Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

AUTOMATIC GENERATION OF A CLASS OF SIMULATION MODELS FROM DATABASES

R. Bruce Taylor

Industrial and Manufacturing Engineering
Oregon State University
Corvallis, OR 97331

Hamdy A. Taha

Department of Industrial Engineering
University of Arkansas
Fayetteville, AR 72701

ABSTRACT

This paper shows the potential of using a higher level language, such as BASIC, to automatically generate syntactically correct simulation models from functional databases. The resulting models may then be executed directly by the general purpose simulation language of choice. The procedure is primarily developed for assembly line operations. However, it can be readily extended to network-based situations of the PERT-/CPM type and MRP types. The proposed approach is based on the observation that although the simulation logic of network-based models is generally simple, the structure of such models is highly input-data dependent. In the context of most general purpose simulation languages, this dependence usually necessitates making changes in the simulation model itself to accommodate the new structure of the network. Such "custom" changes impede both the ease of maintenance and the portability of the model. The outgrowth of pursuing the use of higher level language to generate executable simulation models encompasses many of the goals inherent in the simulation language work, including little or no programming and ease of maintenance and verification. The power of the process comes from having to model the problem generically only once by using a suitable higher level language model generator. Specific situations are then accounted for simply by changing the input data to the model generator.

1 INTRODUCTION

Simulation models are ultimately geared to meet the user's needs. In this regard, the user, who may not have formal training in simulation, should be able to make use of the model simply by preparing the input data in a user-friendly format. Unfortunately, most, if not all, general purpose simulation languages are designed first and foremost to respond to the modeling needs of the (trained) simulation analyst rather than the simulation user. As a result, the direct use of these languages usually requires a level of expertise that may be beyond the capabilities of the user of the model.

In response to this difficulty, a number of approaches have been proposed to bridge the gap between the simulation analyst and the simulation user:

1. Development of a user-friendly interface that can be used to input the model data and to examine the output results. In this regard, the simulation model itself becomes completely transparent to the user. An example of this type is given by Seppanen (1990) where an interface is used to input the data of an assembly-line system coded in a general purpose language. Although the use of the interface is definitely easier than dealing with the simulation model directly, the user must still learn some syntactical rules created by the interface, which in essence is equivalent to learning a specialized "language." Additionally, the size of the problem is limited by the original size of the simulation model.

2. Development of a specialized simulation language that represents the specific modeling needs of certain situations, such as manufacturing or communication network analyses. Some of these languages may be graphic- or menudriven to facilitate the process of constructing the model. Unfortunately, the level of modeling expertise required by these specialized languages still is beyond the capabilities or training of most end-users.

Of the two types presented above,

only the interface approach appears to be more suited to satisfying the needs of the end-user. However, the fact that the user may have to learn additional syntactical rules to implement the interface may be a disadvantage. This paper presents a third approach that calls for directly converting the functional data of the situation under consideration into a model that can be executed by a general purpose simulation language of choice. As such, the user will be dealing with databases with familiar formats rather than with special formats that satisfy the syntax of the general purpose simulation language itself.

The proposed procedure is most suited for modeling a class of problems that are highly input-data dependent but with repeatable logic. The basic premise of the approach presented in this paper is that once the precedence relationships at a node are given, we can write a model generator in a higher-level language (such as BASIC or C) to generate the code that models the node using the syntactical rules of the general-purpose simulation language of choice. The fact that the logic of each node is repeatable is ideal for developing a compact (recursive) model generator that develops the simulation model directly from the database describing the precedence relationships (among others) of the different nodes.

Figure 1 summarizes the approach by showing the relationship between the database, the model generator and the final simulation model.

