Simulation-Based Planning of Robot Tasks in Flexible Manufacturing

Jerzy W. Rozenblit

Dept. of Electrical and Computer Engr.
The University of Arizona
Tucson, Arizona 85721, U.S.A.

Witold Jacak

Institute of Technical Cybernetics
Technical University of Wroclaw
50-370 Wroclaw, Poland

Abstract

A framework is proposed for support of design, task planning, and simulation of automated manufacturing systems. The framework establishes a hierarchy of method banks essential for improving the efficiency and cost effectiveness of manufacturing processes. The methods should support automatic generation of sequencing rules, design and configuration of the manufacturing facility and equipment, synthesis of task oriented robot programs, and generation and execution of simulation models of a manufacturing system. In this paper, each layer is addressed and preliminary results that apply simulation to interpret and test task oriented robot programs are discussed.

1 Introduction

Recently, the use of programmable automation and flexible automation has had significant impact on machining and assembly of products [1]. The economic importance of manufacturing and robotics has led to extensive efforts to improve the effectiveness of robot-based process automation. More systematic approaches to design and planning of robot tasks are still needed to further improve performance. Flexible manufacturing systems (FMS)—treated here as both machining and assembly systems—possess a number of unique features and characteristics which require that their design and operation strategies be substantially different from those used in conventional job shop and transfer line facilities [2].

A flexible manufacturing system is a set of machines connected by a flexible material handling facility (such as a robot, a crane, or an automated guided vehicle) and controlled by a computer [1]. Some of the distinguishing features of FMS are [1]:

- High degree of automation of machines and material handling systems.

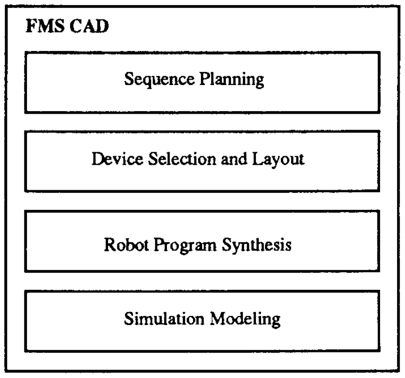

Figure 1: Architecture for FMS CAD

- The layout of machines is strongly influenced by the type of material handling system used and the structure of the manufacturing process.

- The design of the system influences its operation.

An integrated CAD framework for design of FMSs should address the above issues. In this paper, we stipulate an architecture for such a framework and discuss its basic layers shown in Figure 1.

We begin by summarizing assembly sequence planning. Then, a design approach for selection and configuration of manufacturing system's components is discussed. Next, synthesis of a task-oriented robot program is presented. The synthesis process is supported by discrete event simulation models of the robot's world and interpretation and planning techniques. The resulting amalgamation of simulation and planning facilitates studies of alternatives realizations of robot's motion and thus supports the overall performance evaluation of the manufacturing system. At the conclusion, we justify the need for this amalgamation.

2 Assembly Sequence Planning

Automatic synthesis of a program of robot's actions requires the integration of sensory and motor control information with an internal representation of parts, geometries, and relationships. We propose that the robot program synthesis be realized by a hierarchical system of task planning. The system consists of two basic layers, i.e., the task planning layer and task level-programming layer.

Task level planning is based on the description of the product and the operations of the machining process, a description of the system and its resources such as devices, robots, fixtures or sensors, and a specification of a precedence relation over the set of operations. The task-level plan describes the decomposition of the machining task into the sequence of elementary operations of robots and devices, the assignment of machining subtasks to system resources and a model for the coordination and scheduling of system resources [3, 4].

The basic problem at the task planning level is the derivation of an ordered sequence of robot's actions that can be used to perform a machining or an assembly task. To solve it, a graph representation can be used. Sanderson et.al. [4] define a framework in which AND/OR graphs represent all possible sequences of assembly plans. A decomposition of the machining task corresponds to a cut set of this graph. Feasible decompositions, with respect to precedence relation of assembly operations, are used to create an AND/OR graph that represents all valid operation sequences. The path with minimal deadlock cases is searched. To eliminate deadlock cases, the execution-preconditions for each operation are created. 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.

