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

A preliminary group technology classification scheme for manufacturing simulation models

Nur E. Ozdemirel, Assoc. Prof. Dr. Gerald T. Mackulak, Assoc. Prof. Dr. Jeffery K. Cochran
Systems Simulation Laboratory
College of Engineering
Arizona State University
Tempe, AZ 85257-5106

ABSTRACT

Although computer simulation is a well-studied and highly structured research area, its use in industry is still limited due to the fact that simulation modeling requires expert knowledge. We believe that this expertise must be embedded in an integrated simulation software to provide non-expert user access to this valuable tool. Simulation experts know very well that there are similarities between the models they build for different manufacturing systems. In this paper we present a GT classification scheme for the manufacturing simulation models which is developed by observing those similarities. The GT scheme will constitute a basis for creating a model base consisting of generic models and submodels, within the framework of an intelligent simulation environment. Specific user models will be constructed by coupling, modifying, and configuring the generic models rather than building from scratch, reducing the effort expected of the user in the process of model construction.

1. TAXONOMICAL APPROACHES TO SIMULATION

Classification of simulation models is not an entirely new idea. In fact various simulation taxonomies available in the literature are commonly known and already included in many simulation textbooks. Among them, perhaps the most elementary classification is *discrete* simulation versus *continuous* simulation which is based on how the values of the system state variables change. Other basic classifications are *static* versus *dynamic* simulation, and *deterministic* versus *stochastic* simulation.

In discrete event simulation, strategy-related characteristics of simulation languages or models constitute a basis for further classification. Hooper (1982) presents an algorithmic analysis of the three simulation strategies: *event scheduling*, *activity scanning*, and *process interaction*. The algorithmic differences between the strategies are, in most cases, not transparent to the user and will not be discussed here. World view of a language, however, differs depending on the simulation strategy; when a certain language is used in developing a model, the model must be formulated in terms of concepts consistent with the world view of that language.

Highland (1977) proposes additional classification criteria based on the following characteristics of simulation models:

- purpose of simulation (e.g., system evaluation, prediction, optimization).
- model characteristics (i.e., time frame, system composition, system size, system state, environmental interaction, and elemental components).
- relationships among entities (e.g., symbiotic, synergistic, replicative, antithetic).
- attribute characteristics (e.g., time/space relationships and volatility).
- variable characteristics (i.e., statistical nature of the variables, interrelationships between the variables, environmental variables, and impact state variables).

Finally, a common way of classification found in the literature is by the application area. Highland (1977) groups simulation applications in six main classes each of which is further divided into subclasses. The main classes are:

- Computer systems
- Business, industry, & agribusiness
- Governmental & social systems
- Ecological & environmental systems
- World modeling
- Biosciences

The above well-known classification schemes are useful in documentation and training, but they offer limited information regarding the simulation model structure. They do not, for example, tell what the basic simulation entities are for a particular domain, how those entities interact, and what kind of material and information flow characteristics are involved. They also lack information regarding the structure of simulation routines and how those routines should be interfaced. The purpose of this paper is to discuss the development of a model base consisting of generic models and/or modules from which specific models can be constructed. The constructed models must satisfy specific user requirements or goals. These models will allow creation of simulation scenarios without user abstraction, a skill that has been found lacking in non-expert modelers. To achieve these objectives, the knowledge regarding model structures must be built into the model base. We have chosen to define this knowledge through specification of a classification scheme. This classification scheme will constitute a framework for the development of such a model base, and will be based on the structural properties of simulation models.

2. TOWARDS THE AUTOMATION OF SIMULATION MODEL DEVELOPMENT

As a result of recent developments in applied artificial intelligence and knowledge based systems (KB), many researchers from other disciplines focused their attention on the potential utilization of AI/ES techniques in their areas. KB can assist simulationists in many ways such as model construction, input preparation, and output analysis (designing and performing statistical experiments). The emphasis of this research is to build a generic simulation model base, and to assist the user in developing the appropriate specific model for his/her purposes through interaction with the model base.

The literature reveals that efforts for creating an intelligent simulation environment are numerous. Although various approaches are taken by different researchers, four main schools of thought are identified with regard to model construction: hierarchical modular model development, object-oriented simulation, rule-based modeling, and intelligent user interfaces.