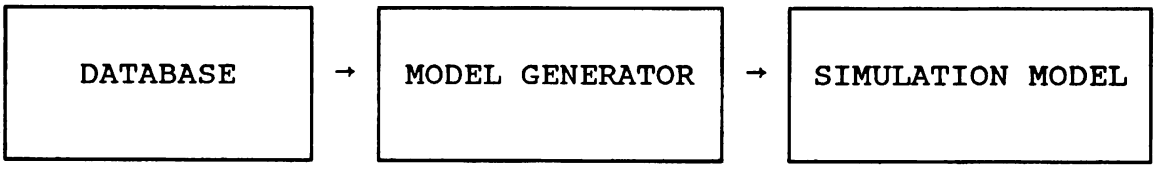

Figure 1: Generation of the Simulation Model from Database

2 A CLASS OF SIMULATION PROBLEMS

The analysis of assembly-line systems in manufacturing, MRP, and PERT/CPM type activities usually is based on modeling the problems as a network (nodes and branches) that specifies the precedence relationships governing the operation of the system. In simulation, the models representing such network-based situations are highly input-data dependent, in the sense that it is usually difficult to develop a general simulation model that accounts for all of these situations. In effect, the structure of the simulation model is usually changed to account for the specific node-branch relationship dictated by the original network. (See [Pritsker 1986, p. 216] and [Taha 1988, p. 343] for illustrations of PERT networks simulation.)

The logic of network simulation models is usually simple; namely, accounting for the input-output relationship at each node. It is also characterized by the repeatability of the logic at each node. In essence, the difference in the modeling of each node basically centers around making changes that accommodate the precedence relationships associated with each node.

The discussion above indicates that the class of problems that can be handled by the proposed approach must satisfy two conditions:

(a) Repeatability of the model components.

(b) The modeled system involves a set of consistent parameters that can be ascribed to each repeated component of the model.

These conditions apply quite well to network-based situations for which a varied range of applications exist as will be demonstrated below.

3 EXAMPLE APPLICATION

The example application describes an assembly manufacturing operation. The example is a simplified version of a more complete problem given in Malstrom (1981). It involves manufacturing a cap-cylinder assembly used for storing chemical samples from a number of components. Some of the components are purchased (P) and others are made (M) locally in the plant. The operation involves four major component assemblies as summarized in Figure 2 in the form of a part explosion diagram.

