*sqrt(xx[1]*xx[1]+xx[3]*xx[3])*xx[3]-g ;
}                                                   /* end routine */
```

Appendix 4 C programs to Display Parameters as a Part of Computer-aided Specification Module for Parameters in SimGest

```
void display_parameter()                    /* display information about all the
                                               parameter */
{
  int  paranum = 0;                         /* variable for number of parameters */
  int  all_para = 1;                        /* variable for number of parameter
                                               set */
  int  set = 0;
  int  yes_par;                             /* flag for parameters */
  char window_title[80];                    /* buffer for title of window */

  strcpy(window_title,"Parameters for Model ");
  strcat(window_title,id_spec);
  if (pn == 0)                              /* if no parameters in the  model */
    para_pop();                             /* warning message */
  else{
    if (Pi != 0){                           /* parameter in parameter set is
                                               defined */
      while(all_para <= pn){
        fill_parameter(all_para);           /* obtain values of parameters in the
                                               simulation tables */
        all_para++;
        }
      yes_par = 1;                          /* set flag for parameter true */
      def_para(yes_par);                    /* create panels for parameters on the
                                               window */
      list_defined_par(paranum,set);        /* put values of parameters on the
                                               window */
     }
     else{
       yes_par = 0;                         /* set flag for parameter false */
       def_para(yes_par);                   /* create panels for parameters on the
                                               window */
       list_blank_par(paranum,set);         /* put blank for parameters */
     }
    xv_set(listpara_frame,FRAME_LEFT_FOOTER,id_spec,NULL);
    window_fit_height(para_list);           /* make height of the window */
    window_fit_height(listpara_frame);      /* make width of the window */
    xv_set(listpara_frame,FRAME_LABEL,window_title,XV_SHOW,TRUE,NULL);
   }
}
```

```
fill_parameter(set_no)                          /* obtain values of parameters in the
                                                   simulation tables */
int  set_no;                                       /* variable for number of parameter
                                                      set */
{
 FILE *ftp;                                        /* file pointer */
 int  i=0;
 char inter_file[80];                              /* buffers */
 char name[80], value[80];
 char ascii_form[80];

 inttoa(&set_no,ascii_form);                       /* get pointer to the parameter
                                                      simulation tables corresponding to
                                                      the number of parameter set */

 strcpy(inter_file,"par.indat");
 strcat(inter_file,ascii_form);
 ftp=fopen(inter_file,"r");
 for(i=0;i<pn;i++)                                 /* loop until index i equals the total
                                                      number of parameters */
   {
    fscanf(ftp,"%s %s",name,value);                /* get name and value of a parameter
                                                      */
                                                   /* put the name in the buffer */
    strcpy(parameter[i].para_set[set_no-1].p_name,name);
                                                   /* put the value in the buffer */
    strcpy(parameter[i].para_set[set_no-1].p_value,value);
   }                                               /* end loop */
 fclose(ftp);
}
```

```
def_para(defined)                               /* create the panel for parameters */
int   defined;                                     /* flag for parameter set */
{
 int  para_no = 0;                                 /* variable for number of parameter */
 Rect *rect;                                       /* window parameter */
  if (defined == 1)                                /* if flag for parameter set is true */
      xv_set(list_set_number,PANEL_VALUE,1,        /* set number of parameter set on the
                                                      window */
          PANEL_MIN_VALUE,1,
          PANEL_MAX_VALUE,Pi,NULL);

    if (text_create_flag == 0){                    /* if panel of the window not created */
       while (para_no < pn) {                      /* while loop */
```

```
                                                    /* create panel */
            gest_parameter[para_no].gest_para=(Panel_item)xv_create(para_list,
               PANEL_TEXT,
               PANEL_NEXT_ROW,-1,
               PANEL_VALUE_X,120,
               PANEL_VALUE_DISPLAY_LENGTH,20,
               NULL);
            rect = (Rect *) xv_get(gest_parameter[para_no].gest_para,XV_RECT);
            gest_parameter[para_no].para_note=(Panel_item)xv_create(para_list,
               PANEL_BUTTON,
               XV_X,rect_right(rect)+35,
               PANEL_LABEL_STRING,"Note",
               PANEL_NOTIFY_PROC,  para_comments,
               NULL);
            para_no++;
            }
      }
   text_create_flag = 1;                            /* set flag indicating panels created */
}




list_defined_par(par_no,par_set)                    /* display values of parameters */
int   par_no,par_set;                                   /* variables for number of parameter
                                                          and parameter set */
{
  while(par_no < pn)
     {
      xv_set(gest_parameter[par_no].gest_para,   /* parameters name and value*/
                                                 /* put name of parameter on the window */
          PANEL_LABEL_STRING,namep[par_no],
                                                 /* put value of parameter on the window */
          PANEL_VALUE,parameter[par_no].para_set[par_set].p_value,
          NULL);
      par_no++;
      }
}
```

```
list_blank_par(par_no)                          /* display blank for parameters */
int  par_no;                                        /* variable for number of parameter */
{
 while(par_no < pn)
    {
    xv_set(gest_parameter[par_no].gest_para,
        PANEL_LABEL_STRING,namep[par_no],       /* put name of parameter on the
                                                   window */
        PANEL_VALUE,NULL,                       /* put blank for value of parameter on
                                                   the window */
        NULL);
    par_no++;
    }
}
```

Appendix 5 C programs to Display Initial Conditions as a Part of Computer-aided Specification Module for Experimental Conditions in SimGest