Hierarchical modular model development: This approach is based on the DEVS formalism developed by B. P. Zeigler which is discussed in many articles, including Rosenblit (1986), and Zeigler (1984, 1987). It can be outlined, in simple terms, as (a) decomposing the system to be simulated into its components to obtain a hierarchical modular structure, (b) building submodels for each component, and (c) coupling those submodels in an appropriate manner to come up with a final form of the model. An implementation of this approach, GEST, is described in Oren (1984). A recent implementation in the LISP-based, object-oriented environment of PC-Scheme is discussed in Zeigler (1987).

Object-oriented simulation: The concepts objects and classes of objects are not entirely new to simulation. They were introduced in Simula, one of the oldest simulation languages. These concepts, however, and the programming languages to handle them have not been matured prior to the advancement of AI techniques. The object-oriented simulation is based on defining the simulation objects and interactions among them by using their descriptive, structural, behavioral, and taxonomical properties. Among the implementations of object-oriented simulation are ISIS (Fox 1984), Transcell (Bourne 1984), ROSS (Faught 1980) and Smalltalk (Ulgen 1986).

Rule-based modeling: Production rules are utilized in some object-oriented implementations such as ISIS and ROSS. A group of authors suggest using purely rule-based (goal-oriented) programming languages, such as Prolog, for simulation modeling. A goal-oriented simulation language, T-Prolog, is presented in Adelsberger (1984). T-CP (Cleary 1985), which is based on Concurrent Prolog, is another language of this sort. Swaan (1983) suggests using production rules within the context of subsystem simulation approach. Production rules can be used to build a number of submodels for the subsystems of a system, and will serve as the building blocks of the final model.

Intelligent user interfaces: An intelligent user interface or intelligent front end is a program which facilitates automated model construction and analysis in a conventional simulation language. Implementations in the literature differ on a wide range in terms of model-building capabilities. Four groups can be identified:

1. A single general purpose model to be configured (e.g., Miles 1986).
2. A number of complete generic models to be modified (e.g., Mackulak 1987).
3. Model construction from smaller modules (e.g., Medeiros 1983).
4. Model building from scratch using simulation languages (e.g., Haddock 1987)

3. THE INTELLIGENT SIMULATION ENVIRONMENT

Some of the implementations mentioned above, such as Zeigler's PC-Scheme implementation of hiererchical modular modeling, object oriented Smalltalk, and rule based T-Prolog, provide general purpose programming tools for developing simulation models in a modular fashion, and emphasize that the models must be built from smaller modules. None of them, however, attempt to identify the modules that might be required for a particular domain such as discrete part manufacturing. The user is supposed to identify the simulation entities, their attributes, and the way those entities interact. The user also has to construct the modules and specify the constraints under which they can be coupled. Too much is expected of the user in terms of learning new languages, gathering and organizing information related with simulation entities, and coding model modules.

Some implementations, such as ISIS or Transcel, come with the built-in description of the entities and modules, but these are problem-specific and too restrictive to be general-purpose packages.

Cochran (1987) lists the features of a perfect simulation environment, including no user coding of any kind, and pre-existing (generic) models to modify rather than build. Generic models and/or modules are indeed essential to develop a general-purpose simulation package for the discrete manufacturing domain which eliminates user coding at the same time. The ultimate purpose of the GT classification scheme presented in the following section is to identify and construct the generic modules in a model base, and use them as building blocks of complete models.

4. THE GT CLASSIFICATION AND CODING SCHEME

In developing the classification scheme, first the main components of a simulation model (and a system) are identified from a structural perspective. Those components are outlined in Figure 1. A subscheme is developed for each main component based on a GT approach. In Figures 2 through 6, those subschemes are illustrated again by tree diagrams. Nodes in a tree diagram represent subcomponents of a main component, or properties of those components. A horizontal line joining the branches of a tree means that one or more of the following components may be included in the model, whereas a regular branching corresponds to an "either/or" situation. Based on the tree diagrams, a GT coding subscheme is created by defining a group of digits for each component. The overall coding scheme will be obtained simply by appending groups of digits. The coding scheme is a hybrid one rather than a monocode or polycode, hence the total number of digits varies for each model.