The objective of the simulation is to generate a cost estimate of the

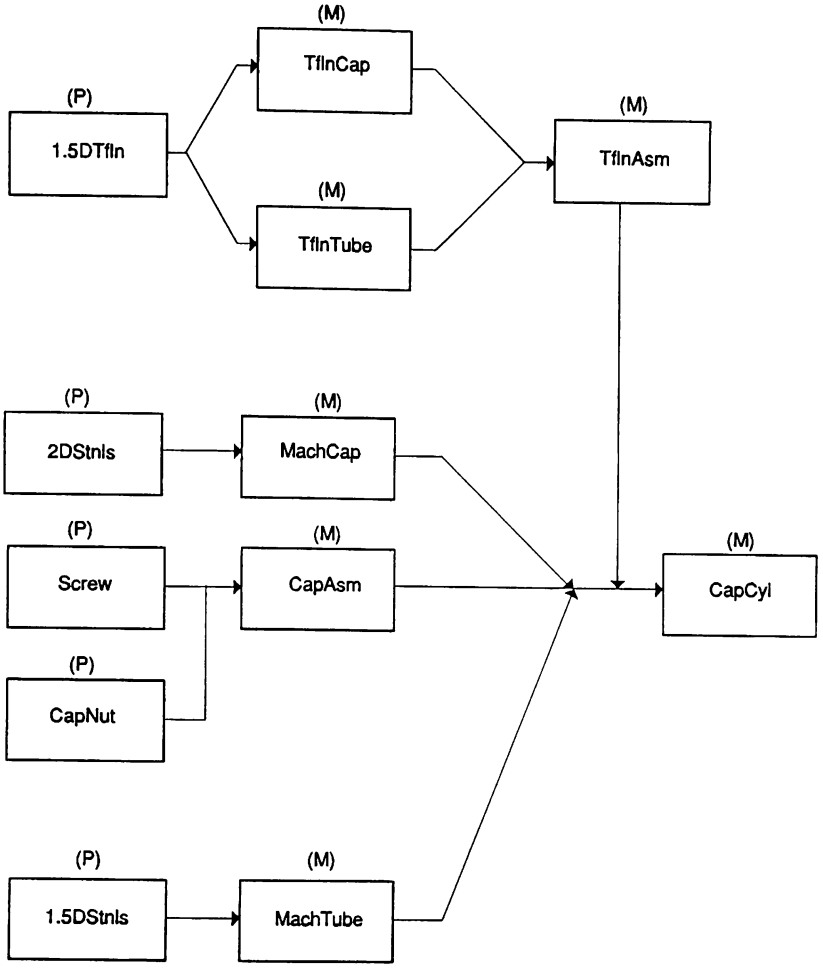

Figure 2: Part Explosion Diagram for the Example Application

final assembly CapCyl. The database for the situation is shown in Table 1. The organization of the data as shown is typical of the manner in which the user will prepare the input for the model generator. In essence, the data are in raw form. Notice that some of the times are expressed as samples from probability distributions. Such data may also be replaced by deterministic values if desired, or by any mathematical expression acceptable to the language of choice.

It is equally plausible to change the objective of the simulation to provide other results. For example, we may wish to estimate the throughput time for the final assembly; or we may wish to develop the model to account for possible bottlenecks in the flow of materials. Any of these objectives can be accounted for simply by providing the proper input and by changing the basic structure of the model generator.

4 MODEL GENERATOR

The model generator is usually written by the simulation analyst (rather than by the user) using a higher level language. The generator uses the input data of Table 1 to generate the simulation model for a specific general purpose simulation language chosen by the analyst. In our presentation, we choose the SIMNET II simulation language for this development because it allows the indexing of all the model elements (nodes, resources, statistical variables, and switches), a quality that is particularly suited for use with an iterative generic model generator. The basic idea for coding the model generator (which is written in BASIC) is to create the purchase (P) parts by using a SIMNET II source node. The created transactions will then feed into their next made (M) part node. As the transactions traverse the branches, they compute the desired costs.

Table 1: Database for the Example Application

Description	P/M	Unit $	Setup Time	Time/ Piece	$/hr	No. Comp.	Preceding Components	Quant/ Assembly
"CapCyl"	88							
"1.5DTfln"	"p"	19.78						
"TflnCap"	"M"		.5	"un(.11,.31)"	18	1	"1.5DTfln"	.083
"TflnTube"	"M"		.75	"un(.33,.53)"	18	1	"1.5DTfln"	.5
"TflnAsm"	"M"		0	" 0 "	0	2	"TflnCap"	1
							"TflnTube"	1
"2DStnls"	"p"	26.16						
"MachCap"	"M"		2.75	"un(.63,.83)"	18	1	"2DStnls"	.167
"Screw"	"p"	.09						
"CapNut"	"p"	.09						
"CapAsm"	"M"		0	" 0 "	0	3	"MachCap"	1
							"Screw"	1
							"CapNut"	1
"1.5DStnls"	"p"	14.72						
"MachTube"	"M"		2.75	"un(.74,.94)"	18	1	"1.5DStnls"	.5
"CapCyl"	"M"		.75	"un(.03,.05)"	16	3	"TflnAsm"	1
							"CapAsm"	1
							"MachTube"	1

Notice the repetitive property of the model alluded to earlier. Each node represents either a purchased (P) or a made (M) part. Both purchased and made parts have their own specific set of data which are consistent among all the parts. Given the precedence relationships for each node, the simulation analyst can then write the model generator generically to create the SIMNET II model. As an illustration, Figure 3 provides the portion of the BASIC program that is used to generate the main logic of the simulation. The other segments of the generator deal with checking the input data and with generating the control statements of the simulation run.

