Proceedings of the 1980 Winter Simulation Conference
T.I. Ören, C.M. Shub, P.F. Roth (eds.)

DESIGN OF SEMA: A SOFTWARE SYSTEM FOR COMPUTER-AIDED MODELLING AND SIMULATION OF SEQUENTIAL MACHINES

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

ABSTRACT: SEMA (SEquential MAchines) is a theory-based comprehensive computer-aided modelling and model processing system. The modelling methodologies of the current version of SEMA are finite-state machines (Moore and Mealy) and Markov chains. In this paper, modelling features, computer-aided modelling facilities, and the highlights of some other features of the SEMA system are discussed.

1. INTRODUCTION

The state-of-the-art of theory-based modelling languages is maturing (Elmqvist 1978; Elzas 1979) Klir 1978; Ören 1974, 1978, 1979, 1980c; Ören and den Dulk 1978; Ören and Zeigler 1979; Uytenhove 1978; and Zeigler 1979, 1980). Recently, Ören (1980b) proposed a comprehensive framework to discuss the possibilities for computer-aided modelling and model processing systems.

The modelling methodologies of the current version of SEMA are finite-state machines (Moore and Mealy) and Markov chains. There are simulation languages such as FISSL (Wilkens 1977) and YaRUS (Kuznetsov et al. 1972) based on finite-state machines. However, SEMA is being developed as a theory-based comprehensive computer-aided modelling and model processing system.

In this paper, modelling features, computer-aided modelling facilities, and the highlights of some other features of the SEMA system are discussed.

2. SEMA SOFTWARE SYSTEM

SEMA is a software system which consists of three major components: 1) a model-oriented specification language, 2) a user-oriented dialogue system, and 3) a program library. SEMA is not a procedural language. Therefore, a SEMA program does not specify how the computations should be carried out. Instead, by using the SEMA system one can specify: 1) models expressed as sequential machines, 2) algorithmic manipulation of such models, 3) computerized simulation of systems described by sequential machines, and 5) manipulation of computerized files of such models. As discussed later in the article, the user is also assisted by the system in these specifications.

In the design phase of SEMA, a computerized software documentation package, called SODPAC, is being used. SODPAC, designed by and implemented under the guidance of the senior author, already proved to be very helpful for documenting and detecting inconsistencies in language specifications.

A SEMA program consists of structurally well-defined three sections: 1) specification of the model, 2) specification of the algorithmic manipulation of the models, and specification of simulation studies.

The user completes each section before proceeding to the next. Once the model is complete, the user may specify which, if any, algorithms are to be performed on the model, and then finally specify the simulation experiment.

2.1 Specification of Model

In SEMA, a model may consists of one or more component models. If there are more than one component models, than their coupling specifies their input/output interface.

Currently, the modelling formalisms to specify component models are limited to two basic types of finite-state automata (Moore and Meally machines) and to Markov chain. However, some other types of sequential machine formalisms, such as cellular automata, will also be added to the system.

In the sequel basic elements of the SEMA language are given. The rules of the meta-language used to specify the syntax of the SEMA language are as follows:

1) The terminal symbols are enclosed within quotation marks.
2) The non-terminal symbols are written in lower-case characters.
3) The angular brackets [ ] denote option (zero or one time).
4) The curly brackets { } denote repetition (one or more times).
5) The parentheses ( ) denote syntactic groupings.
6) The equality sign is read "consists of."
7) The vertical bar is read "or" and denotes exclusive or.
8) Every rules is terminated by a period.

Moore machines

The syntax of the specification of a Moore machine in SEMA is as follows:

```
moore-model =
   "MODEL" model-identifier "IS" "MOORE"
   inputs
   states
   outputs
   state-transitions
   [ moore-output-function ]
   "END MODEL" model-identifier.
```

A Moore model may be conceived as having two sections: a static section and a dynamic section. The static section consists of the specification of the basic descriptive variables of a Moore machine, i.e., the inputs, states, and outputs.