```
void display_initial_conditions()
{
 if (ini_show == 0){                          /* if the selection of displaying
                                                 initial condition is true */
    vanish_con_aux();
    vanish_time();
    con_title_gone();
    var_title_create();
    var_create();                             /* call routine to create panels for
                                                 initial conditions */
     if (whole_model == 1){                   /* if the initial condition is defined
                                                 in the model */
        initilization();                      /* call routine to initialize */
        get_var_info();                       /* call routine to get initial
                                                 conditions */
       }
      else
        get_undefined_var();                  /* call routine to put blank for
                                                 initial conditions */
      rebuild_con_menu();
      rebuild_aux_menu();
      rebuild_time_menu();
      ini_show = 1;
      }
 else{
    var_gone();                               /* close the initial conditions
                                                 display on the window */
    var_title_gone();
    vanish_con_aux();
    vanish_time();
    con_title_gone();
    rebuild_con_menu();
    rebuild_aux_menu();
    rebuild_time_menu();
    ini_show = 0;
    }
  window_fit_height(var_list);                /* make height of the window */
  window_fit_height(state_frame);             /* make width of the window */
 }
```

```
var_create()                                        /* create panels for intial conditions */
{
 int  var_no = 0;                                       /* variable for number of state
                                                           variables */
 Rect  *rect;
 while (var_no < m) {                                   /* while loop */
                                                        /* create panel for variable on
                                                           window */
    gest_variable[var_no].gest_state=(Panel_item)xv_create(var_list,
        PANEL_TEXT,
        PANEL_NEXT_ROW,-1,
        PANEL_VALUE_X,160,
        PANEL_LABEL_STRING,name1[var_no],
        PANEL_VALUE_DISPLAY_LENGTH,20,
        NULL);
      rect = (Rect *) xv_get(gest_variable[var_no].gest_state,XV_RECT);
      gest_variable[var_no].var_note=(Panel_item)xv_create(var_list,
         PANEL_BUTTON,
         XV_X,rect_right(rect)+35,
         PANEL_LABEL_STRING,"Note",
         NULL);
      var_no++;
    }
 }


initilization()                                     /* initialize */
{
 FILE  *fp;                                             /* file pointer */
 char  infile[40];                                      /* buffer for file name */
 int   var_no=0;                                        /* variable for number of state
                                                           variable */
 if (initilize_done == 0){
    strcpy(infile,"ini.indat");                         /* get file name for simulation
                                                           table for initial conditions */
    obtain_file_name(infile);                           /* call routines to obtain initial
                                                           conditions */
    }
 initilize_done = 1;                                    /* set flag indicating initialization
                                                           is done */
}
```

```
get_var_info()                                          /* get initial conditions */
{
 int   varnum = 0;                                      /* variable for number of state
                                                           variable */
 int   set;                                             /* variable for number of
                                                           experiment set */
 set = xv_get(list_exp_set,PANEL_VALUE);                /* get number of experiment set */
 while(varnum < m){                                     /* while loop */
    xv_set(gest_variable[varnum].gest_state,
      PANEL_LABEL_STRING,name1[varnum],                 /* put name of state variable on
                                                           window */
                                                        /* put obtained initial conditions
                                                           on window */
      PANEL_VALUE,communicate[varnum].ini_value[set-1],
      NULL);
    varnum++;
    }
}


get_undefined_var()                                     /* put blank for initial condtions */
{
 int   varnum = 0;                                      /* variable for number of state
                                                           variable */
 while(varnum < m){                                     /* while loop */
    xv_set(gest_variable[varnum].gest_state,
      PANEL_LABEL_STRING,name1[varnum],                 /* put name of state variable on
                                                           window */
      PANEL_VALUE,NULL,                                 /* put blank for initial conditions */
      NULL);
    varnum++;
    }
}
```

Appendix 6 C Program for the Computer-aided Specification and Execution of Simulation Studies

```
void simulation_study()                    /* specification and execution of
                                              simulation studies*/
{
FILE  *fp;                                 /* file pointer */
int    no;                                 /* variable */
char   run_spec[80];                       /* buffers */
char   run_name[80];
char   asi[10];
strcpy(run_spec,"run.indat1");             /* make name for simulation
                                              table for run condition */
fp = fopen(run_spec,"w");                  /* open the table to store the
                                              name */
no = xv_get(sig_para_set,PANEL_VALUE);     /* get the user selected
                                              number of parameter set */
inttoa(&no,asi);                           /* make name for simulation
                                              table for parameters */
strcpy(run_name,"par.indat");
strcat(run_name,asi);
if (pn != 0)                               /* if parameter exists in the
                                              model */
  fprintf(fp,"%s\n",run_name);             /* put the name of the table
                                              into the table for run
                                              condition */
no = xv_get(sig_exp_set,PANEL_VALUE);      /* get the user selected
                                              number of experiment set */
inttoa(&no,asi);                           /* make name for simulation
                                              table for experiments */
strcpy(run_name,"glbl.indat");             /* simulation time file */
strcat(run_name,asi);
fprintf(fp,"%s\n",run_name);               /* put the name into the table
                                              for run condition */
strcpy(run_name,"ini.indat");
strcat(run_name,asi);
fprintf(fp,"%s",run_name);
fclose(fp);
}
```