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

A FRAMEWORK FOR SIMULATION DESIGN OF FLEXIBLE MANUFACTURING SYSTEMS *

Marco Chierotti
Jerzy W. Rozenblit

Department of Electrical and
Computer Engineering
The University of Arizona
Tucson, Arizona 85721
U.S.A

Witold Jacak†

Department of Systems and
Information Science
University of Linz
A-4040 Linz
Austria

ABSTRACT

A framework is being developed for simulation-based design of flexible manufacturing systems. The framework integrates generation of assembly plans, design and configuration of the manufacturing facility and equipment, synthesis of task oriented robot programs, and simulation of a manufacturing system. In this paper, each layer is briefly summarized and a simulation case study of a system for electric motor assembly is presented.

1 METHODOLOGY

In recent years, the use of programmable and flexible systems has enabled partial or complete automation of product machining and assembly. The economic pressure for increases in quality, productivity, and efficiency of manufacturing processes has motivated the development of more complex and detailed design methodologies for flexible manufacturing systems (Kusiak 1990). The proposed simulation-based framework integrates generation of assembly plans, design and configuration of a manufacturing facility and equipment, synthesis of task oriented robot programs, and simulation of the manufacturing system.

In our methodology, the design process is driven by manufacturing objectives. First, a technological process is established. This plan is a sequence of machining and/or assembly operations. The Task Planning Layer organizes the techological process into a feasible sequence of elementary operations.

1.1 Task Planning

Task planning methods generate a task-level plan which describes the decomposition of the machining and/or assembly task into the sequence of elementary operations of robot and devices, the assignment of machining subtasks to system resources, and a model for the coordination and scheduling of system resources in a flexible manufacturing system (FMS).

An FMS is a set of programmable machines (technological devices) and product stores (buffers) connected by a flexible material handling facility (such as a robot or an automated guided vehicle), and controlled by a computer connected with a system of sensors.

An FMS can perform such technological operations as fabrication, machining, or assembly. The state of an FMS is monitored by the sensory system. The FMS state depends on the states of every machine and store. In order to generate a plan of a task realization, actions have to be applied to affect a change of the FMS state. In general, an action can be applied to a sensors-monitored state of an FMS if a set of preconditions are met. An implementation of the production plan can be therefore seen as a sequence of FMS state transitions obtained by applying fundamental actions executable by the system. To establish the sequence of actions and their preconditions list, the task planning problem is divided into two subproblems: a) the techonological process planning problem (called production route planning problem), and b) the deadlock avoidance planning problem.

The production route planning goal is to find an ordered sequence of technological operations with a minimum number of deadlock instances. The technological process planning is based on the description of the operations of the machining or assembly process, the precedence relation over the set of operations, the description of the FMS geometry, and the description of resources. To solve this problem, an AND/OR graph representation is used (Homem De Mello and Sanderson 1990), (Sanderson, Homem De Mello, and Zhang 1990). A decomposition of the machining task corresponds to a cut set of this graph. Feasible de-

*Supported in part by Siemens Corporate Research, Princeton, New Jersey

†On leave from the Institute of Technical Cybernetics, Technical University of Wroclaw Wroclaw 50-370, Poland

compositions, with respect to a precedence relation of assembly operations, are used to create an AND/OR graph that represents all valid operation sequences.

A production route is a set of ordered sequences of technological devices or stores (called resources) required by successive operations.

Once a fundamental plan is defined as a sequence of robot actions, we determine the programs for robot operations and the geometric trajectories of robot movements that implement each action. This is accomplished in the Task-Level Programming Layer.

1.2 Task-Level Programming

The action plan for an assembly task determines the robot's program of operations needed to service the process. Such a program is a sequence of instructions (motion, grasp, and sensors instructions) expressed in a Task-Oriented Robot Programming Language (TORPL). Each elementary action has an associated set of instructions in such a language (Lozano-Perez 1989).

The fundamental function of the Task Programming Layer is to synthesize the robot's motion trajectories that realize the MOVE and GRASP instructions. Trajectories also determine the duration of the moves. To generate the trajectories, we must have available the geometrical models of all the machines and stores of the production system as well as models of the robot's kinematics and dynamics.