The dynamic section consists of the state-transition and the output function.

In SEMA, a Moore machine may have one or more input "channels" and one or more output "channels." The Moore machine shown in Fig. 1a has one input channel and one output channel. However, the machine depicted in Fig. 1b has two input channels and three output channels. Every input (or output) channel represents an input (or output) variable. If there is only one input channel, it is not necessary to give a specific name to this input variable; the generic name "INPUT" may suffice. One may simply specify the input values. However, if there are more than one input channel then, every input variable has to be named separately and their values have to be specified individually. Similar considerations apply to output variables.

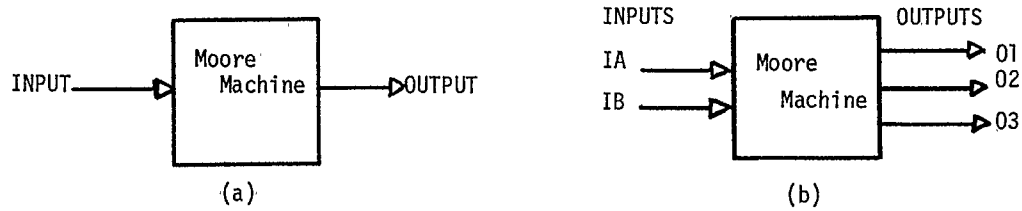

Figure 1. Moore Machines with Single or Multiple Input/Output Channels

In SEMA inputs are specified as follows:

```
inputs =
     "INPUT VALUES" input-value { "," input-value } ";"
   | "INPUT VARIABLE"["S"] input-variable [ { "," input-variable } ] ";"
     { input-variable ":"  input-value { "," input-value } ";"} .
```

Examples:

```
INPUT VALUES A,B,C,D;

INPUT VARIABLES IA,IB;
 IA : 0,1;
 IB : A,B,C;
```

To specify the inputs of the Moore machine given in Fig. 3a, one would normally use the specification starting with "INPUT VALUES," since there is only one input variable. However, if prefered one could name the input variable and then enumerate its values. In the case of Fig. 3b, since there are two input channels one has to use the definition starting with "INPUT VARIABLES."

The language definition for "outputs" is similar in form to "inputs" except that it adds the possibility of just having the state values outputed directly instead of a specific output value. In other words the output-value is the same as the state-value.

The state definition in SEMA is again similar to input definition. It is possible to specify the state variable(s) and enumerate its (their) values.

The state-transitions are specified in tabular format,with an entry for each (state, input) pair. The language definition for state transitions is as follows: (Note that the "-" represents a "don't care" transition of an incompletely specified machine).

```
states =
     "STATE VALUES" state-value { "," state-value } ";"
   | "STATE VARIABLE"["S"] state-variable [ { "," state-variable } ] ";"
     { state-variable ":" state-value { "," state-value } ";" } .
```

Example: (State transitions for a model with single state variable)

```
STATE TRANSITIONS
(* STATE          INPUT 0   1   2     *)
     Q:                 S,  T,  U   ;
     R:                 R,  S,  U   ;
     S:                 S,  U,  R   ;
     T:                 T,  R,  S   ;
     U:                 U,  T,  T   ;
END STATE TRANSITIONS
```

Example: (State transitions for a model with multiple state variables)

```
STATE TRANSITIONS
(* STATE            INPUT                  *)
(* K,L               0       1       2     *)
  (0,A):          (0,B)   (1,A)   (0,B);
  (0,B):          (1,A)   (0,B)   (1,B);
  (1,A):          (0,A)   (1,A)   (0,B);
  (1,B):          (1,B)     -     (1,A);
END STATE TRANSITIONS
```

The language definition of the moore-output-function is as follows:

```
moore-output-functions =
     "OUTPUT FUNCTION"
     { state-value ":" output-value [ { "," output-value } ] ";" }
     "END OUTPUT "

     "OUTPUT FUNCTION"
      { "(" state-value { "," state-value } ")" ":"
      output-value [ { "," output-value } ] ";" }
     "END OUTPUT" .
```

The second part of the rule is for those machines where the states are represented by n-tuples (i.e., where there are several state variables).