Detailed descriptions of assembly sequence planning based on the AND/OR graph representation are given by Homem de Mello, Sanderson, and Zhang [3, 4] who strongly advocate the integration of the assembly planning with task-level programming. We discuss simulation support of task programming in Section 4. Prior to that, we define a layer for support of selection and configuration of devices in a manufacturing facility.

3 System Design—Device Selection and Layout

Design of a manufacturing facility capable of carrying out an assembly sequence plan is an integral phase of the proposed framework [1]. 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. In this section, we examine how a general system design methodology can aid in this integration process.

The system design approach proposed by Rozenblit [5, 6] 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 that 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.

Design begins with developing a representation of system components and their variants. We have proposed a representation scheme called the *system entity structure* (SES) [6, 7] that captures the following three design relationships: decomposition, taxonomy, and coupling. Decomposition knowledge means that the structure has schemes for representing the manner in which an object is decomposed into components. Taxonomic knowledge is a representation for the kinds of variants that are possible for an object, i.e., how it can be categorized and subclassified. 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 [8]. They are used to manipulate the elements in the design domain by selecting and synthesizing the domain's components. This selection and synthesis process is called *pruning* [9]. Pruning results in a recommendation for a *model composition tree*, i.e., the set of hierarchically arranged entities corresponding to system's components. A composition tree is generated from the system entity structure by selecting a unique entity for specializations and a unique aspect for an entity with several decompositions.

The final step in the framework is the evaluation of alternative designs. This is accomplished by simulation of models derived from the composition trees. Discrete Event System Specification (DEVS) [7] is used as a modeling formalism used for system specification in the methodology. DEVS provides a formal representation of discrete event systems. (We define DEVS in Section 4.1)

In the what follows, we demonstrate how this methodology applies to the FMS CAD framework discussed here.

3.1 Device Selection and Configuration

The system entity is a structured representation of a flexible manufacturing facility. To configure a specific system's layout (i.e., collection of devices and their arrangement) for a given product and assembly plan, pruning is invoked. For illustration, consider the following problem (adopted from Rozenblit and Zeigler [10]): Flexible testing of printed circuit boards (PCB) involves several types of devices that may be configured in several ways depending on the type of board and test plan (here, our interpretation of a test plan is semantically equivalent to that of a machining plan). Figure 2 depicts a generic entity structure representation of the facility called Automatic Test Facility (ATF). The components of ATF include test cells, transport devices, production stores, and auxiliary facilities. A test cell can be an in-circuit tester or a functional tester. A transport device can be a conveyer, a crane, or an automatic guided vehicle (agv). A production store can be a post assembly dock, a scrap store, or a stock store. A burn-in, an inspection cell, and a repair workstation are auxiliary facilities.

Flexible testing consists in generating a configuration of components selected from the set of the ATF's entities for a specific type of PCB. For example, in order to test a bare-board (board with circuit connections fetched on it but without any attached devices), only an inspection facility that checks for shorts and opens may be sufficient. On the other hand, another type of board may be tested using all of the ATF's components.

Some fundamental criteria for selecting various kinds of components are given. In-circuit testers require access to all circuit nodes on the board and isolation of devices under test. A special fixture called bed-of-nails is also mandatory in order to perform an in-circuit test. In-circuit tests do not detect timing defects whereas the functional tests do. The latter tests require an edge connector. A burn-in auxiliary facility is used to operate a board dynamically at an elevated temperature. Such a facility is usually followed by in-

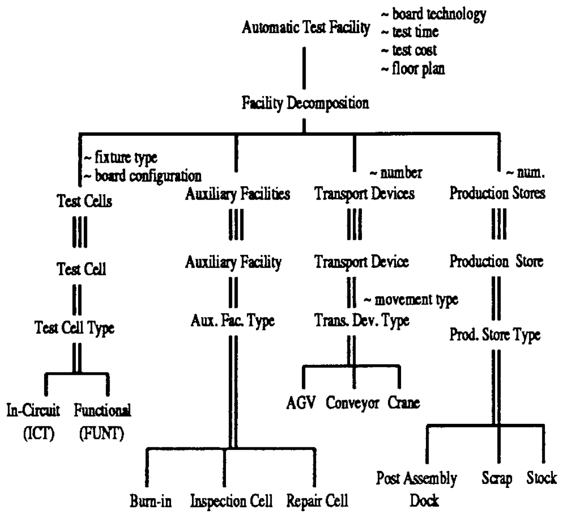

Figure 2: System Entity Structure of ATF

spection of the board.

