Publication and subscription information:
Volume 15, Number 4, December 1998, ISSN 0740-6797 / 98 $3.00+.10. P.O. Box 17900, San Diego, California 92177. Telephone (619) 277-3888.
Subscriptions: $160 per year, U.S. Special rates to members of The Society for Computer Simulation.
Change of address: Send both old and new address to (include old label, if possible): Circulation Office, P.O. Box 17900, San Diego, California 92177. Allow six weeks for delivery. Use subscription number on all correspondence. Undelivered copies: Please return to address above.

TRANSACTIONS

of the Society for Computer Simulation International

VOLUME 15 NUMBER 4 DECEMBER 1998

Model-Based Workcell Task Planning and Control

Witold Jacak* and Jerzy Rozenblit**

* Johannes Kepler University, Institute of Systems Science, A-4040 Linz, Austria;
** Department of Electrical and Computer Engineering, The University of Arizona, Tucson, Arizona, USA

A framework for the design of a model-based control system for flexible manufacturing is presented in this paper. The system consists of three basic layers: the Task Planning, *the* Task-Level Programming, *and the* Simulation *layer. Task planning is based on the description of operations and their precedence relation. The resulting fundamental plan describes the decomposition of a manufacturing task into an ordered sequence of robot actions. The implementation of the plan is carried out using a* task-level programming *approach in which the detailed paths and trajectories, gross and fine motion, grasping, and sensing instructions, are specified. Plan sequences are verified by using a simulation modeling approach based on hierarchical, modular discrete-event system specifications.*

Keywords: Flexible manufacturing, discrete-event simulation, task planning, hierarchical control

Part I. Task Planning

1. Introduction

In recent years, the use of programmable and flexible systems has enabled partial or complete automation of machining and assembly of products. The economic importance of manufacturing has led to extensive efforts to improve the efficiency and cost effectiveness of automated production systems. More systematic approaches to the design and planning of such systems are needed to further enhance performance and to enable their cost-effective, real-world implementations.

Design of intelligent robotic systems, robotic assemblies, and intelligent, flexible manufacturing systems (FMS) uses concepts from different disciplines of science and engineering. The main objective in design of such systems is to ensure that they synthesize and execute their plans (sequences of actions). Moreover, advanced systems should be able to execute their plans in both normal and anomalous situations (which may require replanning).

From the artificial intelligence (AI) standpoint, central issues in plan generation are the system state representation, the system architecture, and the planning strategies [4, 2, 3, 18]. AI planning systems use explicit symbolic, logical, or temporal models of discrete operations to construct sequences of actions to achieve specific goals. Planning strategies are dominated by heuristic methodologies [1, 4].

From the general systems theory point of view, the key issue in design of automated systems is the theory of hierarchical, intelligent control. The theory is employed to organize, coordinate, and execute an anthropomorphic task with minimum supervision and interaction with a human operator [5, 7]. The system's hierarchical architecture is typically designed based on the rule of increasing intelligence with decreasing precision [6]. In robotic assemblies, the focus is on the development of automated techniques for plan generation and execution [17, 18].

This paper proposes a hierarchical task and control planner of an autonomous, flexible machining cell. The planner is designed using systems theory and hierarchical, modular simulation modeling concepts. A flexible machining cell is selected because it provides a good abstraction for the application of our framework to any complex production control problem. The cell is defined here as a production system composed of CNC-machines serviced by robots.

A variety of tasks can be executed on such machines [8]. We specify tasks as partially ordered sets of operations. They must be executed in order to "manufacture" a part or a system by employing available workstations (machines) and robots.

In a manufacturing environment, FMSs are generally constructed based on a hierarchical control architecture [9, 10]. The control hierarchy consists of the following levels: (1) *facility*, (2) *cell*, and (3) *workstation and equipment*. The levels in the control architecture have the following functions:

- The facility level implements the manufacturing, engineering, resource, and task management functions.
- The control functions at the cell level are jobs sequencing, scheduling, material handling, supervision, and coordination of the physical activities of workstations and robots.
- Machining operations are performed at the workstation level.

The overall architecture of the FMS control system is shown in Figure 1.