In the following sections, the subschemes developed for each of the main system components--including stations, material handling equipment, jobs, queues, equipment breakdowns and scheduled maintenance--are discussed in detail.

4.1. Stations

All kinds of resources that may be found in a discrete manufacturing system are considered under this component with the exception of material handling resources.

The concept of work stations is a common way of classifying manufacturing activities on the shop floor. Therefore, the basic shop floor unit is assumed to be a work station with a number of identical parallel servers. Three types of stations are identified: machining, assembly, and inspection. Machining stations are for typical discrete manufacturing processes such as drilling and milling of a single type of job at a time. Assembly

operation is treated separately because it involves multiple jobs arriving at a station as separate entities, and leaving the station as a single entity. Inspection differs from both machining and assembly operations since it implies a branching (defectives/nondefectives) which is a structural model property.

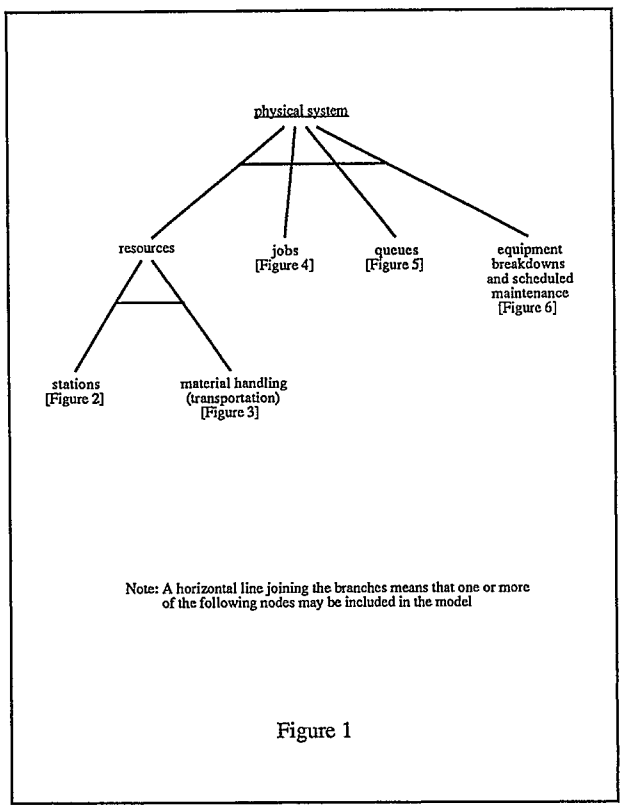

Figure 1

Presence of a station means that there are one or more identical servers which will be treated as the primary resource. Interpretation of the primary resource is up to the user. It may be a machine tool for a machining station, inspector for an inspection station, or simply a bench on which the assembly operation is carried out. The primary resource will be seized when a job arrives at the station, and released when it leaves.

In Figure 2, the branching at the machining stations (node A) describes features of the stations which will be repeated for assembly and inspection stations as well. There may be one or more secondary resources for each type of station such as human operators, robots, and transportable tools (jigs, fixtures, cutters, etc.). Secondary resources are those that are shared by the parallel servers of a station (within station), or by a number of stations (between stations). If a secondary resource is not shared at all, there is no point in treating it as a separate entity. For example, if a human operator is assigned to one and only one machine tool, the machine and its operator will be represented as a single entity, the primary resource.

In a flexible manufacturing system, work stations may be automated at various levels. For example, actual processing may be completely automated at a station, but the machine setup may require a human operator. Therefore, unlike the primary resource, secondary resources are not necessarily utilized during the entire operation. An operation is divided into three suboperations--setup, machine loading/unloading, and processing--which define the usage pattern of the resources. A secondary resource may be used for any combination of these suboperations.

