Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Automatic programming assistant for network simulation models

Fan T. Tseng
Bernard J. Schroer
University of Alabama
in Huntsville
Huntsville, AL 35899

S. X. Zhang
Northwestern Polytechnical
University
Xian, Shaanxi
China

John W. Wolfsberger
NASA Marshall Space
Flight Center
Marshall Space Flight
Center, AL 35812

ABSTRACT

This paper presents the development of a simulation tool to assist the modeler of prelaunch countdown sequences define the problem and then automatically write the corresponding code in the target simulation language GPSS/PC. Included in this paper are a description of the Automatic Network Programming System (ANPS) and a sample problem using ANPS.

INTRODUCTION

There has been a considerable interest in improving the process of simulation model development. One area of interest is the development of simulation support environments. Henriksen (1983) suggests a simulation software development environment composed of a set of integrated software tools. Standridge (1983) proposes the integration of software tools and databases management techniques on each stage of the simulation model development process. Pidd (1984) also outlines a simulation support environment concept for handling one simulation problem at a time.

Overstreet and Nance (1985) emphasize the need of a specification language to assist in analysis of discrete event simulation models. Balci (1986) describe the requirements for general model development environments with focus on discrete event simulation modeling. Balci and Nance (1987) report a simulation support system for prototyping the automation-based paradigm. Rozenblit and Ziegler (1985) set up a conceptual framework for constructing knowledge-based, computer-aided environments for system analysis. Ziegler (1987) develops an object-oriented environment for hierarchical, modular discrete-event modeling.

A second area of interest is the automation of the simulation modeling process. The two main stages in the process are problem and code generation. Automatic Programming concepts are used in this area. Automatic Programming (AP) has been defined as the automation of some aspects of the computer programming process (Barr and Feigenbaum 1982). This automation is generally accomplished by developing another program, an automatic programming system, that raises the level of specifying computer program instructions. In other words, an AP system helps programmers write programs. Shooman (1983) have indicated that AP techniques consists of a dialogue between the designer and the computer that integrates existing software modules, or subroutines, into a main program to obtain reliable, moderately efficient program code.

An AP system should improve the overall environment for defining and writing programs (Brazier and Shannon 1987). Consequently, there should be a reduction in the amount of detail that the programmer needs to know. This improved environment should results in a more natural way for the user to define his problem and in a way that more closely resembles his way of thinking and looking at problems.

A number of attempts have been made at developing AP systems. One of the earliest was the Natural Language Programming for Queuing Simulations (NLPQ) (Heidorn 1974). Using an interactive dialogue, the NLPQ system created an internal description of the problem. From this internal description, the system generated the simulation code in the target language GPSS. The Electronics Manufacturing Simulation System (EMSS) (Ford and Schroer 1987) uses a natural language interface to define electronics assembly processes and then automatically writes SIMAN simulation code.

Brazier and Shannon (1987) have developed an AP system for modeling automated guided vehicle systems (AGVS). The system uses an interactive dialogue to define the AGVS. The system is written in Turbo Prolog for the IBM/PC. Once the AGVS has been defined, the system writes the corresponding SIMAN code. A Knowledge Based Model Construction (KBMC) has been developed to automate queuing model building and code generation (Murray and Sheppard 1988). Through an interactive dialogue the KBMC defines the problem specification and then automatically writes the SIMAN simulation code.

DRAFT is a program generator tool to assist the user to write simulation code (Mathewson 1984). The system also uses an interactive user dialog to solicit model parameters.

RESEARCH OBJECTIVE

In the 1960's, the Boeing Company (Synder et al. 1967) developed a simulation of the Saturn V prelaunch acctivities starting at T-24 hours and going through T-0. The simulation model consisted of over 1100 vehicle components and 400 ground support equipment components. A detailed time line was initially developed for these components. Next, the operations data, reliability data, and maintenance data were defined and input to the model. The principal model output was the probability of launching a vehicle within a given launch window. The model was written in GPSS-II and ran on the IBM 360.

The original Boeing model was expanded to include multiple launch windows and the operational sequence when a launch window was missed and the vehicle had to be recycled to the next launch window (Schroer 1969). This model consisted of two sequences: a main sequence identical to the original model, and a recycle sequence. The main sequence consisted of those events in the planned countdown between T-26 hours and T-0. The recycle sequence consisted of a number of backout sequences containing those events

required to return the countdown to some preceding point. The recycle sequence also consisted of a recycle hold containing those events required to sustain the vehicle status at a particular time in the countdown. The model was written in GPSS-II and contained 2300 blocks.