Once the generator has been verified to produce the correct simulation code, the user can invoke the generator to model any similar problem simply by changing the input data as given in Table 1. The resulting model is guaranteed to be free of logical and syntactical errors regardless of the structure or complexity of the model representing the problem. In this regard, the simulation language itself is totally transparent to the end-user.

Appendix A provides a listing of the BASIC generator used to develop the SIMNET II simulation model for the assembly situation described above. The resulting SIMNET II model is given in Appendix B. An investigation of the SIMNET II model should reveal the difficulty of attempting to change the simulation "manually" to reflect the change in input data. The possibility of making logical errors could go undetected because the repetitive nature of the code and the fact that many statements in the model "look alike." The use of the model generator in this case will alleviate this problem altogether regardless of the size or the complexity of the original network. Indeed, the greater the complexity of the network the more useful and necessary this procedure becomes.

```
'Generate Node Statements
FOR i = 1 TO numcs
        s$ = ""
        i$=str$(i)
        SELECT CASE pm$(i)
        CASE "P"
                PRINT #1, USING " \          \ *s;;;;;lim=1:"; n$(i)
                FOR n = 0 TO nbra(i) - 1
                        s$ = "              *b;" + n$(bra(i, n)) + ";;" + _
                        "a(1)="+str$(braunits(i,n))+"*cost("+i$+")*quant;"
                        s$=s$+"v("+STR$(bra(i, n))+")=v("+STR$(bra(i, n))+")+a(1)%:"
                        GOSUB sprnt
                NEXT n
        CASE "M"
                PRINT #1, USING " \          \ *A:"; n$(i)
                FOR n = 0 TO nbra(i) - 1
                        s$ = "              *b;" + n$(bra(i, n)) + "/" + _
                          STR$(nbra(i)) + ";sw(" + i$ + ")=ON?;"
                        s$=s$+"a(1)=(setup("+i$+")+"+timed$(i)+"*quant)*cost("+i$+");"
                        s$=s$+"v("+STR$(bra(i, n)) + ")=v(" + STR$(bra(i, n))
                        s$=s$+")+a(1) +v(" + i$ +");sw("+i$+ ")=OFF%:"
                        GOSUB sprnt
                NEXT n
        END SELECT

NEXT i
```

Figure 3: BASIC Segment for Creating Simulation Model Logic

5 OTHER APPLICATION AREAS

There are two other areas where the procedure proposed here should prove most effective and, indeed, highly desirable. Both areas are in the "spirit" of the assembly line system described above. The first area is the familiar analysis of PERT activity networks using simulation (See [Taha (1988), p. 343]). Another promising area which the authors are currently investigating deals with the evaluation of academic curricula in colleges and universities. A curriculum involves the course work a student must complete to graduate. The objective of the simulation is to determine whether the course offerings and schedule in an academic department will allow a (typical) student to graduate in a certain number of terms. We can experiment with the impact of different schedules and course offerings simply by changing the input data and then using the model generator to create the associated simulation model. Given the large number of academic programs in a typical college, any attempts to change the simulation model itself manually (rather than use the model generator) should indeed be an exercise in futility.

Notice the similarity between both the PERT and course offering areas and the assembly line application described above. In PERT, the source nodes are equivalent to the purchased parts. Similarly, in the course offering area, a course that has no prerequisite is similar to a purchased part. Although the network representing a curriculum is complex, the database structure of the model is straightforward and easy to create. A simulation built directly in a general purpose simulation language would accomplish the purpose. However, it is not easily modifiable from year to year because of the somewhat cryptic nature of simulation programs. It also would not serve well for boiler-plate to be used by other institutions because of the complexity of the network structure that would be used to describe the model.