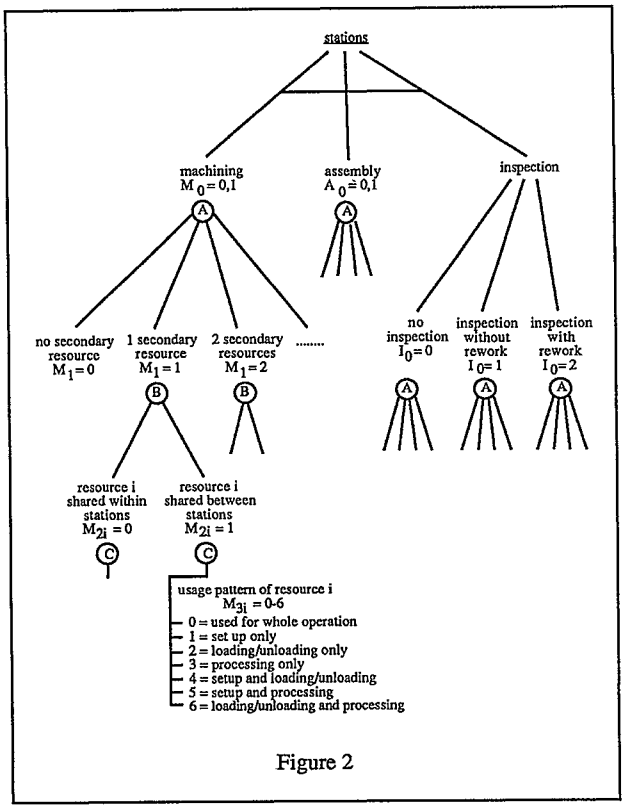

Figure 2

Coding scheme.

machining : $M_0M_1(M_{21}M_{31})(M_{22}M_{32})\ldots$
assembly : $A_0A_1(A_{21}A_{31})(A_{22}A_{32})\ldots$
inspection : $I_0I_1(I_{21}I_{31})(I_{22}I_{32})\ldots$

where
M_0, A_0 = 0,1 (machining and assembly stations exist or not)
I_0 = 0,1,2 (no inspection, inspection without rework, inspection with rework)
M_1, A_1, I_1 = 0,1,2,...,r (number of secondary resources)
M_{2i}, A_{2i}, I_{2i} = 0,1 for i = 1,2,...,r (ith resource is shared within or between the stations)
M_{3i}, A_{3i}, I_{3i} = 0-6 for i = 1,2,...,r (usage pattern of ith secondary resource).

Examples. 10-0-10 : machining stations without secondary resources, no assembly stations, inspection without rework and with no secondary resources.

10-12(03)(16)-21(00) : machining stations with no secondary resources, assembly stations with two types of secondary resources (first one shared within the assembly stations and used for processing only, second

one shared by all the assembly stations and used for loading/unloading and processing), inspection with rework, and one secondary resource which is shared within the inspection stations and used during the entire inspection operation.

With this scheme, it is possible to code a wide range of manufacturing systems having discrete part processing, inspection, or assembly operation only, or any combination of the three. Furthermore, the scheme supports different levels of abstraction. Secondary resources can be omitted for a higher level of abstraction, or their usage pattern can be ignored by specifying M_3, A_3, I_3 as 0 (used for the entire operation).

4.2. Material Handling

By material handling, transportation between the stations is implied. Material handling within the stations is ignored since a station consists of identical parallel servers, but machine loading/unloading is taken into account in the stations component.

Various types of material handling equipment may be used in a simulation model (see Figure 3). The equipment type can be specified for each pair of stations when configuring the model, or each job type can be assigned an equipment type depending on the features of the job, such as size or weight. In the former case, the linkage between the stations will be fixed throughout a simulation run, whereas it will be flexible in the latter case.

Coding scheme. $T_1T_2T_3T_4$
where
- T_1, T_2, T_3 = 0, 1 (existence of manual transportation, conveyor and agv)
- T_4 = 1, 2 (equipment specified for station pairs or job types).
- T_4 will be used when more than one of the previous digits is nonzero.

Examples. 000 : transportation is ignored.
001 : agv only.
0112 : no manual transportation, conveyor, and agv, equipment type specified for each job type (flexible).

4.3. Jobs

Jobs are the entities flowing through the system, such as orders, parts, or subassemblies. In Figure 4, jobs are classified as discrete parts and assembly jobs. Discrete parts are individual parts or batches that do not require assembly. Assembly jobs may be physical assemblies, or paper work and the raw material to manufacture an order can also be treated as an assembly job since production cannot begin unless both are available. A combination of the two is also possible. For example, two parts may be assembled after being processed separately at different machining stations, and the subassembly may be routed to another machining station for further processing. For consistency purposes, discrete parts and assembly jobs may be specified only if machining stations and assembly stations exist, respectively.