Figure 1 is a system overview of the major elements within a launch vehicle models. The knowledge base supporting the system contains the launch vehicle definition and operations, ground support equipment definition and operations, system maintenance parameters, and systems reliability parameters. The data from the knowledge base are used to define precedence relationships, dependency relationships, and activity times and to then construct the sequence of activities, or the time line. Figure 2 is a portion of a typical final time line. The system reliability and maintenance data are used to define the mean time to failure and mean time to repair for each activity and the corresponding types of distribution. The time line and reliability and maintenance data are then used in constructing the simulation model of the network. The experimental parameters are added and the model executed. The model results are used in conducting various trade studies of the prelaunch countdown operations.

After a review of the literature, it appears that the above class of network, or prelaunch countdown sequence, problems can be solved with these new automatic programming techniques. Furthermore, it appears that by using AP techniques, these problems can be solved more quickly and can be more easily verified and validated. The end result would be improved model clarity, increased productivity, rapid prototyping and easier system maintenance.

The goal of the research presented in this paper is to develop a simulation tool to assist the modelers of prelaunch countdown sequences define and develop the problem specification and then automatically generate the corresponding GPSS simulation code. This simulation tool is called Automated Network Programming Systems (ANPS).

AUTOMATED PROGRAMMING SYSTEM

The ANPS system is designed using the techniques of the automatic programming as its foundation. The ANPS system is built on the interactive dialogue approach to create a main program that includes the appropriate calls to the selected subroutines. The ANPS system consists of the following elements:

- Interactive user interface program
- Automatic code generator program
- Library of GPSS/PC subroutines or macros

Figure 3 is an overview of the ANPS system operation. Once the user has defined the problem domain, the user sits at a personal computer and responds to questions from the interactive interface program. Based on the responses, the interface program creates an internal problem specification file. This file includes the time line for the network, the attributes for the activities, and the dependent relationships between the activities. The problem specification is then used as input by ANPS to the automatic code

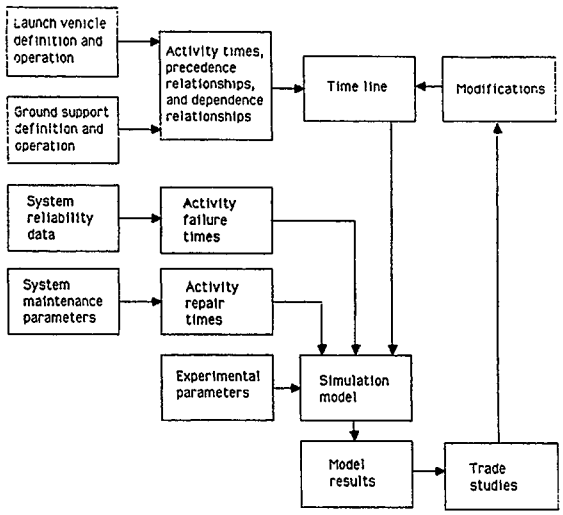

Figure 1. System overview of prelaunch network

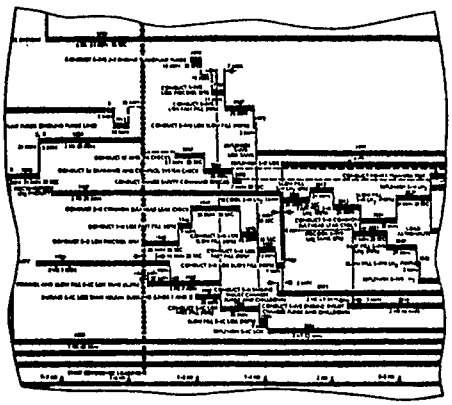

Figure 2. Typical time line

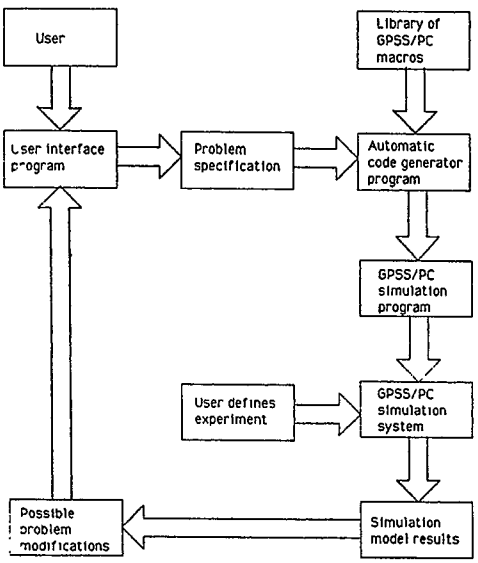

Figure 3. ANPS System overview

generator program. The code generator program selects and interfaces the appropriate macros from the GPSS library and then generates the simulation code for the main program in the target language GPSS/PC.