The robot's motion trajectory planning process is decomposed into two subproblems: 1) planning of the collision-free geometric track of motion, and 2) planning of the motion dynamics along the computed track.

The planner determines the collision-free track of the robot motion from the initial to the final effector locations based on a) geometric and kinematic description of the robot, and b) its environment and the initial and final positions of the effector-end. This problem has been addressed in various ways and is widely reported in literature (Brooks 1983), (Jacak 1989a), (Lozano-Perez 1989). The methods which solve the problem in question depend on the assumed mathematical model of the robot's kinematics. Now, the optimal speed and acceleration of movements along the computed track should be computed. This task is solved by the trajectory planner.

The trajectory planner receives the geometrical tracks as input and determines a time history of position, velocity, acceleration and input torques which are then input to the trajectory tracker. On this level, the robot is represented by the manipulator dynamics model (Shin and McKay 1986), (Jacak 1989b).

Hence, we can obtain an optimal trajectory and the time of manipulator movement along a geometrical track. Such a planner can generate variant interpretations of robot action plans. For each instruction of the robot's program, we can change the geometry of the motion or change the motion dynamics along the track by selecting criteria of minimal-time or minimal-energy planning. Variant interpretations of the language instructions result in different realizations of robot actions. This motivated us to introduce a procedure that would automatically verify the semantics of the robot's program. This procedure is described briefly in the next section.

1.3 Discrete Event Simulation of Robot Actions

The variants of motion interpretation obtained from the motion planning level are tested by a simulator. Simulation is used to select the most effective variant. The program synthesis process requires that we introduce conditional instructions which depend on the states of each machine d_i of the machining/assembly line and the operational instructions that realize robot actions. Thus, to define a simulator of the program, we model conditions that enable program instructions. Each machine d_i has the following DEVS (Zeigler 1984) representation:

$$Dev_i =< X,\ S,\ Y,\ \delta_{int},\ \delta_{ext},\ \lambda,\ t_a >$$

where:

$$X_i = \{x0_i, x1_i, x2_i\} \ \mid \ i = 1, \ldots, L$$

is the set of external events defined as follows:
$x0_i$ = SETUP working parameters
$x1_i$ = PLACE part ON i-th position,
$x2_i$ = PICKUP part AT i-th position,

The state set is defined as:

$$S_i \ = \ \{s_a, s_b, s_c, s_d, s_e\}$$

where:

- s_a signifies that **machine is off line**
- s_b signifies that **machine is free**
- s_c signifies that **machine is loading**
- s_d signifies that **machine is busy processing an operation**
- s_e signifies that **machine has completed an operation and is not free**

The internal transition function for each machine i is given as follows:
$\delta^i_{int}(s^i_a) = s^i_a$
$\delta^i_{int}(s^i_b) = s^i_b$
$\delta^i_{int}(s^i_c) = s^i_c$
$\delta^i_{int}(s^i_d) = s^i_e$
$\delta^i_{int}(s^i_e) = s^i_e$

The external transition function for each machine i is defined as:
$\delta^i_{ext}((s^i_a, x0_i) = s^i_b$
$\delta^i_{ext}((s^i_b, x1_i) = s^i_c$
$\delta^i_{ext}((s^i_c, x1_i) = s^i_d$
$\delta^i_{ext}((s^i_e, x2_i) = s^i_b$
$\delta^i_{ext}((.,.),.) = failure$ for all other states

The output function is simply defined as $\lambda^i(s) = s$.

The time advance functions for Dev_i determine the time needed to process a part in the i-th machine. They are defined as follows: if $s = s^i_d$, then $ta^i(s) = \tau^k_i$, otherwise $ta^i(s) = \infty$, where τ^k_i is the tooling/assembly time of operation k for machine i.

The above specification defines a model of the technological machines. The activation of each machine Dev_i is caused by an external event generated by the model of the robot. This model is realized by a generator of an experimental frame component (Jacak and Rozenblit 1991) associated with the production system model. Since the events generated by each robot depend on the states of the workcells $Dev_i|i = 1, \ldots, L$, we define an acceptor which observes the state of each workcell.