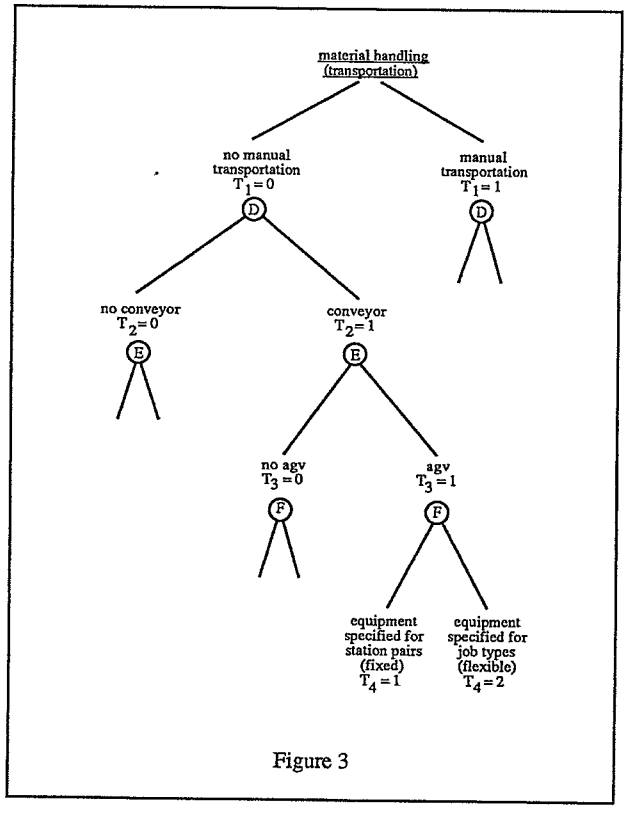

Figure 3

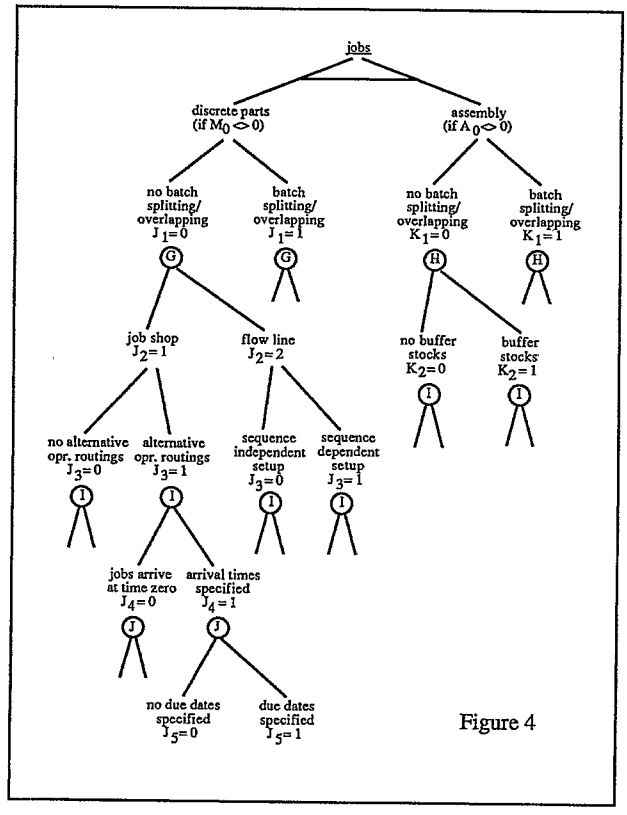

Figure 4

Jobs are thought of as batches since, from model structure perspective, other types of discrete manufacturing--such as mass production, flow line, and job shop--are all special cases of batch production. If batch splitting/overlapping is to be allowed, then conditions for splitting and overlapping must be specified when configuring the model.

An important characteristic of a job shop is alternative operations routings for the jobs. In flow line production, on the other hand, the typical question is whether the set up times are sequence dependent or not. For this reason, J_2 is used to differentiate between the two production types, and J_3 is specified accordingly.