Example: (Output function of a Moore machine with single state variable and multiple output variables)

```
OUTPUT FUNCTION       (* OUTPUT VARIABLES *)
(* State                   OUTA    OUTB   *)
     Q:                     0,      0;
     R:                     1,      1;
     S:                     1,      2;
     T:                     0,      3;
     U:                     1,      3;
END OUTPUT
```

Example: (Output function of a Moore machine with multiple state variables)

```
OUTPUT FUNCTION
(* STATE          OUTPUT      *)
   (A,B):          0;
   (A,C):          1;
   (D,B):          0;
   (D,C):          2;
END OUTPUT
```

Meally machines

The specification of a Meally machine is similar to the specification of a Moore machine. The only difference is as follows: in a Meally machine, current output depends on the current state and input, while in a Moore machine, current output depends on the current state only. The definition of Meally output function is as follows:

```
meally-output-function =
      output-for-single-output-variable
    | output-for-multiple-output-variable .


output-for-single-output-variable =
     "OUTPUT FUNCTION"
     { state-value ":" output-value { "," output-value } ";" }
     "END OUTPUT"

   | "OUTPUT FUNCTION"
     { "(" state-value { "," state-value } ")" ":"
     output-value { "," output-value } ";"  }
     "END OUTPUT" .
```

Example:

```
OUTPUT FUNCTION
(* STATE        INPUT     0   1   2     *)
    Q:                    1,  0,  3;
    R:                    2,  2,  1;
    S:                    1,  3,  2;
    T:                    0,  2,  1;
    U:                    3,  1,  1;
END OUTPUT
```

```
output-for-multiple-output-variable
     "OUTPUT FUNCTION"
     { state-value ":" { "(" output-value { "," output-value } ")" } ";" }
     "END OUTPUT"


   | "OUTPUT FUNCTION"
      { "(" state-value { "," state-value } ")" ":"
      { "(" output-value { "," output-value } ")" } ";" }
     "END OUTPUT" .
```

Example:

```
OUTPUT FUNCTION
(* STATE        INPUT        0      1      2      *)
    A:                     (1,2)  (0,1)  (2,3)
    B:                     (0,3)  (2,2)  (1,3)
    C:                     (3,3)  (3,1)  (2,0)
END OUTPUT
```

Markov chains

Markov chains have been incorporated into the SEMA language to help model certain stochastic systems. The representation of the Markov chain is again in tabular form, with a probability value for each (state, state) pair. Also each state may have a output value represented by the moore-output function. The language definition is as follows:

```
markov-chain =
   "MODEL" model-identifier "IS" "MARKOV"
   states
   outputs
   markov-state-transitions
   [ moore-output-function ]
   "END MODEL" model-identifier .


markov-state-transitions =
   "STATE TRANSITIONS"
   { state-value ":" probability-value { "," probability-value } ";" }
   "END STATE TRANSITIONS" .
```

Example:

```
STATE TRANSITIONS
(* STATE          STATE    A   B   C   D     *)
    A:                    .2, .3, .4, .1;
    B:                    .0, .1, .8, .1;
    C:                    .1, .1, .7, .1;
    D:                    .2, .4, .4, .0;
END STATE TRANSITIONS
```

Coupling

The Coupling is the specification of the input/output interface of the component models. It is the final phase of model building, and can be thought of as, the tying together of the component models to form the resultant model.

