Automatic Robot Programming by CAST

Witold Jacak

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

Jerzy W. Rozenblit

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

1 Introduction

In recent years, programmable and flexible automation 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 robot task planning systems. Such systems enable an automatic synthesis of programs which control robot actions. Many of the domain-independent approaches [6] ,[8],[11] to the automatic generation of robot action sequences do not map well into the general machining, assembly, or fabrication problem.

The robot action sequence generation is only one phase in the hierarchy of steps required to plan the robot's behavior in programmable automation systems. More systematic approaches to the design and planning of actions are needed in order to make the robot plan generation applicable to practical problems, to enhance their performance, and to enable their cost-effective implementation. At the implementation level, the action plan generation system should also be capable of reasoning about the geometry and time of actions.

In this paper we show that a stratified methodology can be applied to solve a problem of automatic robot programming. This can be accomplished since robot actions can be modelled in terms of different conceptual frameworks, namely, in terms of operational, geometric, kinematic, and dynamic frameworks.

In general a *task* in a flexible production system is defined as a partially ordered set of technological (machining or assembly) operations. To generate a program of actions for a task, we have to find sensor-dependent time trajectories of the robot's motions. These trajectories must realize all the operations defined for the task.

This problem may have several possible solutions. They depend on the technological operations and their order, sensor-dependent actions, geometric forms of the manipulator's paths, and the dynamics of movements along the paths. Thus it is necessary to apply a hierarchical decomposition of the manufacturing task at hand into subproblems. We then solve the program synthesis problem at several levels the system's behavior modeling abstraction. The system for automatic generation of robot programs consists of two basic layers: a) *Task Planning Layer*, and b) *Task-Level Programming Layer.*

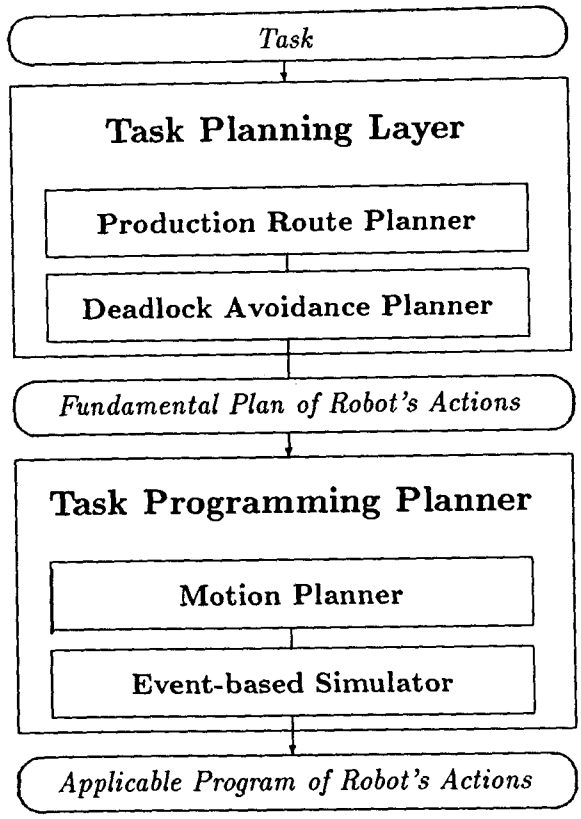

Fig.1 Structure of Robot's Program Synthesis System

Task-Level Planning is carried out based on a description of the operations of the underlying manufacturing process, a description of the manufacturing system and its resources such as devices, robots, fixtures, or sensors, and a description of the precedence relation over the set of operations. The planner uses robot-independent planning techniques. The resulting fundamental plan describes the decomposition of the task into a *sequence of elementary operations of robots and devices* called *Actions*. The basic problem here is the derivation of an ordered sequence of robot actions that can be used to perform the task. To solve it, two subproblems are defined. The first subproblem consists in finding an ordered, feasible sequence of technological operations, called a *production route* which can be transformed directly into the sequence of robot actions. In the second subproblem we need to find the set of preconditions for each action, which guarantee that deadlock does not occur once the actions are executed. The fundamental action plan for a manufacturing task determines the robot's program of manipulations required to carry out this task. Such a program is a sequence of motion, grasp, and sensors instructions expressed in the *Task-Oriented Robot Programming Language* (TORPL).