In assembly line balancing, estimating the buffer stock sizes is a significant problem, and the scheme will allow the existence of buffer stocks by choosing K_2 as 1. Note that subcontracted parts can also be simulated using the buffer stock concept.

The user may also want to specify nonzero arrival times and due dates for assembly and discrete manufacturing jobs. This will be necessary especially when simulating "hot jobs." The last two digits of both discrete and assembly job codes deal with this property.

Coding scheme. $J_1J_2J_3J_4J_5$ - $K_1K_2K_3K_4$

where

J_1, K_1 = 0,1 (batch splitting/overlapping required or not)
J_2 = 1, 2 (job shop or flow line)
J_3 = 0, 1 (varies depending on the choice of J_2)
K_2 = 0, 1 (existence of buffer stocks)
J_4, K_3 = 0, 1 (arrival times specified or not)
J_5, K_4 = 0,1 (due dates specified or not).
J_i will be used if M_0 <> 0, and K_i will be used if A_0 <> 0.

4.4. Queues

Queues are where the jobs and the resources interact, and they must be treated as separate entities (see Figure 5). Obviously, the most important feature of the job queues is the dispatching rule. A number of alternatives can be considered for removing the jobs from the queues, including FIFO, SPT, LPT, SRPT, LRPT, etc. Choice of the dispatching rule does not change the model structure, but it may affect the attribute set. For example, if the minimum slack (due date - remaining processing time) is to be the criterion for removing entities from the queues, we need to ask for and store due dates in one of the transaction attributes.

Remaining digits for the queues are self-explanatory, except that parallel queues for each station (Q_2 = 1) may not be feasible because of the limitations of using a microcomputer.

Coding scheme. $Q_1Q_2Q_3Q_4Q_5$

where

Q_1 = 1- ? (dispatching rule)
Q_2 = 0, 1 (single queue or multiple queues per station)
Q_3 = 0, 1 (queue sizes limited or not)
Q_4 = 0, 1 (balking option)
Q_5 = 1, 2 (push or pull inventory system).
Q_4 will be used if Q_3 = 1, and Q_5 will be used if Q_4 = 1.

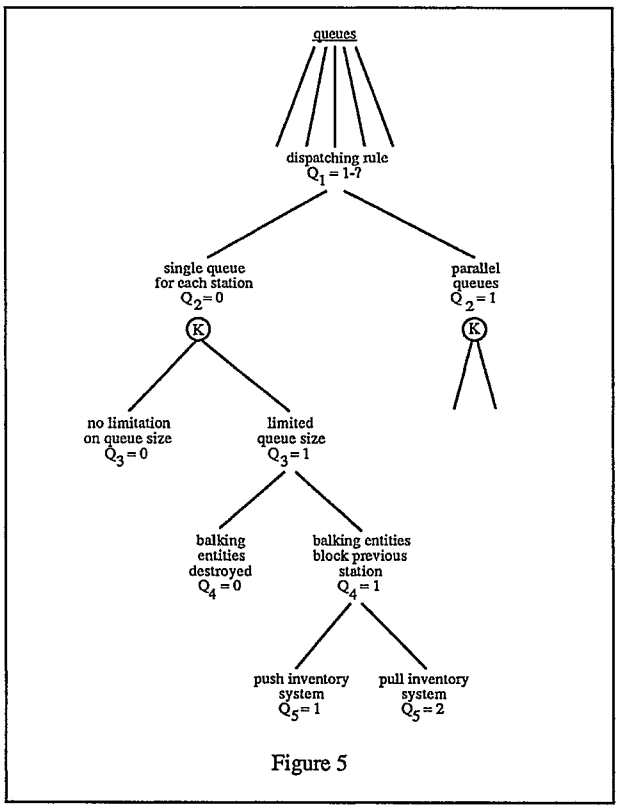

Figure 5

4.5. Equipment Breakdowns and Scheduled Maintenance

Equipment breakdowns are one of the main sources of uncertainty in the manufacturing environment. In addition, scheduled maintenance activities may cause unavailability of resources for certain time periods. The coding scheme which is illustrated in Figure 6 considers both equipment breakdowns and scheduled maintenance. Separate model modules can be created to simulate breakdowns and/or maintenance, and can be added to the final model if the user wants to simulate them. Breakdown and maintenance modules will be very similar, except that in the former interarrival times between breakdown transactions will be probabilistic, whereas in the latter those will be constant.