The output of the code generator program is a GPSS/PC program file. The experimental frame, such as the run statements, are then added and the GPSS program executed. To change the GPSS model, the user must recall the problem specification file. The interface program provides the user with a number of options to change or modify the problem specification, including a text editor to directly change the GPSS code.

The ANPS system is written in Turbo Prolog (Borland 1986) for the IBM class of personal computer. The library of macros is written in GPSS/PC. ANPS contains 1,218 lines of Prolog code and 86 subroutines. The simulation code generated by ANPS is GPSS/PC (Minuteman 1986).

Library of GPSS Macros. The power and robustness of an automatic programming system, such as ANPS, is its library of macros or subroutines. This library of macros is generally domain specific. When new macros have been defined, expert programmers are needed to write the code and to assure the proper interfaces.

The following functions are currently contained in the ANPS library of macros:

- Activity-event interation function
- Activity failure function
- Fixed activity operation function
- Continuous activity operation function

The activity-event interaction function ensures that all the activities are performed and events are achieved according to the precedence relationships. Each node, or event, will not occur until all the incoming activities have been performed. After all incoming activities have been completed, the event triggers all the outgoing activities.

The activity failure function simulates an activity failure. When an activity fails, the activity also causes its dependent activities to enter a hold state. The time to repair (TTR) the facility is generated to determine the failed activity's own interruption time. For the dependent activities that are active at the time of failure, TTR is their interruption, or hold time. On the other hand, if an activity has already been interrupted at that time, the interruption time is determined by the longer time of the TTR and the time caused by the previous interruption.

The fixed activity time operation function simulates the operation of each fixed activity. This function also generates the time to failure (TTF) of the activity. If the activity fails during its operation, the transaction is forwarded to the activity failure function.

The continuous activity time operation function simulates the operation of each continuous activity. The completion of a variable activity depends on other activities incident to the same ending node. The activity is not completed until those other activities are completed. If the activity fails the transaction is forwarded to the activity failure function.

System Constraints. The following constraints are imposed on the ANPS system:

- An activity failure will cause the activity to be delayed until the failure has been repaired.
- All dependent activities will also be delayed for the same time until the failure has been repaired.
- If another activity fails during the delay of a dependent activity and the dependent activity is also dependent on the just failed activity, the additional time to repair, if any, is added to the delay of the dependent activity.
- A dependent activity that has been delayed cannot fail during the delay time and wil not cause other dependent activities to be delayed.

EXPERIMENT

Figure 4 describes a time line for a typical countdown sequence problem consisting of six fixed activities and one continous activity. The dotted lines indicate time line constraints. For example, activities ACT5 and ACT3 must be completed before starting ACT6. This time line can be redrawn in the form of a network diagram as shown in Figure 5. Note that activities ACT2, ACT4, and ACT7 are on the critical path.

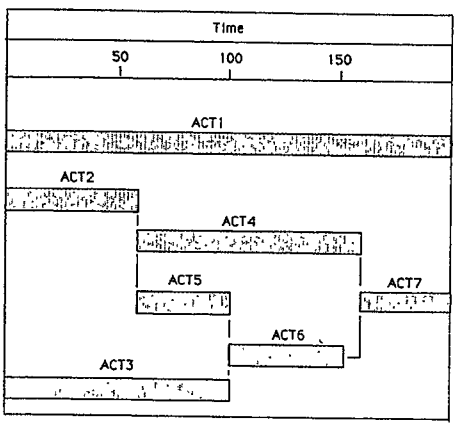

Figure 4. System time line

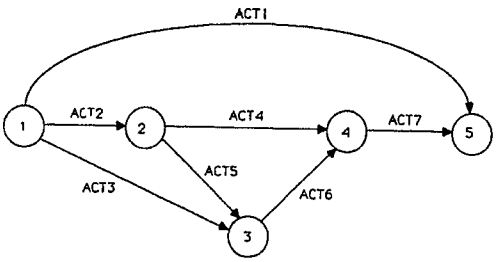

Figure 5. Sample reliability network

Table I gives the time parameters for the activities. These parameters include activity duration, time to failure, and time to repair. Note that activity ACT1 has a variable operation time. That is, this activity will operate during the entire duration of the system. An example of a variable activity is system power. System power may be required during the entire system operation and may be essential to a number of specific activities.