The implementation of the fundamental plan is carried out using a *task-level programming* approach in which detailed paths and trajectories, gross and fine motion, grasping and sensing instructions are specified. Variant interpretations of the plan's TORPL-instructions result in different realizations of the robot actions. To create and verify all valid interpretations of the motion program, a two-level system is needed. The first level is the *Motion Planner* [16], [12], for each individual robot action. The planner creates

variants of collision-free time-trajectories of the manipulator which executes each action. Such a planner uses robot-dependent planning techniques and the discrete dynamical system formalism [21]. The second level is the *Discrete Event Simulator* of the manufacturing process. The simulator uses the Discrete Event System Specification (DEVS) [9] [10] formalism to model actions and technological devices. The motion interpretations variants obtained from the Motion Planner form an experimental frame [9] of the DEVS simulator. The simulation carried out at the second level is used to select the most effective variant for realizing the robot's program steps. The structure of the multilayer task planning system is shown in Fig.1. In this paper we introduce basic research issues associated with each level of the planner and discuss how different Computer Aided System Theory tools and methods can address these issues. The paper is organized as follows: Section 2 gives a detailed description and formulation of the automatic robot programming problem. Section 3 presents the first layer of the task planning system. The second layer is described in Section 4. The robot task planning problem is presented next.

2 Formulation of Robot Program Generation Problem

A flexible manufacturing system (FMS) is a set of programmable machines (technological devices) D and product stores (buffers) M 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 [5],[6].

An FMS performs technological operations, e.g., fabrication, machining, or assembly operations. All machines and material handling systems (robots) have a high degree of automation. The current state of an FMS is monitored by the sensory system (*Sensor*) that collects and provides aggregate information about the state of the entire system, i.e.,

$$State = Sensor(FMS\ state) \tag{1}$$

The $FMS - state$ depends on the states of every machine and store. The state set of each machine $d_i \in D$ is defined as $S_i = \{s_a, s_b, s_c\}$ where

- s_a signifies that **machine is free**
- s_b signifies that **machine has completed an operation and is not free**
- s_c signifies that **machine is busy processing an operation**

A task realized by an FMS is represented by a pair

$$Task = (O\ ,\ \prec) \tag{2}$$

where O is a finite set of technological operations necessary to process the task and $\prec \subset O \times O$ is the weak-ordering precedence relation in which $o_i \prec o_j$ denotes that the operation o_i *must precede* the operation o_j.

In this paper, the analysis is restricted to a class of tasks which transform only one type of a part. Such tasks are called *single-batch pipeline machining processes.* Each machining operation o_i from the set O is assigned to a technological device (machine) d_i which can execute it and to a product store m_i where a part can be stored after the operation has been completed. Each machine has its own program for processing a part. To automatically generate the robot's program, we must determine for each robot r servicing the process, a set of the FMS-state-dependent time trajectories of the robot's motions:

$$T_r = \{q_r(x) : \tau_x \rightarrow Q_r \mid x = Sensor(FMS\ state)\} \tag{3}$$

These trajectories realize the transfers of parts between machines. For each part, they ensure that all operations from $Task$ are executed. The trajectory $q_r(x)$ is a function mapping the time interval τ_x into the joint space Q_r of the robot r. In addition, the planned sequence of robot actions should minimize the makespan [6].

We propose that the robot program synthesis be realized by a multilayer, knowledge-based robot task planning system. The architecture of this system is described in the ensuing section.

3 Machining Task Planning Layer

The first phase of the robot's program synthesis, namely the phase of the *fundamental plan* generation is performed by the task planner. The basic problem at this level is the derivation of an ordered sequence of robot actions that can be used to perform a machining task. The set of fundamental actions that a robot can execute is defined. These actions are represented by production rules [8],[11],[12] whose condition and conclusion parts represent the precondition and add lists, respectively. The fundamental action or a robot are:

$Action_r$:= TRANSFER FROM a TO b - transfer part from machine or store a to machine or store b, and

$Action_r$:= EXECUTE o ON b - start execution of operation o on machine b

In order to generate a plan of a task realization, actions have to be applied to affect changes in the FMS states. An action can be applied to a sensor-monitored state of an FMS if the preconditions are met. To establish the sequence of robot actions and their precondition lists, we decompose the task planning problem into two subproblems: a) the technological process planning problem (called production route planning problem), and b) the deadlock avoidance planning problem, solved on two respective levels of the task planning layer.