Given the above description, a rule base is defined that is used to prune different configurations of the ATF. The problem is defined as follows: "Given a set of parameters, test design attributes, prune a set of workstations from which ATF will be composed."

Sample selection and synthesis rules used to generate alternative configurations of the ATF are given below. The complete knowledge base is presented in [10].

Example of Selection Rules

if access to all circuit nodes on the Unit Under Test (UUT) is available and
  devices under test can be isolated and
  bed-of-nails fixture available for the PCB and
  timing defects detection is not required
then select ICT tester

if access to all circuit nodes on the UUT is available and
  devices under test can be isolated and
  bed-of-nails fixture available for the PCB and
  timing defects detection is required and
  edge connector is available
then
  selected testers are: ICT and FUNCT

Example of Synthesis Rule

if selected testers are: ICT and FUNCT and
  there are no auxiliary facilities
then

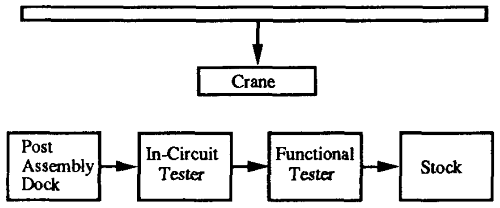

Figure 3: Sample Layout of ATF

testers are: ICT cascaded with FUNCT and
there are no Auxiliary Facilities and
the transport system is Crane
there is a Post Assembly Dock production store
there is a Stock production store

Pruning in our design methodology is supported by an expert system shell [10]. A sample result for the ATF model is depicted in Figure 3. This model was generated in a consultation session in which no dynamical testing at elevated temperature was necessary nor was the on-site repair critical to the test procedure. Given an assembly plan and manufacturing system to carry it out, the program of robot's actions must now be synthesized.

4 Robot Program Synthesis and Simulation

The action plan for a manufacturing process determines the robot's program of actions in servicing the process. The program is usually a sequence of instructions expressed in a task-oriented robot programming language (TORPL) [11, 12]. The implementation of the task-level plan is carried out using a task-level programming approach in which detailed paths and trajectories, gross and fine motion, grasping and sensing is specified. Variant interpretations of the instructions result in different realizations of robot actions. To create and verify all valid interpretations of motion program a two-level system has been proposed [13]. The first level is the motion planner for each individual robot action. It creates variants of collision-free time-trajectories of a manipulator that are used to perform the individual robot action. Such a planner uses robot-dependent planning techniques and discrete system formalism (DEVS). The second level is the discrete event simulator of the entire machining process. It models robots and devices. The variants of motion interpretation obtained from the motion planner are tested by the simulator. The simulation carried out

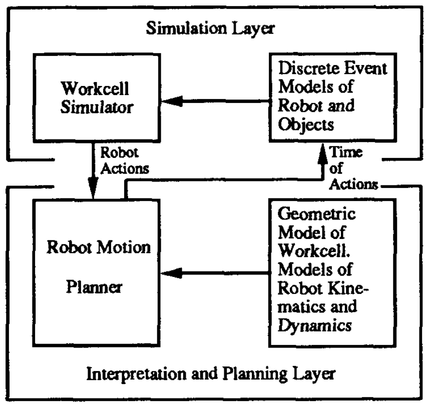
Simulation Layer
Workcell Simulator
Discrete Event Models of Robot and Objects
Robot Actions
Time of Actions
Robot Motion Planner
Geometric Model of Workcell. Models of Robot Kinematics and Dynamics
Interpretation and Planning Layer

Figure 4: Structure of Robot Task Simulation System

at the next level is used to select the most effective variant for realizing the robot's program. The layers for program synthesis and simulation are:

- Layer 1: The simulation layer that comprises: a) a simulator of the technological process, b) a simulator of the robot based on a discrete event model of the manufacturing process.

- Layer 2: The interpretation and planning layer for each individual robot action. A geometric model of the workcells is employed for plan generation.