Table II gives the operational dependencies between the activities. In other words, the table shows the effect of an activity failure on other activities in the system. For example, a failure of activity ACT1 will cause a stopping of activities ACT2, ACT5, ACT6, and ACT7. Likewise, a failure of activity ACT3 will cause a stopping of activities ACT2 and ACT4.

Figure 6 contains a partial listing of the interactive user dialogue for defining the network. This dialogue is for defining activity ACT1, which starts at node 1 and ends at node 5. The activity type is variable, the time to failure follows an exponential distribution with mean of 60, and the time to repair follows the normal distribution with a mean of 20 and standard deviation of 2. Activity ACT1 has four dependent activities; ACT2, ACT5, ACT6, and ACT7.

Activity	Duration	Time to Failure	Time to Repair
ACT1	Variable	E(60)	N(20,2)
ACT2	60	E(30)	N(10,1)
ACT3	100	E(30)	N(10,1)
ACT4	100	E(20)	N(10,1)
ACT5	40	E(400)	N(20,2)
ACT6	50	E(100)	N(10,1)
ACT7	50	E(40)	N(10,1)

Table I. Activity Time Parameters

Figure 7 is a partial listing of the main GPSS program for the network in Figure 5 that was automatically generated by the ANPS system. Note that the network is constructed by a series of TRANSFER, SPLIT, and ASSEMBLE blocks. For example, line 2012 is the start of activity ACT1. Line 2013 splits one transaction to line 2019 and starts activity ACT2. Line 2020 splits another transaction to start activity ACT3.

The output from the simulation is the distribution of time to complete the prelaunch countdown sequence. In the example in Figure 4, the system time is 200. Therefore, if any failures occur, the system time would increase into the launch window, or beyond T-0. These increased time can then be used to compute the probability of launching within a given launch window.

CONCLUSIONS

The Automatic Network Programming System (ANPS), once fully operational, has the potential for use in rapidly modeling reliability networks. Specific applications for ANPS are in modeling prelaunch activities of space vehicles, ground support equipment space vehicle turn around plans, space transportation systems and operational planning, and hardware systems with multiple subsystems.

	Dependent Activities						
Activity	ACT1	ACT2	ACT3	ACT4	ACT5	ACT6	ACT7
ACT1	x	x			x	x	x
ACT2		x					
ACT3		x	x	x			
ACT4				x			
ACT5					x		
ACT6				x		x	
ACT7							x

Table II. Operational Dependencies Between Activities

```
   Name for GPSS Program            : EX1.GPS

1. Number of activities (max 50)    : 7
2.  Activity attributes:
    Activity name :$ACT1
    Activity type (fixed/variable) F/V : V
    Duration distribution type      : UNKNOWN
             mean time              : UNKNOWN
             standard deviation     : UNKNOWN
    Starting node number            : 1
    Ending   node number            : 5
    MTTF     distribution type      : EXP
             mean time              : 60

    MTTR     distribution type      : NOR
             mean time              : 20,
             standard deviation     : 2
    Number of dependent activities : 4

    Name of dependent activity 1: $ACT2
    .
    .
    Name of dependent activity 4: $ACT7
```

```
Type Description
A fixed activity is in operation for a defined time following a given distribution.

A variable activity is in operation for the duration of an activity or multiple activities.
```

```
Type
FIXED
VARIABLE
```