Rather than provide a detailed mathematical description of the experimental frame models here, we describe their functionality. (The reader is referred to (Jacak and Rozenblit 1991) for a complete formal specification of the simulator.) The acceptor is a DEVS that receives as input state descriptions of each machine Dev_i. It selects events which invoke the robot to service a workcell. The acceptor state set is a class of subsets of indexes of workcells Dev_i. The state contains indexes of only those workcells which have completed processing of a part and from which the part can be transported to another workcell (i.e., the preconditions of the next operation are satisfied). The states of the acceptor also determine state components of the frame generator that models the behavior of every robot.

The DEVS-model of each robot contains the state set $S_R = S_a \times Positions \times HS$, where S_a is the state set of the acceptor, $Positions$ is the set of positions of the robot's effector-end in the base-Cartesian space, HS is the set of states of the effector, i.e., $HS = \{Empty, Holding\}$. Jacak and Rozenblit (1991) demonstrate how the above discrete event specifications can be translated into a set of instructions in TORPL.

Design of a manufacturing facility capable of carrying out an assembly sequence plan is an integral phase of the proposed framework. Resources, i.e., machines, robots, material handling devices, and computer hardware must be integrated in a manner most conducive to the realization of the plan. Knowledge-Based Simulation Design methodology is applied to accomplish the integration.

1.4 Knowledge-Based Simulation Design Methodology

Knowledge-Based Simulation Design methodology is applied to accomplish the integration of plant components and resources. The system design approach proposed by Rozenblit and Zeigler (1990), termed Knowledge-Based Simulation Design, focuses on the use of modeling and simulation techniques to build and evaluate models of the system being designed. It treats the design process as a series of activities which include specification of design levels in a hierarchical manner (decomposition), classification of system components into different variants (specialization), selection of components from specializations and decompositions, development of design models, experimentation and evaluation by simulation, and choice of design solutions.

The design model construction process begins with developing a representation of design components and their variants. To appropriately represent the family of design configurations, a representation scheme called the system entity structure (SES) has been proposed. The scheme captures the following three relationships: decomposition, taxonomy, and coupling. The synthesis (coupling) constraints impose a manner in which components identified in decompositions can be connected together. The selection constraints limit choices of variants of objects determined by the taxonomic relations.

Beyond this, procedural knowledge is available in the form of production rules. They are used to manipulate the elements in the design domain by appropriately selecting and synthesizing the domain's components. This selection and synthesis process is called pruning. Pruning results in a recommendation for a model composition tree, i.e., the set of hierarchically arranged objects corresponding to model components.

Performance of design models is evaluated by simulation. A simulation experiment is defined using the experimental frame concept. Briefly, an experimen-

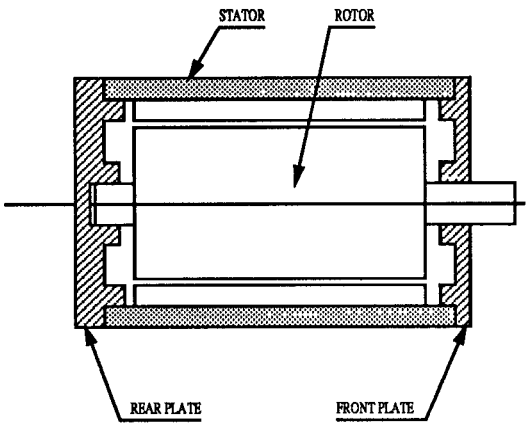

Figure 1: Electric Motor

tal frame defines a set of input, control, output, and summary variables, and input and control trajectories. These objects specify conditions under which a model can be observed and experimented with. It is usually realized as a coupling of three components: a generator (supplying a model with an input segment reflecting the effects of the external environment upon a model), an acceptor (a device monitoring a simulation run), and a transducer (collecting and processing model output data).

The simulation phase of the design framework is followed by evaluation of simulation results and ranking of alternative design models in respective experimental frames. Design models that best conform to design performance criteria, constraints, and requirements serve as the basis for the proposed design solution.

The following example shows the application of the methodology to a simple FMS design problem.