6 CONCLUSION

This paper has presented an approach for generating simulation models in a language of choice using a model generator. The idea is particularly plausible for those "high use" network-based models which, by their very nature, require making changes in the simulation model itself each time a new situation is investigated. Although our demonstration has concentrated on the use of SIMNET II general purpose simulation language, the approach is equally applicable with any other simulation language. The main difference will be whether or not the language offers sufficient syntactical flexibility to allow the use of a generic model to create the simulation statements iteratively. Our experience shows that

the language must have the capability to accept indexed (or subscripted) naming of its various modeling elements. This property is readily available in SIMNET II.

APPENDIX A: MODEL GENERATOR FOR THE EXAMPLE APPLICATION

```
'*********************************************************
'*QuickBASIC program to generate SIMNET Cost Estimation*
'*   Program and Model by R.B. Taylor 3/13/91           *
'*********************************************************
'Array Declarations
maxpre = 10: maxnd = 80
DIM item(maxnd), n$(maxnd), pm$(maxnd), setup(maxnd)
DIM timed$(maxnd), cost(maxnd), q(maxnd)
DIM comp$(maxnd, maxpre), units(maxnd, maxpre)
DIM pre(maxnd, maxpre), bra(maxnd, maxpre), nbra(maxnd)
DIM plen(maxnd), braunits(maxnd, maxpre)

'Load Arrays from File
OPEN "capcyl.dat" FOR INPUT AS 1
INPUT #1, prjnm$, quant
i = 1
WHILE NOT EOF(1)
INPUT #1, n$(i), pm$(i)
SELECT CASE pm$(i)
CASE "P", "p"
        'P- Item#,Name,P,$/unit
        INPUT #1, cost(i)

CASE "M", "m"
        'M- Item#,Name,M,setup time,time/piece,$/hr,#components,
        '   component#,units, repeat for each component
        INPUT #1, cost(i),setup(i), timed$(i), q(i)
        FOR q = 1 TO q(i)
                INPUT #1, comp$(i, q), units(i, q)
        NEXT q
        case else
END SELECT
        i = i + 1
WEND
numcs = i - 1
CLOSE

'Evaluate Network Structure
FOR i = 1 TO numcs
        IF pm$(i) = "M" THEN
        FOR q = 1 TO q(i)
                IF comp$(i, q) > "" THEN
                        p = 0
                        WHILE n$(p) <> comp$(i, q) AND pre(i, q) <= numcs
                                p = p + 1
                        WEND
                        IF p > numcs THEN
                                PRINT "Component not found "; comp$(i, q): END
                        END IF
                        pre(i, q) = p
                        IF nbra(p) = maxpre THEN
                                PRINT "Exceeds max pre... at "; n$(p): END
                        END IF
                        bra(p, nbra(p)) = i
                        braunits(p, nbra(p)) = units(i, q)
                        nbra(p) = nbra(p) + 1
                        plen(p) = plen(p) + units(i, q)
                END IF
        NEXT q
        END IF
NEXT i
```

```
'Generate Control Statements
OPEN "capcyl.snt" FOR OUTPUT AS 1
PRINT #1, "$PROJECT;"; prjnm$; ";"; DATE$; ";TAYLOR:"
PRINT #1, "$DIMENSION: ENTITY("; maxnd; "),A(2),v("; maxnd; "),"; _
                        "cost(";maxnd; "),setup(";maxnd;"):"
PRINT #1, "$VARIABLES: Total;RUN.END;V("; numcs; "):"
PRINT #1, "$SWITCHES: SW(1-"; STR$(numcs); ");ON:"
PRINT #1, " $BEGIN:"

'Generate Node Statements
FOR i = 1 TO numcs
        s$ = ""
        i$=str$(i)
        SELECT CASE pm$(i)
        CASE "P"
                PRINT #1, USING " \          \ *s;;;;;lim=1:"; n$(i)
                FOR n = 0 TO nbra(i) - 1
                        s$ = "              *b;" + n$(bra(i, n)) + ";;" + _
                        "a(1)="+str$(braunits(i,n))+"*cost("+i$+")*quant;"
                        s$=s$+"v("+STR$(bra(i, n))+")=v("+STR$(bra(i, n))+")+a(1)%:"
                        GOSUB sprnt
                NEXT n
        CASE "M"
                PRINT #1, USING " \          \ *A:"; n$(i)
                FOR n = 0 TO nbra(i) - 1
                        s$ = "              *b;" + n$(bra(i, n)) + "/" + _
                          STR$(nbra(i)) + ";sw(" + i$ + ")=ON?;"
                        s$=s$+"a(1)=(setup("+i$+")+"+timed$(i)+"*quant)*cost("+i$+");"
                        s$=s$+"v("+STR$(bra(i, n)) + ")=v(" + STR$(bra(i, n))
                        s$=s$+")+a(1) +v(" + i$ +");sw("+i$+ ")=OFF%:"
                        GOSUB sprnt
                NEXT n
        END SELECT

NEXT i
PRINT #1, "              *b;TERM;;SIM=STOP%:"
PRINT #1, "$END:"
s$= "$ARRAYS:   cost; 1-1/ns/"
for i=1 to numcs
        s$=s$+str$(cost(i))+";"
next i
mid$(s$,len(s$))=":"
gosub sprnt
s$="           setup;1-1/ns/"
for i=1 to numcs
        s$=s$+str$(setup(i))+";"
next i
mid$(s$,len(s$))=":"
gosub sprnt
print #1, "$CONSTANTS: 1-1/quant=";quant;":"
PRINT #1, "$STOP:"
CLOSE
END

sprnt:  'Line length must be less than 72
        frstlin = -1
        WHILE LEN(s$) > 70
                s = 70
                WHILE MID$(s$, s, 1) <> ";" and s>30
                        s = s - 1
                WEND
                PRINT #1, LEFT$(s$, s)
                s$ = "                " + MID$(s$, s + 1)
        WEND
        IF LEN(s$) > 0 THEN PRINT #1, s$: s$ = ""
RETURN
```