In the architecture depicted in Figure 1, the control mechanisms are established in such a way that the upper-level components issue commands to their lower-level descendants and receive feedback upon the completion of command execution by the descendants. The physical components at each level are computer systems and control devices, connected by a communication network, such as a local area network (LAN), with a

Received: February 1997; Revised: April 1998, Accepted: April 1998

TRANSACTIONS of The Society for Computer Simulation International
ISSN 0740-6797/98

Volume 15, Number 4, pp. 167-180

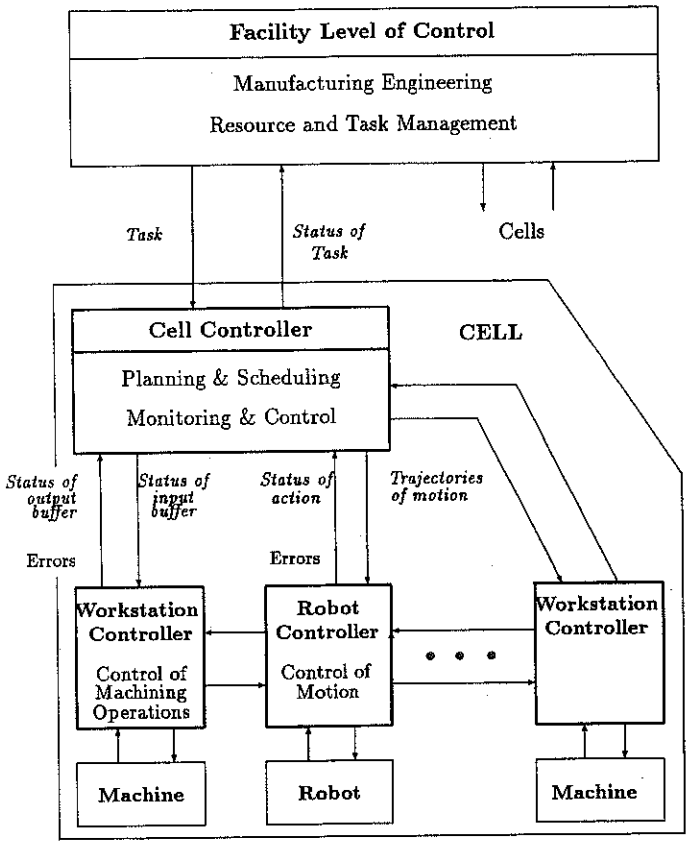

Figure 1. Intelligent hierarchical controller of FMS

manufacturing automation protocol (MAP) [9]. Control software is a key component in achieving a high degree of FMS flexibility.

In the ensuing sections, we propose a design of an intelligent control system of a manufacturing cell, which can plan the sequence of technological operations and motions of robots servicing the cell. The system consists of two basic layers: the *Task Planning* and the *Task-Level Programming* layer. Task planning is based on the description of technological operations and on their precedence relations. The resulting fundamental plan describes the decomposition of a manufacturing task into an ordered sequence of robot and machine actions. The implementation of the plan is carried out using a *task-level programming* approach in which the detailed paths and trajectories, gross and fine motion, grasping, and sensing instructions are specified.

Thus, this article is organized in a manner that reflects the two main facets of our problem (i.e., the task planning and task programming). Part I introduces the basic formal terms and the model of the task planning process. Part II demonstrates how the discrete event specification (DEVS) [20] is used to model and simulate the workcell.

2. Basic Notions

We now proceed to define the production system abstractions. A flexible manufacturing system (FMS) is a set of programmable machines (technological devices), D, called *workstations* and *product stores*, M, connected by a flexible material handling facility, R, (e.g., a robot or an automated guided vehicle), and controlled by a computer network connected to a sensory system. An FMS can perform technological operations such as fabrication, machining, or assembly. All machines and material handling systems (robots) are highly automated. The groups of workstations serviced by robots are called *cells*. Each workstation has its own control and programming system.

A workstation (machine) d can have a buffer $B(d)$. Parts are automatically fed into a machine from the buffer, are machined, and can be stored in the buffer if necessary. Depending on the type of a technological operation to be executed, various tooling programs can be used to control the machining process.

We define technological task formally as follows:

Definition 2.1

A **task** *realized by an FMS cell can be represented by a three-tuple:*

$$Task = (O, <, \alpha) \tag{1}$$

where:

O is a finite set of technological operations required to process parts (e.g., machining, assembly, test, etc.),

$< \subset O \times O$ is a weak-ordering precedence relation such that if $o_i < o_j$, then the operation o_i must precede the operation o_j,

$\alpha = (\alpha_1, \alpha_2)$ where $\alpha_1 \subset O \times D$ is the device assignment relation, and $\alpha_2 \subset O \times M$ is the product store assignment relation.