2 CASE STUDY

The example presented here demonstrates different plans of sequencing operations (operations scheduling problem), selection of devices (machines, material handling systems, and robots) to carry out the operations, synthesis of a program for robots servicing the devices, simulation modeling, and testing and verification of design variants based on the simulation models of the overall system architecture.

2.1 Product Description

The example illustrates design of a system for an electric motor assembly. The motor is shown in Figure 1.

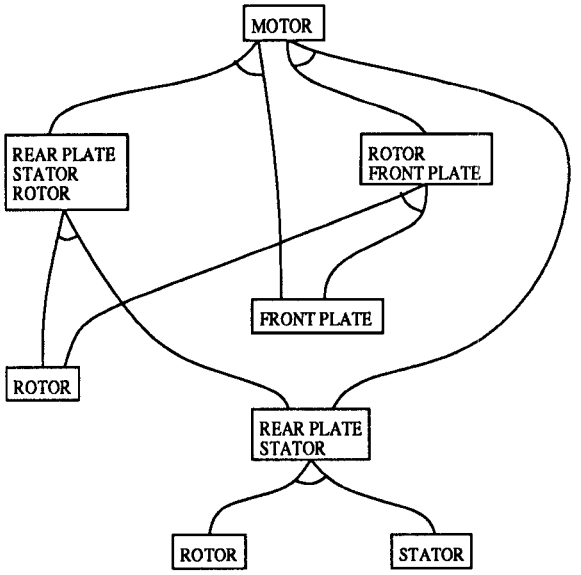

Figure 2: AND/OR Graph

Our model is a simple motor composed of four parts: rotor/bearings, front and rear plates, and stator.

To obtain the final assembly, a sequence of assembly steps must be devised. In order to do so, the product is conceptually disassembled until only elementary parts are left. This process, called generation of feasible assembly sequences, allows the definition of partial assembly states and the sequence of operations needed to move through the assembly state space.

Out of all the possible disassembly/assembly sequences, only some are feasible. Some subassembly states and some assembly operations are impossible because of geometrical and technological constraints, and have to be eliminated. To capture the possible assembly sequences, an AND/OR graph representation is used. Referring to the graph of Figure 2, K-connectors represent feasible decompositions while nodes represent legal subassembly states. The graph shown in the figure considers only a subset of all the geometrically feasible decompositions (some of them have already been discarded). For example, the rear plate could be separated from the whole assembly, giving place to a geometrically feasible decomposition. Nevertheless, the resulting inverse assembly operation would be complex to perform, and the motor without the rear plate would be an unstable subassembly, inpractical to handle and to operate on. Using the above criteria, the AND/OR graph of Figure 2 has been generated. It represents two possible sequences of technologically feasible operations and stable subassemblies.