The SEMA language definition of the coupling is as follows:

```
coupling =
   "COUPLING FOR" model-identifier
   "COMPONENT MODELS" list-of-coupled-models ";"
   {  model-identifier "." ( input-variable | "INPUT" ) "<--"
   [  model-identifier "." ( output-variable | "OUTPUT" ) ] ";" }
   "END COUPLING FOR" model-identifier .


list-of-coupled-models =
   coupled-model { "," coupled-model } .


coupled-model =
     model-identifier [ ":" unsigned-integer ".." unsigned-integer ] .
```

Note that the notation M:1..10, for example, causes the creation of ten copies of the component model M. The component models are identified as M1, M2, ..., M10.

Fig. 2 represents a model which results from the coupling of several component models.

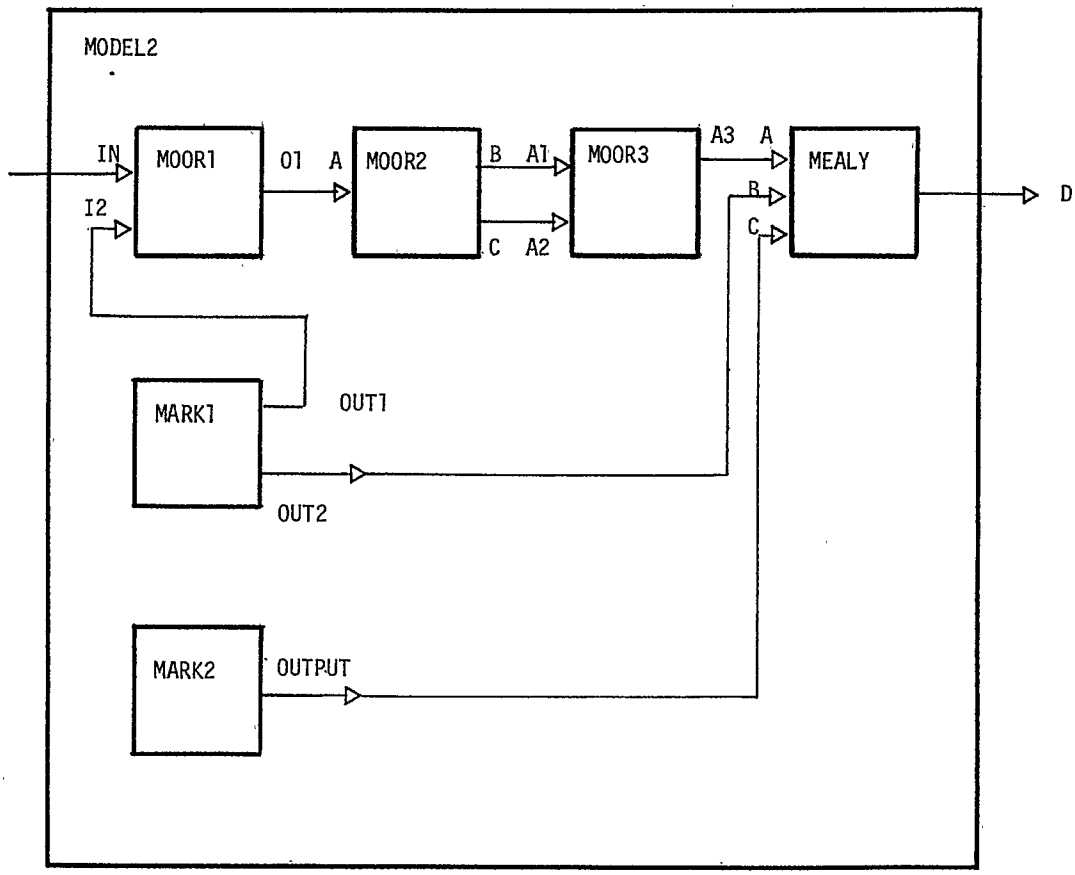

Figure 2. Coupling of several component models.