The simulation layer (Layer 1) automatically synthesizes a model of the technological process by analyzing the technological operations applied to details. This results in a discrete event model specification which serves as the basis for simulating the robot's actions. To simulate, the system must have the knowledge of how individual actions are carried out in the process. This knowledge must be available in order to schedule the correct sequence of actions. (Recall that each action has an associated set of commands of the robot programming language.)

Layer 2 supports scheduling. It interprets and simulates the command based on a geometric model of the robot and its workscene. The planning component of this layer automatically formulates the robot's motion trajectory. It also provides time parameters for the robot's actions. These parameters are used in the simulation carried out at Layer 1. The structure of the proposed system is illustrated in Figure 4.

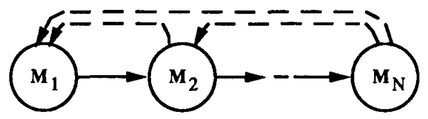

Figure 5: Graph Structure of Manufacturing Process

4.1 Basic Formal Concepts for Robot and Manufacturing System Modeling

Currently, our analysis is restricted to sequential technological processes. Each machine (workcell) is assigned an operation to process a detail. An assembly line consists of a series of workcells $M_i | i = 1, \ldots, N$. In addition, there are two special workcells, the feeder conveyer M_F and the output conveyer M_O. These conveyers can serve as input/output devices for other assembly lines. (For the sake of brevity, specifications of the feeder and output conveyers are omitted in this paper.) Each workcell has its own program for processing a detail. The program determines the time necessary to carry out the operation assigned to a workcell. The operation is executed by a robot. The robot also transports a detail among the workcells. We do not provide any facilities for queuing details at the workcells. A diagram of the system under consideration is shown in Figure 5.

Each vertex in the figure corresponds to a technological device (workcell). The arcs represent possible robot movements between workcells. An arc depicted by a continuous line symbolizes the transport of a detail between workcells. A dashed line represents a possible "empty" move of the robot servicing the line. An empty move is necessary to transfer the robot to a workcell requesting service.

To model the assembly line we employ the Discrete Event System Specification (DEVS) formalism [7].

In DEVS one must specify basic models from which larger ones are built, and how these models are connected together in a hierarchical fashion. A basic model, called an atomic DEVS is defined by the following structure:

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

where

X is a set, the external input event types
S is a set, the sequential states
Y is a set, the external output event types
δ_{int} is a function, the internal transition specification
δ_{ext} is a function, the external transition specification
λ is a function, the output function
t_a is a function, the time advance function

with the following constraints:

(a) The total state set of the system specified by M is
$$Q = \{(s,e) | s \in S, 0 \leq e \leq t_a(s)\};$$
(b) δ_{int} is a mapping from S to S:
$$\delta_{int} : S \rightarrow S;$$
(c) δ_{ext} is a function:
$$\delta_{ext} : Q \times X \rightarrow S;$$
(d) t_a is a mapping from S to the non-negative reals with infinity:
$$t_a : S \rightarrow R,$$
(e) λ is a mapping from S to Y:
$$\lambda : S \rightarrow Y.$$

An interpretation of the DEVS and full explication of its semantics are in [7]. The second form of models, called a coupled model, tells how to couple several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to the hierarchical model construction.

4.2 Simulator Design

The robot's actions in a system depicted in Figure 5 can be realized by elementary operations. Each elementary operation has an associated set of instructions in the task-oriented robot programming language (TORPL). The basic macro-instructions of TORPL are: MOVE (EMPTY, HOLDING) TO position, PICKUP part AT position, PLACE part ON position, WAIT FOR sensor input signal, START output signal. Basic instructions can be combined into a higher level macros such as the PICK-AND-PLACE operation [14, 15]:

```
MOVE EMPTY TO position A
PICKUP part AT position A
MOVE HOLDING part TO position B
PLACE part ON position B
```

The above instructions are used to synthesize the robot's action program. The synthesis process requires introduction of conditional instructions that depend on the states of each device M_i of the assembly line. Thus, to define a simulator of the program, we model conditions that enable program instructions. Each device M_i has the following DEVS representation:

$$M_i = (X_i, S_i, Q_i, \delta^i_{int}, \delta^i_{ext}, ta_i) \ \mid \ i = 1, \ldots, N$$