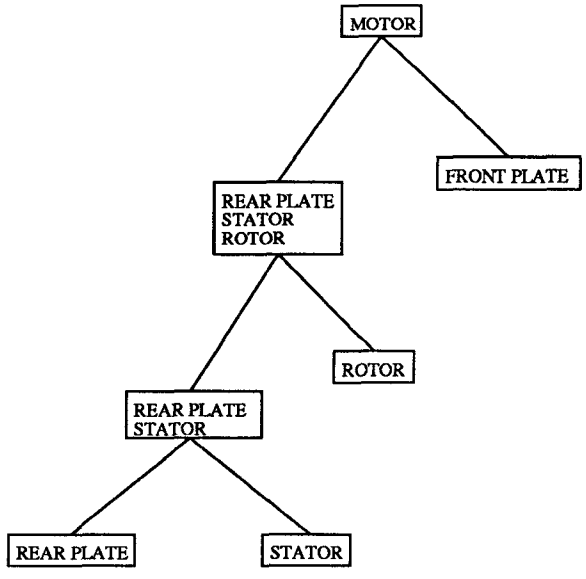

Figure 3: Plan 1 Assembly Tree

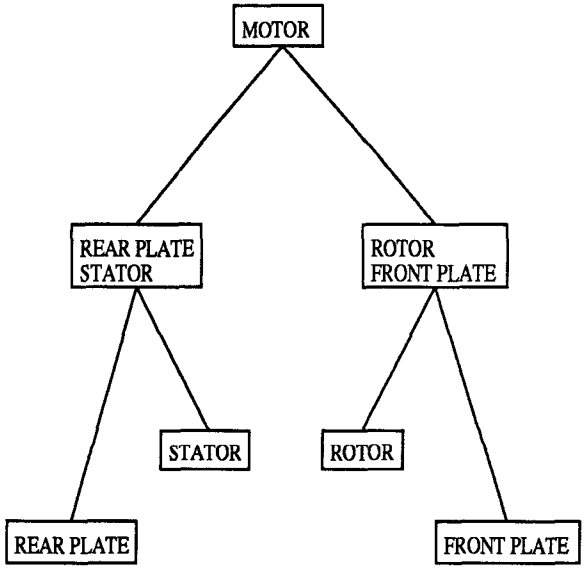

Figure 4: Plan 2 Assembly Tree

Table 1: Assembly Feeder Data

FEEDERS	ROLE	ASS. PART	PREP. TIME
Feeder 0	input	stator	5
Feeder 1	input	rear plate	5
Feeder 2	input	rotor	5
Feeder 3	input	front plate	5
Feeder 4	output	motor	10

From the AND/OR graph, two possible assembly trees are generated, as shown in Figures 3 and 4. These trees constitute the basis for the synthesis of two alternative production plans. Assembly trees embody the precedence relationships between operations. For example, the first tree leads to the implementation of a pipeline structure where single parts are added to a single subassembly. The second tree allows a parallel organization where two partial assemblies are joined to get the final product.

2.2 Production Plan, Requirements and Constraints

Once an assembly tree is generated, assembly operations have to be assigned to actual physical devices. In addition, materials have to be moved among devices. Thus to complete the system design, material handling and storage components are needed.

Due to the simplicity of the assembly plan, machines are clustered in a single workcell serviced by a single robot. For both plans the workcell needs four parts and produces a single finite assembly. Therefore four input and one output feeders must be provided. Input feeders have to be connected to the same number of storage areas in order to receive parts. The output feeder is connected to a final product storage area. Parts are moved using conveyors, therefore five of them are needed. This information is used during the SES pruning process to obtain the system composition tree.

To have a complete assembly plan, timing values for operations must be added. As shown in Table 1, each part is assigned to a feeder. Feeder preparation time is the time needed by an empty feeder receiving an object to make it available on output. Feeders are supposed to have a capacity of two. Machine processing time is determined by the operation's complexity. Machines have to wait the arrival of all needed parts before they can begin processing. In Tables 2 and 3, the list of parts in square brackets indicates that these parts have been connected into a subassembly

Table 2: Assembly Plan 1 Machine Data

MACHINES	PARTS NEEDED	SUBASS. PROD.	PROCESS. TIME
Machine 0	rear plate, stator	[rear plate, stator]	30
Machine 1	[rear plate, stator], rotor	[rear plate, stator, rotor]	20
Machine 2	[rear plate, stator, rotor], rear plate	motor	30

Table 3: Assembly Plan 2 Machine Data

MACHINES	PARTS NEEDED	SUBASS. PROD.	PROCESS. TIME
Machine 0	rear plate, stator	[rear plate, stator]	30
Machine 1	rotor, front plate	[rotor, front plate]	10
Machine 2	[rear plate, stator], [rotor, front plate]	motor	40

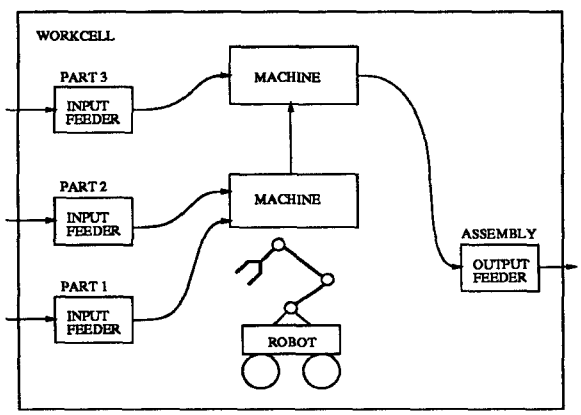

Figure 5: Workcell Physical Organization

Table 4: Assembly Plan 1 Topology

DEVICE	X-coord	Y-coord
Feeder 0	2	6
Feeder 1	2	2
Feeder 2	4	6
Feeder 3	6	6
Feeder 4	6	2
Machine 0	2	4
Machine 1	4	4
Machine 2	6	4

and are now a single product.