3.1 Production Route Planning Level

The production route planning is based on finding an ordered sequence of technological operations from $Task$ with a minimum number of deadlock instances. This corresponds to the operations scheduling problem [5].

An ordered sequence of operations, called *a pipeline sequential machining process*, is given by the following expression:

$$Process = (o_1, o_2, ..., o_L) \tag{4}$$

where:

(A) if for any two operations o_i, o_j from $Task$ $o_i \prec o_j$ holds, then $i < j$

(B) for each $i = 1, ..., L-1$ there should exist a robot which could transfer a part from machine d_i or store m_i (assigned to the operation o_i) to machine d_{i+1} (assigned to the operation o_{i+1})

(C) if two operations o_i and o_k are performed on the same machine, then $|i-k| = 1$ or $|i-k| \rightarrow \max$.

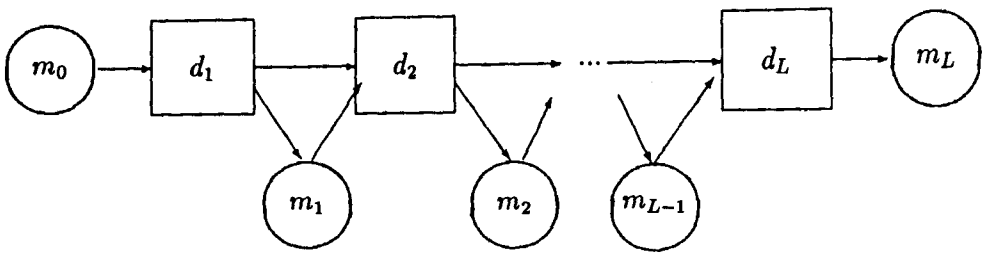

Fig.2 Production route in machining system

The technological process planning is based on the description of the machining operations, the description of the FMS's geometry (i.e., relationships among robots servicing machines and/or stores), the description of resources, and the precedence relation over the set of operations. To solve the process planning problem, a graph representation of *Task* can be used [7]. Feasible decompositions of the machining task with respect to the precedence relation are used to create an AND/OR graph that represents all valid operation sequences. The scheduling problem is solved using a backtracking graph search method with a heuristic penalty function.

Based on the sequence of operations *Process* that results from the machining process planning, we create the fundamental plan of robot actions as follows:

$$Plan = (Action_i^r | \ i = 1, \ldots, L) \tag{5}$$

where:

$$Action_i^r = (\text{TRANSFER FROM } d_i \text{ or } m_i \text{ TO } d_{i+1}, \text{EXECUTE } o_{i+1} \text{ ON } d_{i+1})$$

In addition, we define a set of ordered sequences of technological devices and stores (called resources) required by successive operations from the list $Process = (o_j | j = 1, ..., L)$. This set of new sequences called *production routes* is denoted by $\mathbf{P}$. Each production route $\mathbf{p} \in \mathbf{P}$ is an ordered list of resources and has $2L + 1$ stages, where L denotes the length of the list *Process*. A production route $\mathbf{p}$ is created on-line during the execution of the operations. In general, a route is defined as follows:

$$\mathbf{p} = (p(i) | i = 0, 1, \ldots, 2L) \tag{6}$$

where:

(a) $p(0) = m_0$ where m_0 denotes á feeder conveyer,

(b) for $i = 2j - 1$ and $j = 1, \ldots, L$, $p(i) = d_j$

(c) for $i = 2j$ and $j = 1, \ldots, L$, $p(i) = 0$ if direct transfer to machine d_{j+1} is possible or $p(i) = m_j$ otherwise.

Such a production route has always $2L + 1$ elements. Some of them can be equal to zero. The "minimal" production route has $L + 1$ stages nonequal to zero. The "maximal" route has $2L + 1$ stages where $p(i) = d_i$ for odd i and $p(i) = m_i$ for even i. The production route is illustrated in Fig. 2. The maximal route is denoted by $\mathbf{p}_{max}$.