SEMA representation of the coupling depicted in Fig. 2 is as follows:

Example:

```
COUPLING FOR MODEL2
  COMPONENT MODELS MOOR1,MOOR2,MOOR3,
    MEALY,MARK1,MARK2;
  MOOR1.IN <-- ;
  MOOR1.I2 <-- MARK1.OUT1;
  MOOR2.A  <-- MOOR1.O1;
  MOOR3.A1 <-- MOOR2.B;
  MOOR3.A2 <-- MOOR2.C;
  MEALY.A  <--MOOR3.A3;
  MEALY.B  <--MARK1.OUT2;
  MEALY.C  <--MARK2.OUTPUT;
END COUPLING FOR MODEL2
```

A coupling specification starts with the declaration of the name of the resultant and the component models. Afterwards every component model is considered separately. For every input variable of every component model one has to identify the source of information within the coupled model. External inputs (such as IN of the component model MOOR1) can be identified by not specifying a source of information within the coupling.

The following is an example of the resultant model of coupled two models:

```
MODEL BINARY_COUNTER IS MOORE
   INPUT VALUES 0,1;
   STATE VALUES A,B,C;
   OUTPUT VARIABLES COUNT, CARRY;
      COUNT 0,1;
      CARRY 0,1;
   STATE TRANSITIONS
   (* STATE              INPUTS 0   1
        A:                       A , B;
        B:                       B , C;
        C:                       A , B;
   OUTPUT FUNCTION         (* OUTPUT VARIABLES *)
   (* STATE                   COUNT    CARRY   *)
        A:                      0     ,   0;
        B:                      1     ,   0;
        C:                      0     ,   1;
   END OUTPUT
END BINARY_COUNTER


COUPLING FOR COUNTER
  COMPONENT MODELS BINARY_COUNTER:1..2;
  (* NOTE: ":1..2" CREATES TWO COPIES OF MODEL BINARY_COUNTER *)
  (* AND NAMES THEM BINARY_COUNTER1 AND BINARY_COUNTER2 *)
   BINARY_COUNTER1.INPUT < -- ;    (* EXTERNAL INPUT *)
   BINARY_COUNTER2.INPUT < -- BINARY_COUNTER1.CARRY;
 END COUPLING COUNTER
```

2.2 Algorithmic Manipulation of Models

Given a mathematical model, one can use it for several purposes. In simulation, the obvious use is of course generation and observation of time-trajectories of the descriptive variables of a model. However, a large class of possibilities is algorithmic manipulation of models. Several well-defined algorithms exist since a long time (Gill 1962). Algorithmic model manipulation can be used for model transformation, comparison, decomposition, minimization, coupling, checking consistencies of model specification, and to perform several types of experimentations such as homing experimentation.

2.3 Specification of Simulation Experiments

Specification of the simulation experiments can be done in two parts: by specifying the experimental frame(s) and by specifying the (model, frame) pairs.

As described by Ören and Zeigler (1979), an experimental frame defines a limited set of circumstances under which a system is to be observed or subjected to experimentation. It consists of five blocks, i.e., observational variables, input schedules, initialization settings, termination conditions and specification for data collection and compression. An experimental frame may apply to both a real system or a mathematical model. Also there need not be just one experimental frame, clearly one may wish to observe a model or the real system under any number of different circumstances.

The syntactic definitions of "simulation," "experimental frame," and "specific frame" are as follows:

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


experimental-frame =
   "FRAME" identification "FOR" model-identifier
   { specific-frame }
   report-specification
   "END FRAME" identification .
```

In the experimental frame section the user will specify which coupled model the frame is for and then procede to the specific-frame specification.