Figure 6. Partial listing of interactive user dialogue

```
1930 *
1935 *        MAIN NETWORK
1940 *
1945            GENERATE        .,,1
1950 MORE       SPLIT           1,MM
1955            GATE LS         SWITCH_MORE
1960            LOGIC R         SWITCH_MORE
1965            TRANSFER        ,MORE
1970 MM         MARK            SYSTIME
2000 EV1        ADVANCE
2001            TRANSFER        ,A1
2002 EV2        ADVANCE
2003            TRANSFER        ,A4
2004 EV3        ASSEMBLE        2
2005            TRANSFER        ,A6
2006 EV4        ASSEMBLE        2
2007            TRANSFER        ,A7
2008 EV5        ASSEMBLE        1
2009            LOGIC S         SWITCH_END1
2010 EEV5       ASSEMBLE        2
2011            TRANSFER        ,END1
2012 A1         ASSIGN          2,$ACT1
2013            SPLIT           1,A2
2014            ASSIGN          3,1
2016            LOGIC R         SWITCH_END1
2017            TRANSFER        SBR,VENT_B,RTRN2
2018            TRANSFER        ,EEV5
2019 A2         ASSIGN          2,$ACT2
2020            SPLIT           1,A3
2021            ASSIGN          3,2
2023            TRANSFER        SBR,VENT_A,RTRN2
2024            TRANSFER        ,EV2
2025 A3         ASSIGN          2,$ACT3
2026            ASSIGN          3,3
2028            TRANSFER        SBR,VENT_A,RTRN2
2029            TRANSFER        ,EV3
2031 A4         ASSIGN          2,$ACT4
2032            SPLIT           1,A5
2033            ASSIGN          3,4
2035            TRANSFER        SBR,VENT_A,RTRN2
2036            TRANSFER        ,EV4
2037 A5         ASSIGN          2,$ACT5
2038            ASSIGN          3,5
2040            TRANSFER        SBR,VENT_A,RTRN2
2041            TRANSFER        ,EV3
2043 A6         ASSIGN          2,$ACT6
2044            ASSIGN          3,6
2046            TRANSFER        SBR,VENT_A,RTRN2
2047            TRANSFER        ,EV4
2049 A7         ASSIGN          2,$ACT7
2050            ASSIGN          3,7
2052            TRANSFER        SBR,VENT_A,RTRN2
2053            TRANSFER        ,EV5
2055 END1       TABULATE        SYSTIME
2056 SYSTIME TABLE             MP$SYSTIME,0,50,50
2057            LOGIC S         SWITCH_MORE
2058            TERMINATE       1
```

Figure 7. GPSS listing of main program

There are a number of potential advantages of an automatic programming system such as ANPS. These advantages include:

- Improved Clarity - The GPSS code generated by ANPS is a structured simulation code that is easy to read, trace, and modify.
- Increased Productivity - The ANPS system should result in a significant increase in the lines of simulation code written per hour.
- Rapid Prototyping - Given the availability of the necessary macros, the system permits rapid prototyping. In addition, the system produces executable simulation code that is syntax error free.
- Easier Maintenance - The structured approach minimizes the effort required in locating errors and making program modifications.
- Reduced Modeler Knowledge - Hopefully the modeler's knowledge of the target simulation language should be reduced.

The ANPS system uses the interactive user dialogue to assist the modeler define the problem specification. The interactive user interface:

- Provides for a structured procedure for acquiring information on the system being modeled.
- Expedites the definition of the problem specification.
- Assures a complete and detailed definition of the problem specification.

In contrast to the advantages, there are also several disadvantages. One disadvantage, and probably the most significant, is that ANPS is domain specific. Another problem is that similar domains may require additional new macros or subroutines. An experienced GPSS simulationist must then be used to write the code for these macros. Another disadvantage is that the ANPS system requires more memory and execution time than a nonstructural equivalent program. However, this disadvantage is not as significant as in prior years because computers are becoming faster and have more memory. A fourth disadvantage is the user attitude problem of learning something new and different.

In summary, the ANPS system is still in the research stages of development. Hopefully, some day these types of automatic programming techniques will move from the research stages to actual implementation and operational use by the end user.

ACKNOWLEDGEMENTS

The research was funded in part by grant NAG8-641 from the NASA Marshall Space Flight Center and contract ADECA-UAH-9001 from the Science, Technology, and Energy Division of the Alabama Department of Economic and Community Affairs.

REFERENCES

Balci, O. (1985). "Requirements for Model Development Environments." Technical Report CS83022-R, Department of Computer Science, Virginia Tech., Blacksburg, Virginia, 495-502.

Balci, O., Nance, R.E. (1987). "Simulation Support: Prototyping the Automation-based Paradigm." In: Proceedings of the Winter Simulation Conference, Atlanta, Georgia.

Barr, A. and Feigenbaum, E.A. (1982). The Handbook of Artificial Intelligence, Vol. 2. W. Kaufman, Inc., California.

Brazier, M.K. and Shannon, R.E. (1987). "Automatic Programming of AGVS Simulation Models." In: 1987 Winter Simulation Conference, Atlanta, Georgia, 703-708.

Church, J. (1982). "Simulation Aspects of Flexible Manufacturing Systems Designs and Analysis." In: Proceedings of 1982 Fall Industrial Engineering Conference, Atlanta, Georgia, 426-431.