In the production system considered in this paper, the so called *circular wait* deadlock of pipeline processes can occur. Circular wait occurs if there is a closed chain of task realizations, called *jobs* in which each job is waiting for a machine held by the next job in the chain [2]. To eliminate deadlock, the execution preconditions for each robot action

are determined. This issue is addressed on the second level of the task planning layer, which we call *deadlock avoidance planning level.*

3.2 Deadlock Avoidance Planning Level

Each execution of *Process* is called a job and is characterized by a production route p. When a production route includes multiple uses of resources, jobs executing a single route could become deadlocked. To avoid deadlock, the maximal production route $\mathbf{p}_{max}$ is partitioned into sublists called *zones.* A unique set of Z zones is defined as follows:

$$\mathbf{p}_{max} = (z_k | k = 1, ..., Z) \tag{7}$$

where: $z_k = s_k u_k$ and u_k is the sublist of resources which appear only once in the production route $\mathbf{p}_{max}$ and s_k is a sublist of resources which are used more than once in this route. Given this decomposition, we specify preconditions for each action based on theorems presented in [2],[3],[4]. These theorems determine the policy of required device allocation to the current job, dependent on the states of the resources in each zone.

The preconditions of $Action_i$ are formulated as a boolean function of sensor signals describing the states of the machines and production stores [5],[6]., i.e.,

$$\text{ConAct}_i \ : \ Sensor(FMS\ states) \rightarrow \{0, 1\}$$

If the preconditions are satisfied by the current state of the FMS, then the parameters of $Action_i$ can be established. The parameters of $Action_i$, represented by the function Param_i, describe the objects between which action TRANSFER has to be performed, i.e.,

$$\text{Param}_i \ : \ Sensor(FMS\ states) \rightarrow \{(d_i, d_{i+1}), (d_i, m_i), (m_i, d_{i+1})\}$$

This completes the definition of the fundamental plan of robot actions. Each action $Action_i$ from the *Plan* has the structure shown in Fig 3.

Given the sequence of *Actions* we should determine the program for robot operations and the geometrical trajectories of robot movements for each operation.

4 Task-level Programming Layer

The elementary actions of a robot can be expressed as instructions of a robot programming language, called *Task-Oriented Robot Programming Language* (TORPL). The instructions can be interpreted in many different ways. The basic macro-instructions of TROPL are : MOVE (EMPTY, HOLDING) TO *position*, GRASP, PICKUP AT *position*, PLACE ON *position*, WAIT FOR sensor input signal, INITIALIZE output signal, OPEN GRIPPER, CLOSE GRIPPER [13],[14],[16]. The basic instructions can be combined into higher level macros such as the PICK-AND-PLACE instruction [15],[16].

In this set of instructions, the action TRANSFER FROM a TO b can be interpreted as the PICK-AND-PLACE FROM x TO y macro-instruction, where x denotes geometrical data of the output port of the device (store) a, and y is the position and orientation of the input port of the device b. This macro-instruction is decomposed into a sequence of more primitive instructions as illustrated below [14],[15],[16]:

```
TRANSFER FROM a TO b  :=
   [begin PICK-AND-PLACE]
      MOVE EMPTY TO x+v  (x=position of output port of a
                          and v is approach vector)
            PICKUP AT x   :=
            [begin PICKUP]
```