```
specific-frame =
   "MODEL" model-identifier
   initialization
   [ input-scheduling ]
   { data-collection }
   "END MODEL" model-identifier .
```

For each component model, one uses a specific frame to specify the initial values of the state variables, the input schedules, and data collection requirements.

If more than one frame is specified for the frame specification, then the same number of initial states may be specified in the initialization specification. For example, if FRAME F:1..5 FOR MODEL 8 is specified for the frame specification, then five values of initial states would be specified for the initial state specification. This avoids needless repetition when only the initial state is changed in each frame for a model.

(Model, frame) pair specification allows a user to specify which model and experimental frame are to be used in a simulation study. The language specification is as follows:

```
model-frame-pair =
   "SIMULATE"
   model-identifier "/" "FRAME" identification [ { "," identification } ] ";" .
```

Examples:

```
SIMULATE MODEL2/FRAME 1;          (* ONLY ONE FRAME IS SPECIFIED *)
SIMULATE MODEL3/FRAME 1,2;        (* TWO FRAMES ARE SPECIFIED    *)
SIMULATE MODEL4/FRAME F:1..10;    (* TEN FRAMES ARE SPECIFIED    *)
```

In the last example the model MODEL4 is simulated first with experimental frame F1, second with frame F2 etc. up to and including frame F10.

3. COMPUTER-AIDED MODELLING AND MODEL PROCESSING

With the advent of interactive computer facilities through remote terminals, user-machine communications may reach a new height. The computer may be used not only to compile and execute a program, but to aid the user in constructing a model, in performing algorithmic consistency checks (Ören 1980), in algorithmic model processing as well as in documenting them, etc.