The state set of each M_i is defined as $S_i = \{A, B, C\}$ where

- A signifies that DEVICE IS NOT WORKING AND IS FREE
- B signifies that DEVICE IS NOT WORKING AND IS NOT FREE
- C signifies that DEVICE IS WORKING AND IS NOT FREE

Assume that the i-th position denotes the location of a machine at which a detail is placed. The set of external events for M_i is defined by the commands of the TORPL, namely:

$$\{X_i = x1_i, x2_i, x0 \mid i = 1, \ldots, N\}$$

where:
$x1_i$ = PLACE part on i-th position
$x2_i$ = PICKUP part at i-th position
$x0$ = DO NOTHING

The internal transition function for each device i is given as follows:
$\delta^i_{int}(\mathrm{A}) = \mathrm{A}$
$\delta^i_{int}(\mathrm{B}) = \mathrm{B}$
$\delta^i_{int}(\mathrm{C}) = \mathrm{B}$

The external transition function for each device i is defined as:
$\delta^i_{ext}((A, e), x1_i) = C$
$\delta^i_{ext}((B, e), x2_i) = A$
$\delta^i_{ext}((s, e), x0) = s$
$\delta^i_{ext}((.,.),.) = failure$ for all other states

The time advance functions for M_i determine the time needed to process a detail in the i-th device. They are defined as follows: if $s = C$, then $ta^i(s) = \tau_i$ (the tooling/assembly time for device i), otherwise $ta^i(s) = 0$.

The above specification defines a model of technological devices of the assembly line. The activation of each device M_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 [7] associated with the assembly line model. Since the events generated by the robot depend on the states of the workcells $M_i | i = 1, \ldots, N$, we define an acceptor which observes the state of each workcell. The block diagram of the entire simulation system is given in Figure 6.

Rather than provide a detailed mathematical description of the experimental frame models here, we

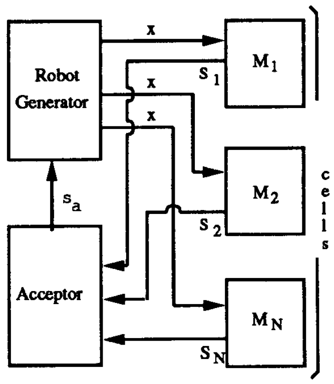

Figure 6: Discrete Event Simulator of Manufacturing System

describe their functionality. (The reader is referred to [13] for a complete formal specification of the simulator of Figure 6.)

The acceptor is a DEVS that receives as input state descriptions of each device M_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 M_i. The state contains indexes of only those workcells which have completed processing of a detail and from which the detail can be transported to another workcell (i.e., there exists a free workcell). The states of the acceptor also determine state components of the frame generator that models the robot.

The robot's model 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}. The robot's state set is partitioned into two subsets. The first subset corresponds to the EMPTY state of the effector-end, the second to the HOLDING state. The internal transition functions are represented by the following sequences in TORPL:

```
MOVE EMPTY FROM k-position TO i-position
PICKUP part AT i-position
```

for the EMPTY state of the effector-end, and

```
MOVE HOLDING part FROM k-position
                  TO k+1 position
PLACE part on k+1 position
START signal for device k+1
```

for the HOLDING state of the effector-end.

The time advance functions determine: a) the sum of the time of motion from the position k to i and the time of the pickup operation, b) the sum of motion time from the position k to the position k+1 and the time of the place operation on the k+1-th device, for EMPTY and HOLDING states, respectively.

The robot's model also generates external events (i.e., PICKUP, PLACE and START) for devices M_i, which trigger their corresponding simulators. .

The model we have proposed facilitates convenient generation of the robot's program. In order to construct the program, we translate the generator's transition function into commands of the task-oriented robot programming language: [13].

```
LOOP FOREVER (i= 1,..., N)
     IF ((i-th input signal EQ
          not working and not free) AND
         (i+1-th input signal EQ
          not working and free))
     THEN
        MOVE EMPTY TO i-th position
        PICKUP part AT i-th position
        MOVE HOLDING part TO i+1-th  position
        PLACE part ON i+1-th position
        START i+1-th device
END LOOP
```

The proposed model of a workcell is used as the basis for testing the program with a varying range of motion parameters. The most important parameters are the time it takes to complete an operation, τ_i, and the time the robot takes to service a workcell. The time τ_i depends on the type of device on which the i-th operation is being processed. It is fixed but can be changed by replacing the device. Similarly, the times of PICK UP and PLACE operations are determined by the type of detail and device on which the detail is processed.