Ford, D.R. and Schroer, B.J. (1987). "An Expert Manufacturing Simulation System." Simulation 48, No. 5, 193-200.

Garrison, W.J. (1985). Network II.5 User's Manual. CACI, Los Angeles, California.

GPSS/PC Reference Manual. (1986). Minuteman Software, Stow, Massachusetts.

Haddock, J. and Davis, R.P. (1985). "Building a Simulation Generator for Manufacturing Cell Design and Control." In: Annual International Industrial Engineering Spring Conference Proceedings, Los Angeles, California, 237-244.

Heidorn, G.E. (1974). "English as a Very High Level Language for Simulation Programming." SIGPLAN Notices 9, No. 4, 91-100.

Henriksen, J.D. (1983). "The Integrated Simulation Environment (Simulation Software of the 1990s)." Operations Research 31, 1053-1073.

Khoshnevis, B. and Chen, A.P. (1986). "An Expert Simulation Model Builder." In: Intelligent Simulation Environment, Society for Computer Simulation 17, No. 1, San Diego, California, 129-132.

Mathewson, S.C. (1984). "The Application of Program Generator Software and Its Extensions to Discrete Event Simulation Modeling." IIE Transactions 16, No. 1, 3-18.

Mathewson, S.C. (1985). "Simulation Program Generators: Code and Animation on a PC." Journal of Operations Research Society 36, No. 7, 583-589.

Murray, K.J. and Sheppard, S.V. (1988). "Knowledge-based Simulation Model Specification." Simulation 50, No. 3, 112-119.

O'Keefe, R. (1986). "Simulation and Expert Systems-A Taxonomy and Some Examples." Simulation 46, No. 1, 10-16.

Oren, T.I. and Ziegler, B.P. (1979). "Concepts of Advanced Simulation Methodologies." Simulation 32, No. 3, 69-82.

Overstreet, C.M. and Nance, R.E. (1985). "A Specification Language to Assist in Analysis of Discrete Event Simulation Models." Communications ACM 28, 190-201.

Pidd, M. (1984). Computer Simulation in Management Science. Wiley, Chichester.

Reddy, Y.V., Fox, M., and Husain, N. (1985). "Automating the Analysis of Simulations in KBS." In: Proceedings of the 1985 Conference on Artificial Intelligence, Graphics, and Simulation, Society for Computer Simulation, San Diego, California, 34-40.

Rozenblit, J.W., Ziegler, B.P. (1985). "Concepts for Knowledge-based System Design and Environments." In: Proceedings of the 1985 Winter Simulation Conference, Atlanta, Georgia, 223-231.

Schroer, B.J. (1969). "Saturn V Prelaunch Systems Simulation Model for a Launch Opportunity Containing Multiple Launch Windows." In: Third Conference on Applications of Simulation, Los Angeles, California, 503-511.

Schroer, B.J. and Tseng, F.T. (1987). "Modeling Complex Manufacturing Systems Using Simulation." In: 1987 Winter Simulation Conference, Atlanta, Georgia, 677-682.

Shand, L.J., George, B., Davis, N., and Linderman, R. (1988). "Simulation Studies for the Design of a Space Surveillance Signal Processor." In: Proceedings 1988 Summer Computer Simulation Conference, Seattle, Washington, 723-728.

Shannon, R.E., Mayer, R. and Adelsberger, H.H. (1985). "Expert Systems and Simulation." Simulation 44, No. 6, 257-284.

Shooman, M.L. (1983). Software Engineering. McGraw Hill, New York.

Standridge, C.R. (1983). "Software Aids for Simulation." Simulation 41, 193.

Synder, J.E., Bennich, E.R. and Lindsey, Y.H. (1967). "Implementation of Advanced Simulation Techniques for Predicting the Saturn V Launch Vehicle System Behavior." Journal of Spacecraft and Rockets 4, No. 8, 998-1002.

Synder, J.E., Bennich, E.R. and Lindsey, Y.H. (1967). "Implementation of Advanced Simulation Techniques for Predicting the Saturn V Launch Vehicle System Behavior." In: AIAA 5th Aerospace Sciences Meeting, New York, Paper 67-205.

Turbo Prolog 2.0 Reference Guide. (1986). Borland International, Scotts Valley, California.

Ziegler, B.P. (1987). "Hierarchical, Modular Discrete-event Modeling in Our Object-oriented Environment." Simulation 49, No. 5, 219-230.