Computer-aided modelling and model processing systems may perform basically four groups of tasks: 1) modelling (model generation or fetching model(s) from a model file), 2) assisting the user to assess acceptability of model and acceptability of programs with respect to models, 3) algorithmic model processing, and 4) behaviour processing (Ören 1980 a).

3.1 Computer-Aided Modelling

The SEMA system guides the user in specifying different elements of the component models and their coupling. Once the user signs on to the system there is a constant flow of information between the system and the user via the terminal. The system will prompt the user for specific information, and the user will respond with the correct information. As an example, let us consider modelling a Moore component model (Fig. 3).

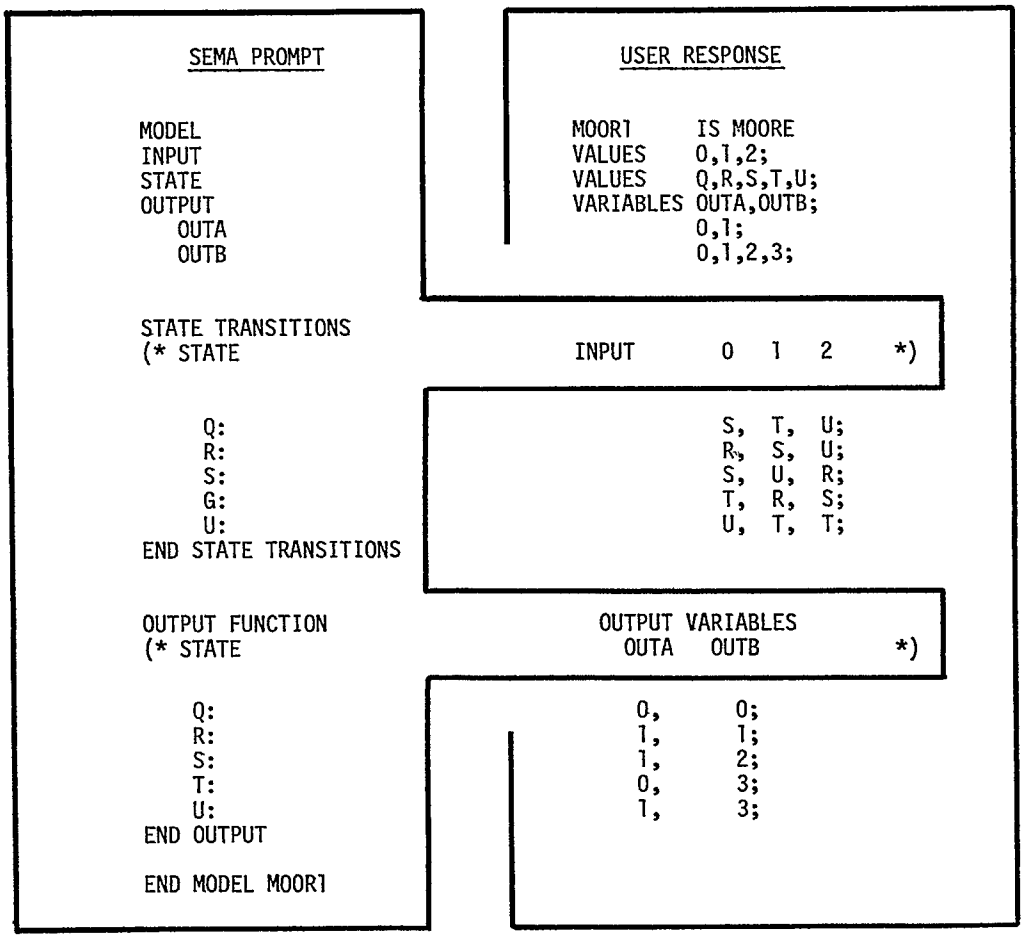

Fig. 3 Computer-aided modelling in the SEMA System

After signing on, the SEMA system will respond with "MODEL" on the terminal. The user will respond with a name for the model, say "MOOR1" then type "IS" "MOORE". From this point on the system will prompt only for information needed for a Moore machine. The next prompt will be for inputs and the system will respond with "INPUT" and the user will enter "VALUES" and specify the values or "VARIABLES" and list the names of the input variables; "IA, IB;" for example. Here the system, knowing the two input variables, and needing the values of the input variables, will respond by displaying the variable name (IA) and the user will supply the values. Then the system responds with another variable name (IB) and again the user will supply the values and so on

until all information for each variable has been specified. The system will respond in a similar manner for states and outputs.

The next response is for state transitions and the system prints out:

```
STATE TRANSITIONS
STATE        INPUT list-of-inputs
```

Thereafter, the system prints a state value, say Q followed by a column and expects the user to enter a list of state values followed by a semicolon . For each line of the state transition matrix, the SEMA system can automatically perform few consistency checks such as: 1) the number of entries should be identical to the number of input values, 2) the entries should be either a state value or the symbol "-". After getting the transition specification for the last element of the state values, the SEMA system can then automatically print

```
END STATE TRANSITIONS
```

and can initiate the dialogue for the specification of the output function. The SEMA system handles in a similar way the output-function as for the state-transitions.

3.2 Model Referencing

When using SEMA it is not allways necessary to create new component models. SEMA will have a file of component models which have already been defined by users. This file can be scanned for a model name. At this point the user may wish to use the documentation module of SEMA to obtain a listing of the model.

3.3 Model Acceptability

The SEMA system has built into it extensive consistency check facilities. Most of these algorithms are executed automatically without the user's knowledge, thus errors are treated systematically, bringing them to the user's attention while he is building a model. The user may then promptly take corrective action.

The algorithms will not only check for simple mistakes in syntax, but also complex input-output value matching for the coupling specification.

The computer-aided system guarantees correct model specification. Once a model is declared in a certain formalism, the system prompts for all the needed information for that formalism and will accept only that information. For example, if a model is declared to be a Moore machine, a Mealy output function can not be used. All the states, inputs, outputs state-transitions, etc. for that are requested to have a model description conformable to a specific modelling formalism.

3.4 Computer-Aided Documentation

SEMA provides two means of using its computer-aided documentation module, by using the LIST command or the DISPLAY command.

The LIST command provides selective information to the user and is used to find out what is available on the system. By typing "LIST ALL MODELS", the name and type of each model on the model file can be listed. Similarly for algorithms, if the user types LIST ALL ALGORITHMS the names of the algorithms on the system will be listed. If the user requires further information about models, couplings, or algorithms he must use the "DISPLAY" command for that specific reference.

The DISPLAY command may also be used for algorithms. In this case the specific algorithm is not listed, but its parameters and a brief summary of its purpose and additional information how to use it are given. The DISPLAY command may also be used displaying the result of simulation runs or the result of algorithmic manipulation performed on a model.

The output of both LIST and DISPLAY commands may be routed to any output-device by adding "ON" output-device to the end of the two commands. The output-device may be any one of terminal, printer, disk, or tape, where the terminal is the default.

4. CONCLUSIONS

The SEMA system is being developed at the University of Ottawa as a Master's thesis. Therefore taking into account the time frame of the thesis, only few modelling formalisms have been incorporated in the current version of the system. Similarly only a selected subset of the algorithms will be implemented for the first version of the system. However, additions of algorithms to the SEMA library will be a straightforward task.

REFERENCES

Elmqvist, H. (1978), A structured model language for large continuous systems, Ph.D. dissertation, Lund Institute of Technology, Lund, Sweden, 226 p.

Elzas, M.S. (1979), "What is needed for robust simulation?" in B.P. Zeigler et al. (eds.) Methodology in systems modelling and simulation, North-Holland, Amsterdam, pp. 57-91.

Gill, A. (1962), Introduction to the theory of finite state machines, McGraw Hill, NY, 1962

Klir, G.J. (1978), "Computer-aided system modelling," in Halfon, E. (ed.) Systems Theory in Ecology, Academic Press, New York.

Kuznetsov, O.P. et al. (1972), "YaRUS - A language for describing the operation of complex automata," Automation and Remote Control, Vol. 33, pp. 956-963, 1195-1203.

Ören, T.I. (1974), "Deductive general system theories and simulation of large-scale systems," Proceedings of Summer Computer Simulation Conference, Houston, Texas, pp. 13-16.

Ören, T.I. (1978), "Rationale for large-scale system simulation software based on cybernetics and general system theories," in T.I. Ören (ed.) Cybernetics and modelling and simulation of large-scale systems, International Association for Cybernetics, Namur, Belgium, pp. 151-179.

Ören, T.I. (1979), "Concepts for advanced computer-assisted modelling," in B.P. Zeigler et al (eds.) Methodology in systems modelling and simulation, North-Holland, Amsterdam, pp. 29-55.

Ören, T.I. (1980a), Computer-aided modelling systems, Technical Report No. TR 80.10, Computer Science Dept., Univ. of Ottawa, 7 p. Will be published in the Proceedings of SIMULATION '80.

Ören, T.I. (1980b), Concepts and criteria to assess acceptability of simulation studies: a frame of reference, Technical Report TR 80.03, Computer Science Dept., University of Ottawa, 51 p.

Ören, T.I. (1980c), "New directions in system simulation - methodology and software," Proceedings of the International Symposium on Systems Analysis and Simulation, Berlin, E. Germany, September 1 - 5, 1980 (In Press).

Ören, T.I. and den Dulk, J. (1978), Ecological models expressed in GEST 78, report prepared for the Department of Theoretical Production Ecology, Dutch Agricultural University, Wageningen, the Netherlands.

Ören, T.I., Zeigler, B.P. (1979), "Concepts for advanced simulation methodologies," Simulation, Vol. 32, No.3, pp. 69-82.

Uyttenhove, Hugo J.J. (1978), Computer-aided systems modelling: An assemblage of methodological tools for systems problem solving, Ph.D. dissertation, SUNY Binghampton, NY.

Wilkens, E.J. (1977), Finite State Techniques in Software Engineering, Proceedings of IEEE COMPSAC 77, pp. 691-697.

Zeigler, B.P. (1979), "Modelling and simulation methodology: state-of-the-art and promising directions," Keynote paper, IMACS Congress, Sorrento, Italy.

Zeigler, B.P. (1980), "Concepts and software for advanced simulation methodologies," in T.I. Ören, C.M. Shub, and P.F. Roth (eds), Simulation with discrete models: A state-of-the-art view.