Coding scheme. B_1B_2

where

B_1 = 0-3 (breakdowns and/or maintenance considered or not)
B_2 = 1-5 (resource types involved).

Maximum length of the overall coding scheme can be estimated although it will vary from one model to another.

Stations	: 18	digits (two secondary
Material handling	: 4	resources per
Jobs	: 9	station)
Queues	: 5	
Equipment breakdowns & scheduled maintenance	: 2	
	38	digits

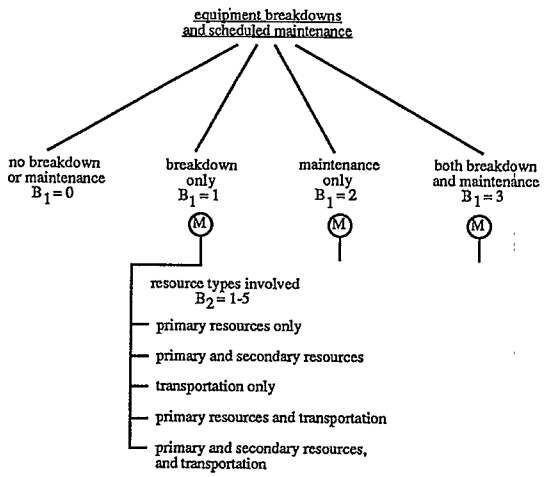

Figure 6

5. EVALUATION

The scheme has been developed based on expert knowledge of discrete manufacturing models and review of 40 simulation studies found in the literature. Those studies have been carefully analyzed in terms of the characteristics of physical systems, modeling assumptions, and objectives of simulation. The common entities and their properties are identified and used in deciding the digits of the scheme. Our goal is to develop a scheme which can classify and code 85 to 90 percent of all discrete manufacturing models. A sample size of 40 is not sufficient. Therefore, we plan to review 200 to 300 models, and try to code them for evaluating and improving our scheme. We also want to use cluster analysis techniques to identify families of models to decide on what generic models and/or modules will be needed in the model base to simulate a significant portion of the systems. Our results through application of this coding scheme will be reported in a later publication.

REFERENCES

Adelsberger, H.H. (1984). "Prolog as a Simulation Language." Proceedings of the 1984 Winter Simulation Conference, Dallas, Texas, 501-504.

Bourne, D.A. and M.S. Fox (1984). "Autonomous Manufacturing: Automating the Job-Shop." *Computer*, 17:9, Sep 1984, 76-86.

Cleary, J., K. Goh, and B. Unger (1985). "Discrete Event Simulation in Prolog." *AI, Graphics and Simulation* (G. Birtwistle, ed.), The Society for Computer Simulation, San Diego, CA, 1985, 8-13.

Cochran, J.K., G.T. Mackulak, D. Castillo, and E. Du (1987). "Configuring Available Software into an AI/ES Environment for Manufacturing Simulation Design on the PC." Proceedings of the Conference on Simulation of Computer Integrated Manufacturing Systems and Robotics, San Diego, CA, Jan 1987, 1-7.

Faught, W.S., P. Klahr, and G.R. Martins (1980). "An Artificial Intelligence Approach to Large-Scale Simulation." Proceedings of the 1980 Summer Computer Simulation Conference, Seattle, WA, 231-239.

Fox, M.S. and S.F. Smith (1984). "ISIS - A Knowledge-Based System for Factory Scheduling." *Expert Systems*, 1:1, Jul 1984, 25-49.

Haddock, J. (1987). "An Expert System Framework Based on a Simulation Generator." *Simulation*, 48:2, Feb 1987, 45-53.

Highland, H.J. (1977). "A Taxonomy Approach to Simulation Model Documentation." Proceedings of the 1977 Winter Simulation Conference, Gaitherburg, Maryland, 725-729.

Hooper, J.W. and K.D. Reilly (1982). "An Algorithmic Analysis of Simulation Strategies." *International Journal of Computer and Information Sciences*, 112, Feb 1982, 101-122.