APPENDIX B: SIMNET II SIMULATION MODEL FOR THE EXAMPLE APPLICATION

```
$PROJECT;CAPCYL;03-15-1991;TAYLOR:
$DIMENSION: ENTITY( 80 ),A(2),v( 80 ),cost( 80 ),setup( 80 ): $VARIABLES: Total;RUN.END;V( 12 ):
$SWITCHES: SW(1- 12);ON:
 $BEGIN:
 1.5DTfln      *s;;;;;lim=1:
               *b;TflnCap;;a(1)= .083*cost( 1)*quant;v( 2)=v( 2)+a(1)%:
               *b;TflnTube;;a(1)= .5*cost( 1)*quant;v( 3)=v( 3)+a(1)%:
 TflnCap       *A:
               *b;TflnAsm/ 1;sw( 2)=ON?;
                 a(1)=(setup( 2)+UN(.11,.31)*quant)*cost( 2);
                 v( 4)=v( 4)+a(1) +v( 2);sw( 2)=OFF%:
 TflnTube      *A:
               *b;TflnAsm/ 1;sw( 3)=ON?;
                 a(1)=(setup( 3)+UN(.33,.53)*quant)*cost( 3);
                 v( 4)=v( 4)+a(1) +v( 3);sw( 3)=OFF%:
 TflnAsm       *A:
               *b;CapCyl/ 1;sw( 4)=ON?;
                 a(1)=(setup( 4)+0*quant)*cost( 4);
                 v( 12)=v( 12)+a(1) +v( 4);sw( 4)=OFF%:
 2DStnls       *s;;;;;lim=1:
               *b;MachCap;;a(1)= .167*cost( 5)*quant;v( 6)=v( 6)+a(1)%:
 MachCap       *A:
               *b;CapAsm/ 1;sw( 6)=ON?;
                 a(1)=(setup( 6)+UN(.63,.83)*quant)*cost( 6);
                 v( 9)=v( 9)+a(1) +v( 6);sw( 6)=OFF%:
 Screw         *s;;;;;lim=1:
               *b;CapAsm;;a(1)= 1*cost( 7)*quant;v( 9)=v( 9)+a(1)%:
 CapNut        *s;;;;;lim=1:
               *b;CapAsm;;a(1)= 1*cost( 8)*quant;v( 9)=v( 9)+a(1)%:
 CapAsm        *A:
               *b;CapCyl/ 1;sw( 9)=ON?;
                 a(1)=(setup( 9)+0*quant)*cost( 9);
                 v( 12)=v( 12)+a(1) +v( 9);sw( 9)=OFF%:
 1.5DStnls     *s;;;;;lim=1:
               *b;MachTube;;a(1)= .5*cost( 10)*quant;
                 v( 11)=v( 11)+a(1)%:
 MachTube      *A:
               *b;CapCyl/ 1;sw( 11)=ON?;
                 a(1)=(setup( 11)+UN(.74,.94)*quant)*cost( 11);
                 v( 12)=v( 12)+a(1) +v( 11);sw( 11)=OFF%:
 CapCyl        *A:
               *b;TERM;;SIM=STOP%:
 $END:
 $ARRAYS:   cost; 1-1/ns/ 19.78; 18; 18; 0; 26.16; 18; .09; .09; 0; 14.72; 18; 16:
           setup;1-1/ns/ 0; .5; .75; 0; 0; 2.75; 0; 0; 0; 0; 2.75; .75:
 $CONSTANTS: 1-1/quant= 88 :
 $STOP:
```