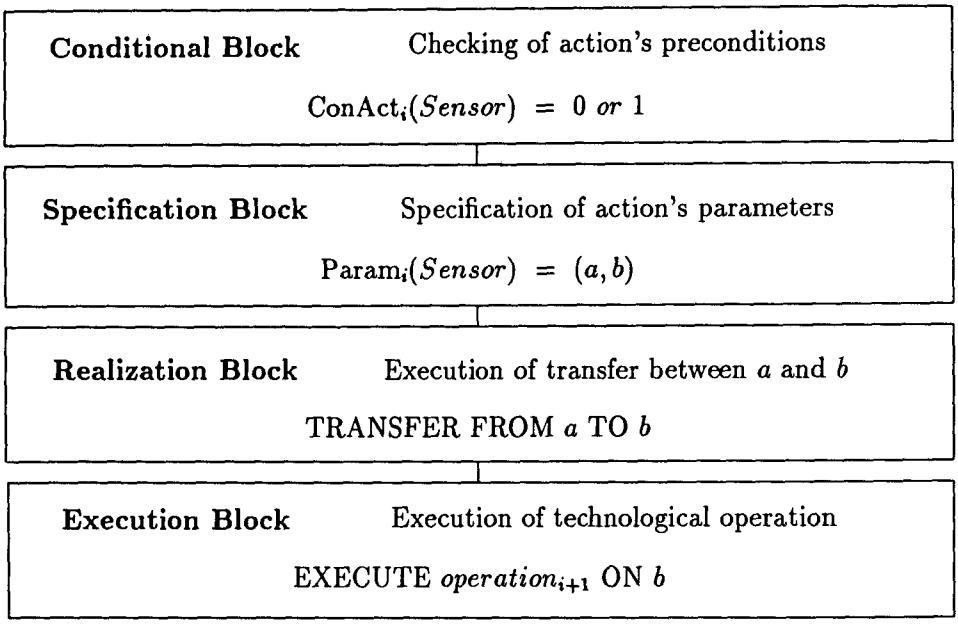

Fig.3 Structure of *Action*$_i$ from *Fundamental Plan*

```
        CENTER GRIPPER grasp orientation of effector
        OPEN GRIPPER
        MOVE EMPTY FROM x+v TO x
            WITH APPROACH = v
            CLOSE GRIPPER
        WAIT FOR contact signal with part
        MOVE HOLDING FROM x TO x+v WITH DEPARTURE = v
    [end PICKUP]

    MOVE HOLDING FROM x+v TO y+v (y=position of input
                                  port of b)
    PLACE ON y
[end PICK-AND-PLACE]
```

The positions x and y are determined by the function $Param_i(Sensor)$ with a specification of geometric parameters for each action. The above instructions are used to synthesize the robot's program. The instructions for the action EXECUTE can be translated into TORPL in a similar manner.

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

4.1 Level of Collision-free Motion Planning

The first level in our framework employs only the kinematic model of the manipulator. The planner should be able to determine the collision-free track of robot motion from the initial to the final effector locations based on a) the 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 [16],[17],[22]. The methods which solve the problem in question depend on the assumed mathematical model of the robot's kinematics. One possible description of the manipulator's kinematics is a discrete dynamical system [18],[19],[21],[20].

$$R = (U, C, \lambda) \tag{8}$$

where:

U is the input signal set. In order to specify the set U the discretization of the robot's Joint Space [22] is performed. In this case the input signal set U is equal to B^n where $B = \{-1, 0, 1\}$ and +1(-1) denotes a one-increment increase (decrease) of an appropriate joint angle [18],[21].

C is the set of robot's configurations (manipulator states) in the Cartesian space, i.e., configuration $c = (p_i | i = 1, \ldots n+1)$ where p_i is the point in the Cartesian base-frame describing the actual position of the i-th joint.

$\lambda : C \times U \rightarrow C$ is the one step state transition function [18],[21].

The construction of the transition function λ of the robot's kinematics model allows us to obtain, by simple computations, successive configurations of the robot with respect to the base frame.

The problem of collision-free robot movement planning amounts to finding a sequence $u_K^* = (u^1, .., u^K)$ of input signals such that the terminal configuration reaches the effector's final position, i.e.,

$$\lambda^*(c_0, u_K^*) = c_K \; and \; p_{n+1}^K = final \; position \; of \; effector$$

and every configuration c_i for i=1,..,K must not collide with any obstacle in the robot's environment.

In order to solve the terminal configuration reachability problem, we apply a graph search procedure to the state-transition graph of the model R. The details concerning the path and grasp planner are given in [16],[22].

From the path planner, we obtain an ordered sequence $c^* = (c_0, .., c_K)$ of the configurations which define how to move the effector end from the initial into the final position. The sequence c^* can be easily transformed into a sequence $q^* = (q_0, .., q_K)$ of the manipulator states described in the Joint Space [20]. Next, a geometric track can be constructed by connecting the configurations from q^*. This can be accomplished by using, for example, cubic splines [23]. Finally, the geometric track can be given in the form of a parameterized curve

$$q : [s_0, s_{max}] \rightarrow Q$$

where the initial and final configurations of the track correspond to the points $s = s_0$ and $s = s_{max}$, respectively.