A workcell topology is important in order to define the robot's timing. Figure 5 shows a simple physical organization of one system under consideration. The robot must pick up parts 1, 2 and 3 from the respective feeders, one at the time, and put them into the machine needing them. When the machine is done processing, the robot must remove the assembly and put it into the machine downstream or into the output feeder.

Robot trajectories, defined by Task Level Programming, are determined by a very simple bidimensional model. Trajectories are straight lines between points, while the velocity profile during the trajectory is determined as follows: The robot accelerates with constant acceleration A_r up to the constant velocity V_r, then decelerates again with deceleration $-A_r$ until it stops. If the trajectory is not long enough to reach the velocity V_r, the robot accelerates up to the middle point of the path, and then decelerates. Numerical values have been set to $V_r = 0.5m/s$ and $A_r = 1m/s^2$.

Tables 4 and 5 contain device coordinates for plan 1 and 2.

2.3 Evaluation Objectives and Experimental Frames

The design objectives orient both the model and the experiment construction. In fact, from the design objectives, performance indexes and experimental conditions are derived. The first performance index is the time needed by the plant to produce a given number of assemblies. The second performance index is the utilization profile of workcell machines. The third performance index is the machines' joint utilization, defined as the percentage of time during which all machines are working at the same time. The fourth performance index is the utilization profile of the robot.

Table 5: Assembly Plan 2 Topology

DEVICE	X-coord	Y-coord
Feeder 0	2	6
Feeder 1	2	2
Feeder 2	4	6
Feeder 3	6	6
Feeder 4	4	2
Machine 0	2	4
Machine 1	6	4
Machine 2	4	4

From the above performance indexes, Experimental Frames are derived. For all of them, no input segments are provided. The storage areas are considered part of the system itself. Parts storage areas are loaded with the right quantity of parts needed, while the finite product storage areas are empty at the beginning of simulation runs.

2.4 Simulation

Simulation of design models is carried out in the DEVS-Scheme environment (Zeigler 1990). DEVS-Scheme is an object-oriented simulation shell for modeling and design that facilitates construction of families of models specified in the DEVS formalism. To specify modular discrete event models requires that we adopt a different view than that fostered by traditional simulation languages. As with modular specification in general, we must view a model as possessing input and output ports through which all interaction with the environment is mediated. In the discrete event case, events determine values appearing on such ports. More specifically, when external events, arising outside the model, are received on its input ports, the model description must determine how it responds to them. Also, internal events arising within the model, change its state, as well as manifest themselves as events on the output ports to be transmitted to other model components.

Basic models can be coupled to form a class of coupled models. This facilitates simulation of hierarchical, modular, multicomponent models (Zeigler 1990).

The synthesized model uses the following basic components: Production Store PSTORE, Plant Controller PLC, Workcell Feeder FEEDER, Workcell Assembly Machine MACHINE, Robot Model ROBOT, and Workcell Controller CONTR. In addition, the experimental frame uses a Workcell Transducer TRANWC, a Robot Transducer TRANROB, and a Production Store Transducer TRANPS. The Production Store Acceptor has been omitted. The simulation stops by itself when all parts have arrived at the Finite Product Store and all devices are idle.

The Production Store PSTORE can contain parts or finite products. In the first case, when requested, it waits for processing time units and then delivers a part (provided it is not empty). In the second case, when a request for storage arrives, PSTORE waits its processing time and then stores the assembly (provided it is not full). The delays incorporated into the Production Stores models simulate the presence of conveyors connecting them with the Workcell.