Mackulak, G.T., J.K. Cochran, and D.L. Shunk (1987). *Generic System Simulation Development on the PC (Phases I-III)*, Arizona State University, Tempe, AZ, 1987.

Medeiros, D.J. and R.P. Sadowski (1983). "Simulation of Robotic Manufacturing Cells: A Modular Approach." *Simulation*, 40:1, Jan 1983, 3-12.

Miles, T. (1986). "Scheduling of a Manufacturing Cell With Simulation." Proceedings of the 1986 Winter Simulation Conference, Washington, D.C., 668-676.

Oren, T.I. (1984). "Model Based Activities: A Paradigm Shift." *Simulation and Model Based Technologies: An Integrative View* (T.I. Oren, B.P. Zeigler and M.S. Elsaz, eds.), Nato ASI Series, F10, Springer-Verlag, Berlin, W. Germany, 1984, 3-40.

Rosenblit, J.W. and B.P. Zeigler (1986). "Entity-based Structures for Model and Experimental Frame Construction." *Modelling and Simulation Methodology in the Artificial Intelligence Era* (M.S. Elsaz, T.I. Oren and B.P. Zeigler, eds.), Elsevier Science Publishers, B.V., Netherlands, 1986, 79-100.

de Swaan Arons, H. (1983). "Expert Systems in the Simulation Domain." *Mathematics and Computers in Simulation*, 25:1, Feb 1983, 10-16.

Ulgen, O.M. (1986). "Simulation Modeling in an Object-Oriented Environment Using Smalltalk-80." Proceedings of the 1986 Winter Simulation Conference, Washington, D.C., 474-484.

Zeigler, B.P. (1984). "System-Theoretic Representation of Simulation Models." *IIE Transactions*, 16:1, Mar 1984, 19-34.

Zeigler, B.P. (1987). "Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment." *Simulation*, 49:5, Nov 1987, 219-230.

AUTHORS' BIOGRAPHIES

NUR E. OZDEMIREL is a Ph.D. candidate in the Department of Industrial and Management Systems Engineering at Arizona State University. She received her B.S. and M.S. degrees in industrial engineering from Middle East Technical University, Ankara, Turkey. Her research interest focuses on generic simulation model development for manufacturing systems. She is a member of SCS.

GERALD T. MACKULAK, Ph.D. is currently an Associate Professor of Engineering in the Department of Industrial and Management Systems Engineering at Arizona State University. He is also Director of the Systems Simulation Laboratory (SSL), a newly created laboratory within the CIM Systems Research Center (CIMSYRC). The Systems Simulation Laboratory has as its' charter the development of new simulation tools and methodologies, with specific concentration in the areas of expert systems and statistical post-processing. Dr. Mackulak received all his degrees from Purdue University in the area of Industrial Engineering, and before joining ASU eight years ago he held positions with U.S. Steel, Burroughs and the simulation consulting firm of Pritsker and Associates. Dr. Mackulak has published extensively in the area of CIM integration methods and has taught both public and private seminars on CIM systems integration, simulation and economic justification of CIM implementations. His current research is centered around developing expert systems that enable even novice users to effectively use simulation.

Industrial & Management Systems Engineering
College of Engineering & Applied Sciences
Arizona State University
Tempe, AZ 85287

JEFFERY K. COCHRAN, Ph.D, P.E. is Associate Professor of Industrial and Management Systems Engineering at Arizona State University. He received the Ph.D. in Industrial Engineering (Operations Research) from Purdue University in 1984, whereupon he joined the ASU faculty as Assistant Professor. At ASU he is also Co-Director of the Systems Simulation Laboratory. Through this laboratory Dr. Cochran is pursuing research interests in intelligent simulation environments, stochastic system optimization, statistical expert systems, microcomputer-based operations research, and decision support, analysis, and graphics methodology. His industrial experience includes work at Battelle Northwest Laboratory, Los Alamos National Laboratory, and NASA's Laboratory for Application of Remote Sensing. He is a member of IIE, ORSA, ASEE, SIAM, SCS and is a Registered Professional Engineer in Arizona.

Industrial & Management Systems Engineering
College of Engineering & Applied Sciences
Arizona State University
Tempe, AZ 85287