Now, the optimal speed and acceleration of movements along the computed track should be calculated. This task is solved on the trajectory planning level.

4.2 Level of Optimal Trajectory of Robot Motion Planning

The trajectory planner receives the geometrical tracks as input and determines a time history of position, velocity, acceleration and input torques, which are then fed to the trajectory tracker. On this level, the robot is represented by the manipulator's dynamics model [22],[23].

If the equations of the parameterized track $q(s)$ are substituted into the dynamic equations, they take on the following form [23],[24]:

$$m(s)\dot{\mu} + n(s)\mu^2 + r(s)\mu + g(s) = F \tag{9}$$

$$\dot{s} = \mu$$

where: $m(s)$ is the pseudo-inertia factor, $n(s)$ is the pseudo-Coriolis force factor, $r(s)$ the viscous friction factor and $g(s)$ is the pseudo- gravitational force factor.

Here μ is the time-derivative of the parameter s. The cost of motion along a track is assumed to be:

$$Cost = \int_{s_0}^{s_{max}} L(s, \mu, F)ds$$

The trajectory planning problem is then reduced to minimizing cost, subject to the dynamic model specification and the constraints on the torque F. This is a classical dynamical optimization problem for a non-linear system. It can be solved by using the dynamic programming method [23], [24]. Hence, we can obtain an optimal trajectory and the time of the manipulator's 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 for minimal-time or minimal-energy planning. Variant interpretations of the language instructions result in different realizations of the 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.

4.3 Level of 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 line (see function *ConAct* in Section 3.2) and the operational instructions that realize the actions. Thus, to define a simulator of the program, we model conditions that enable program instructions. Each machine d_i has the following DEVS [15] representation:

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

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 Dev_i 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:
$$\lambda : S \rightarrow Y.$$

A complete explanation of DEVS and its semantics is presented in [10].

The state set S_i of each Dev_i is defined as $S_i = \{s_a^i, s_b^i, s_c^i\}$ as given in Section 2.

Assume that the i-th position denotes the location of an input/output port of machine at which a part is placed. The set of external events for Dev_i is defined by the commands of the TORPL, namely:

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

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 machine i is given as follows:

$$\delta_{int}^i(s_a^i) = s_a^i \quad \delta_{int}^i(s_b^i) = s_b^i \quad \delta_{int}^i(s_c^i) = s_b^i$$

The external transition function for each machine i is defined as:

$$\delta_{ext}^i((s_a^i, x1_i) = s_c^i \quad \delta_{ext}^i((s_b^i, x2_i) = s_a^i$$
$$\delta_{ext}^i((s, x0) = s \quad \delta_{ext}^i((.,.),.) = failure \text{ for all other states}$$

The time advance function for Dev_i determines the time needed to process a part in the i-th machine. It is defined as follows: if $s = s_c^i$, then $ta^i(s) = \tau_i^k$ (the tooling/assembly time of operation k for machine i), otherwise $ta^i(s) = \infty$.

The above specification defines a model of the machining system. The activation of each machine Dev_i is caused by an external event generated by the robot's model. This model is realized by a generator of an experimental frame component [10] 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. The block diagram of the entire simulation system is given in Figure 4.

Rather than provide a detailed mathematical description of the experimental frame models here, we describe their functionality. (The reader is referred to [15] for a complete formal specification of the simulator of Figure 4.) The acceptor is a DEVS that receives as input state descriptions of each machine Dev_i. It selects events which invoke a 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 next operation are satisfied (see function $ConAct$)). 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., $HS = \{Empty, Holding\}$. The internal transition functions are represented by the following sequences in TORPL:

DEVS Simulator of Work-Cells

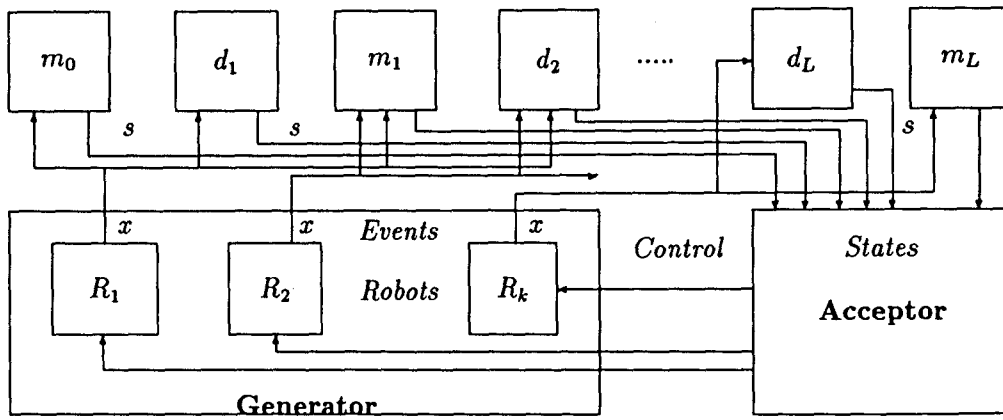

Fig.4 Discrete Event Simulator of Machining System

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

for the *Empty* state of the effector-end, and

```
MOVE HOLDING part TO k position
PLACE part ON k position
INITIALIZE start signal for machine k
```

for the *Holding* state of the effector-end.

This robot model reflects adequately each segment of the synthesized program. The time advance functions determine a) the sum of the time of motion to position and the time of the pickup operation, b) the sum of motion time from the position i to the position k and the time of the place operation on the k-th machine, for *Empty* and *Holding* states, respectively. The robot's model also generates external events (i.e., PICKUP, and PLACE) for machines Dev_i, which trigger their corresponding simulators.

The simulation model of the machining system is 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 k, τ_i^k, and the time the robot requires to service a workcell. The time τ_i^k depends on the type of machine on which the i-th operation is being processed. It is fixed but can be changed by replacing the machine. Similarly, the times of PICK UP and PLACE operations are determined by the type of part and machine on which the part is processed.

The times of the robot's inter-operational moves (transfers), $\tau_{i,j}^R$, depend on the geometry of the work-scene and the cost function of the robot's motion. This cost function determines the dynamics of motion along the geometric tracks and the duration of the moves. These data must be accessible in order to simulate the entire production system.

The simulation level completes the multilayer system for planning and programming robot tasks in flexible machining.

5 Summary and Conclusions

A comprehensive framework for generating a robot's program for an automated production system will require an integration of several layers of system theory-based support methods and tools. Each layer of the robot's program synthesis system requires different CAST tools. The tools for each level are:

- *Level 1:* graph search methods
- *Level 2:* Petri net methodology
- *Level 3:* discrete dynamical system methods
- *Level 4:* discrete optimization methods
- *Level 5:* event based system formalism

Our current research focuses on developing an architecture that will facilitate:

- automatic generation of different plans of sequencing operations realizing a given technological task (operations scheduling problem)
- synthesis of programs for robots servicing the devices
- planning and interpretation of robots' motion programs
- synthesis of autonomous robotic system's simulation models
- testing and verification of effectiveness of program execution 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. Most existing planning 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 work-scene. 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.

References

[1] F.Pichler, CAST Modeling Approaches in Engineering Design *Lecture Notes in Computer Science* **410**, 52-68, 1990

[2] E.G. Coffman, M. Elphick, A. Shoshani. System Deadlock.*Computing Surveys*, **3**(2),67-78, 1971

[3] H.M. Deitel *An Introduction to Operating Systems.* Addison-Wesley,1983

[4] B.H.Krogh, Z.Banaszak. Deadlock Avoidance in Pipeline Concurrent Processes. *Proc. of Workshop on Real-Time Programming* IFAC/IFIP, 1989

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

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

[7] 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.

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

[9] 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

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

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

[12] W.Jacak, Robot Task and Motion Planning. in:*AI, Simulation and Planning in High Autonomy Systems*, IEEE Computer Society Press, 168-176, 1990

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

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

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

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

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

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

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

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

[21] W. Jacak. Discrete Kinematic Modelling Techniques in Cartesian Space for Robotic System. in: *Advances in Control and Dynamics Systems*, ed. C.T. Leondes, Academic Press, (in print) 1991

[22] M. Brady ed. *Robot Motion: Planning and Control* MIT Press, 1986

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

[24] K.Shin, N.McKay, Minimum Time Control of Robotic Manipulator with Geometric Path Constrains. *IEEE Trans. on Automatic Control*, **30**(6), 531-541, 1985