The simulation starts with Production Stores loaded with the required number of parts and the output Production Store set to empty. All workcell devices are off line. To begin production, the Cell Controller CONTR receives the production plan as input data. The production plan assigns parts to cell Feeders, designates them as input or output devices, determines the parts needed by the machines to produce an assembly, and defines the devices' timing.

The Cell Controller sets itself in a working status and broadcasts the plan to all cell devices. Each device scans the plan, extracts the information it needs in order to operate, and sends its new status to the Cell Controller. At this point the production phase begins.

Messages to and from the Production Stores are sent and received by the Plant Controller PLC that manages communications and material flow among the Production Stores and the Workcell. During the operations, the Workcell generates messages for the Plant Controller, sending lists of needed parts to be received from Part Production Stores and lists of Finite Products to be sent to the Finite Product Store. When the Plant Controller receives requests for assistance from the Workcell, it puts them in a waiting list and then broadcasts this list to all Production Stores. Production Stores which are able to satisfy the requests wait their processing time and then send a reply to the Plant Controller that routes it to the workcell. At the same time, the Plant Controller deletes from its waiting list the request that corresponds to the reply.

We now analyze the Workcell's operation. When the Plant Controller sends messages to place or pickup parts on/from the workcell, the Cell Controller routes them to the corresponding input/output Feeders. At any moment, the Cell Controller keeps track of the status of each Feeder, Machine and the Robot, and generates commands for the Robot and external messages for the Plant Controller, accordingly.

When parts from the Parts Production Stores ar-

rive at the Cell Feeders, Feeders communicate this to the Cell Controller. The type of part offered is also relayed. At the same time, the Cell Controller knows the status of all machines. From this information, the Cell Controller is able to generate a list of external requests for the Plant Controller and a list of possible tasks for the Robot. An external request is generated when an input feeder can receive parts and/or when an output feeder has parts to ship out. A task for the Robot is generated when a device (feeder or machine) has a part or subassembly available and some other device inside the workcell needs it. A Robot Task is generated only if the Robot is idle at that moment. If it is not, the Cell Controller waits until the Robot asks for one. If several tasks are possible, the Cell Controller chooses the task for which the Robot has to travel the shortest distance.

When the Robot receives a task, it moves from its current position to the device where it has to pick up a part, communicates to the device its action, then holding the part, moves to the second device, and communicates to the device that it is placing the part. When the Robot is done, it stops near the destination device and asks the Cell Controller for a new task.

The Workcell Transducer receives a signal every time a cell device changes its status and updates its time table and its internal clock. The Workcell Transducer timetable contains as many entries as the number of machines in the system, each entry has four partial time counters, one per each possible machine status. An additional time counter registers the joint utilization. The Robot Transducer receives a signal every time the robot communicates with the Cell Controller, picks up or places a part. It keeps track of the time the Robot is idle, moves empty, or moves holding a part.

At a global level, the Final Product Store Transducer, records the number of assemblies and the times of their arrival at the Final Product Store.

2.5 Simulation Results

For both plans the production of a batch of ten motors has been simulated. For Plan 1, the time needed to produce the motors was 1744.30 units, the utilization profile of workcell machines is shown in Table 6, the joint utilization of machines was zero. The Robot was idle for 13.59% of the time, moved empty for 48.29%, and moved holding a part for 38.12%. For Plan 2, the time was 1699.41 units, the utilization profile of workcell machines is shown in Table 7. The joint utilization of machines was zero. The Robot was idle for 16.63% of the time, moved empty for 43.27%, and moved holding a part for 40.10%.

Table 6: Plan 1 Machines Utilization [%]

MACHINE	s_a	s_b	s_c	s_d
M_0	27.45	36.87	17.20	18.48
M_1	40.98	27.85	11.47	19.70
M_2	27.63	38.62	17.20	16.55

Table 7: Plan 2 Machines Utilization [%]

MACHINE	s_a	s_b	s_c	s_d
M_0	33.88	12.65	17.65	35.82
M_1	48.12	31.93	5.88	14.07
M_2	35.71	31.63	23.54	9.12

Although the two designs have different architectures (pipeline vs. parallel), the results show that they have equivalent dynamic performances. Complementary design aspects, such as cost and availability of tools and fixtures, will guide the designer's final choice.