The times of robot's inter-operational moves (transfers), $\tau_{i,j}$, depend on the geometry of the workscene and the cost function of the robot motion. This cost function determines the dynamics of motion along the geometric trajectories.

The planner's fundamental function is to synthesize the robot's motion trajectories. The trajectories realize the MOVE instructions of the program. They also determine the duration of the moves. These data must be accessible in order to simulate the entire assembly system. To generate the trajectories, Layer 2 of Figure 4 must have models of the robot and assembly line available. It must also have initial and final positions (coordinates) of each move given. The motion planning aspect of our approach is described in detail in [13, 16, 17].

5 Summary and Conclusions

A comprehensive framework for design of an automated (e.g., manufacturing, diagnostic, testing, etc.) system will require integration of several layers of support methods and tools. We have proposed an architecture that should facilitate:

- automatic generation of different plans of sequencing operations (operations scheduling problem)
- selection of devices (machines, material handling systems, and robots) to carry out the operations
- synthesis of programs for robots servicing the devices
- planning and interpretation of robots' motion programs in the generated geometrical model of workscene with respect to various criteria
- synthesis of the autonomous robotic system's simulation
- testing and verification of design variants based on the interpreted programs of robots' actions and simulation modeling of the overall system architecture

The integration of all the above features is a complex task, with each of the functions being a research topic in itself. However, the need for simulation component in FMS CAD is becoming increasingly obvious due to a number of reasons: 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 cannot plan collision free motion. They do not facilitate simulation of a workcell in order to evaluate its efficiency. They cannot emulate a programming language that would actively use a simulation model. Such languages do not exist yet. Our future work will focus on the automatic generation of such a language.

We shall also develop methods to integrate the assembly planning with device selection and layout. At this point, the pruning rule base contains constrains imposed by the assembly sequence, which imply a certain topology of the manufacturing system. A methodology for mapping assembly sequence plans into the pruning knowledge base will be established.

Acknowledgments

The work of Jerzy Rozenblit has been supported in part by Siemens Corporation, Corporate Research Division, Princeton, New Jersey. Witold Jacak was supported in part by the Polish Academy of Sciences. Ray Huang's expertise of LaTeXwas invaluable.

References

[1] A. Kusiak *Intelligent Manufacturing Systems.* Prentice Hall. 1990

[2] J.E. Lenz. *Flexible Manufacturing.* Marcel Dekker, Inc., 1989

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

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

[5] J.W. Rozenblit. A Conceptual Basis for Integrated, Model-Based System Design, Ph.D. Dissertation, Department of Computer Science, Wayne State University, Detroit, Michigan, 1985

[6] J.W. Rozenblit and B.P. Zeigler. Design and Modelling Concepts, in: *International Encyclopedia of Robotics, Applications and Automation,* (ed. Dorf, R.) John Wiley and Sons, New York, 308-322, 1988

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

[8] N.J. Nilsson. *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA. 1980

[9] J.W. Rozenblit and Y. Huang. Constraint-Driven Generation of Model Structures. *Proc of 1987 Winter Simulation Conf.,* Atlanta, GA. 604-611, 1987

[10] J.W. Rozenblit and B.P. Zeigler. Knowledge-Based Simulation Design Methodology: A Flexible Test Architecture Example. *SCS Transactions*, **7**(3), 195-228, 1990

[11] B. Farerjon. Object Level Programming of Industrial Robots. *IEEE Int. Conf. on Robotics and Automation,* **2**, 1406-1411. 1986

[12] R. Speed. Off-line Programming of Industrial Robots. *Proc. of ISIR 87*, 2110-2123, 1987.

[13] W. Jacak and J.W. Rozenblit. Automatic Simulation, Interpretation and Testing of a Task-Oriented Robot Program for a Sequential Technological Process, ***Robotica*** (to appear) 1991.

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

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

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

[17] W. Jacak. Modeling and Simulation of Robot Motions. *Lecture Notes in Computer Science*, **410**, 751-758, Springer Verlag, 1990.