REFERENCES

Crooks, J.G. (1987), "Generators, Generic Models, and Methodology" *Journal of the Operational Research Society*, Vol. 38, No. 8, pp. 765-768

Malstrom, E.M. (1981), *What Every Engineer Should Know About Manufacturing Cost Estimating*, Dekker, New York, NY.

Pritsker A. B. (1986), *Introduction to Simulation and SLAM II*, Wiley, New York.

Seppanen, M. S. (1990), "ALSS II: The Advanced Assembly Line System Simulator," *Proceedings of the 1990 Winter Simulation Conference*, O. Balci, R. Sadowski, and R. Nance (eds), SCS, San Diego, California (pp. 625-631).

Taha, H. A. (1988), *Simulation Modeling and SIMNET*, Prentice-Hall, New Jersey.

Taha, H. A. (1990), *Simulation with SIMNET II*, SimTec, Inc., Arkansas.

AUTHOR BIOGRAPHIES

R. BRUCE TAYLOR is an Assistant Professor of Industrial and Manufacturing Engineering at Oregon State University. He holds a B.S. in Systems Engineering from UCLA and an M.S.E. in Computer Science Engineering and a Ph.D. in Industrial Engineering from the University of Arkansas. Before entering academics, he did systems control work in the US Air Force and was a computer specialist with the US Forestry Service. He continues to work with the USFS on field use of hand-held computers. His current teaching and research areas are simulation and computer integrated manufacturing. He is a senior member of Institute of Industrial Engineers (IIE) and is also a member of American Society for Engineering Education (ASEE).
Telephone (503) 737-6073
FAX (503) 737-5241
e-mail Taylorrb@conan.ie.orst.edu

HAMDY A. TAHA is Professor of Industrial Engineering at the University of Arkansas and President of SimTec, Inc. He holds a B.S. degree in Electrical Engineering (Alexandria University, 1958), M.S. degree in Industrial Engineering (Stanford University, 1961), and Ph.D. degree in Industrial Engineering (Arizona State University, 1964). He is the developer of the SIMNET simulation language and the author of four books in operations research and simulation. His most recent book is *Simulation Modeling and SIMNET*, Prentice-Hall, 1988. His consulting experience is focused on the application of operations research and simulation to the oil industry. He has worked on consulting projects in the U.S., Mexico, and the Middle East.
Telephone (501) 575-6031
FAX (501) 575-4346
e-mail HT27009@UAFSYSB.uark.edu