3 CONCLUSIONS

A comprehensive framework for design of automated manufacturing system requires integration of several layers of support methods and tools. We are developing an FMS CAD framework in which simulation plays a pivotal role. The need for simulation component in FMS CAD is becoming increasingly obvious. Most existing systems facilitate only one mode of operation, i.e., the off-line input of robot's program and subsequent testing of the program by graphic animation of robot's motions in a geometric model of the workscene. The systems are capable of detecting collisions. However, they do not facilitate simulation of the overall automation system in order to evaluate the proposed FMS design. Starting from manufacturing objectives, our methodology generates a technological process as a sequence of machining and/or assembly operations, and organizes it into elementary operations of robots and devices. These operations are translated into motion commands for robots. Different variants of motion interpretation are tested by a simulator. Simulation is used to select the most effective variant.

Our current work is focused on simulation based optimization of assembly systems layout. Given a set of topological costraints imposed on the system layout, devices are placed on the workscene in such a way that material handling costs are minimized.

REFERENCES

Brooks,R. 1983. Planning Collision-Free Motions for Pick- and-Place Operations. *Int. J. of Robotics Research* **2**(4): 19-44.

Homem De Mello, L.S., and A.C. Sanderson. 1990. AND/OR Graph Representation of Assembly Plans. *IEEE Trans. on Robotics and Automation*, **6**(2): 188-199.

Jacak, W. 1989. Strategies for Searching Collision-Free Manipulator Motions: Automata Theory Approach. *Robotica*, **7**: 129-138.

Jacak, W. 1989. A Discrete Kinematic Model of Robot in the Cartesian Space. *IEEE Trans. on Robotics and Automation*,**5**(4): 435-446.

Jacak, W., and J.W. Rozenblit. 1991. Automatic Simulation of a Robot Program for a Sequential Manufacturing Process, *Robotica* (in print).

Kusiak, A. 1990. *Intelligent Manufacturing Systems.* Prentice Hall.

Lozano-Perez, T. 1989. Task-Level Planning of Pick-and- Place Robot Motions. *IEEE Trans. on Computer* **38**(3): 21-29.

Rozenblit, J.W., B.P. Zeigler. 1990. Knowledge-Based Simulation Design Methodology: A Flexible Test Architecture Application. *Transactions of The Society for Computer Simulation* **7**(3): 195-228.

Sanderson, A.C., L.S. Homem De Mello, and H. Zhang. 1990. Assembly Sequence Planning. *AI Magazine*, **11**(1), Spring.

Shin, K., N. McKay. 1986. A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators. *IEEE Trans. on Automatic Control*, **31**(6): 491-500.

Zeigler, B.P. 1984. *Multifacetted Modelling and Discrete Event Simulation,* Academic Press.

Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models,* Academic Press.

AUTHOR BIOGRAPHIES

MARCO CHIEROTTI is a Graduate Student of Electrical and Computer Engineering at The University of Arizona, Tucson. He received the M.S. degree in Mechanical Engineering from the University of Genova, Italy, in 1985. His principal interests focus on modeling and computer simulation, knowledge-based system design, and artificial intelligence.

JERZY W. ROZENBLIT is an Assistant Professor of Electrical and Computer Engineering at The University of Arizona, Tucson. He received the Ph.D. and M.S. degrees in Computer Science from Wayne State University, Detroit, in 1985 and 1983, respectively, and the M.S. degree in Control Engineering from the Technical University of Wroclaw, Poland, in 1980. He specializes in modeling and computer simulation, knowledge-based system design, and artificial intelligence. His principal research activities focus on the development of expert, computer-aided environments for engineering design support.

WITOLD JACAK is currently Professor of Systems Science at Johannes Kepler University in Linz, Austria. He received the Ph.D. degree in control and system engineering from the Institute of Techical Cybernetics, Technical University of Wroclaw, in 1977, and the M.S.E.E. degree in electronics from the Technical University of Wroclaw, Poland, in 1973. His research interests include artificial intelligence and robotics, CAD/CAM systems, modeling, simulation, and system theory applications to robot motion planning.