$(o,d) \in \alpha_1$ denotes that the operation o can be performed on the workstation d. If $(o, m) \in \alpha_2$, then m is the production store from the set M where parts can be stored after the operation o has been completed.

In this paper, we restrict the analysis to machining processes. The operation o_i from the set O can be performed by each workstation d from the set $\alpha_1(o_i) \subset D$. Furthermore, we assume that α_1 is a function; i.e., the operation o_i can be performed only by one workstation $d_i = \alpha_1(o_i)$. Now, presume additionally that the empty set belongs to the set M. Hence $\alpha_2(o_i) = \emptyset$ means that a part after the operation o_i is not stored in any production store—it remains in the buffer of the machine $\alpha_1(o_i)$.

Each workstation $d \in D$ has its own program for processing a part. This program realizes a technological operation o from the set $\alpha_1^{-1}(d)$. Parts are transferred between machines by the robots which service the cell. A robot $r \in R$ can service only those machines which are within its service space $Srv\_Sp(r) \subset E^3$. The set of devices which lie in the robot's r service space is denoted by $Range(r) \subset D \cup M$. More specifically, $d \in Range(r)$ if all positions of the buffer $B(d)$ (i.e., buffer of machine d) lie in the service space of r. Moreover, we assume that for each operation o_i there exists a robot r_i which can transfer parts between the workstation d_i and stores m_i from the set $\alpha_2(o_i)$, i.e.:

$$\{d_i, m_i\} \subset Range(r_i)$$

Based on $Range(r)$, we define the relation β which describes the transfer of parts after each technological operation:

$$\beta \subset (O \times O) \times R \tag{2}$$

where:

$$((o_i,o_j), r) \in \beta \Leftrightarrow \{d_i,d_j\} \subset Range(r) \vee \{m_i,d_j\} \subset Range(r)$$

In a special case β is a partial function, i.e.: $\beta : O \times O \rightarrow R$.

To automatically generate the cell-control program, we must first determine the succession of technological operations. Then, for each robot r servicing the process, a set of cell-state dependent time trajectories of the robot's motions must be established for all the operations of a given task:

$$T_s = \{q_r(s) : \tau_s \rightarrow Q_r \mid s \in \textbf{Cell\_States}\} \qquad (3)$$

where the trajectory $q_r(x)$ is a function mapping the time interval τ_s into the joint space Q_r of the robot r [14, 25]. These functions realize the transfers of parts between machines of the cell and, for each part, they ensure that all the operations from *Task* are executed in a pre-planned order. In addition, the sequence of operations and related robot actions should minimize the parts mean flow time or other criteria [10].

The problem of intelligent control algorithm synthesis may have a large number of possible solutions. The solutions can be different with respect to sequences of technological operations, sequences of sensor-dependent robot actions, geometric forms of a manipulator's paths, and the dynamics of movements along the paths. Thus, to reduce the complexity of this problem, we propose to apply a hierarchical decomposition process to break down the original problem into two subproblems. This method solves the cell control program synthesis which we formulate at each successive level of the cell's behavioral modeling abstraction. The proposed control program consists of two layers: the *Task-Level Planning* and the *Task-Level Programming* layer.

Task-Level Planning is carried out based on a description of the technological operations, a description of the workcell and its resources such as machines, robots, fixtures, or sensors, and a description of the precedence relation over the set of operations. The resulting fundamental cell action plan describes a decomposition of the task into an ordered sequence of technological operations called *Process*. Then, the *Process* sequence is translated into an ordered sequence of robot and machine actions which are used to realize the task.

The solution is generated as follows: two subproblems are defined. The first is to find an ordered, feasible sequence of technological operations which can be transformed directly into the sequence of robot and machine actions. In the second subproblem, we define the set of preconditions for each action, which guarantee that a deadlock does not occur among the actions being executed. The fundamental cell-action plan for a machining task determines the robot's program required to carry out this task. Such a program is a sequence of motion, grasp, and sensor instructions expressed in the *Task-Oriented Robot Programming Language* (TORPL) [22, 23].

The implementation and interpretation of the fundamental plan is carried out using the *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, we again propose a two-level system.

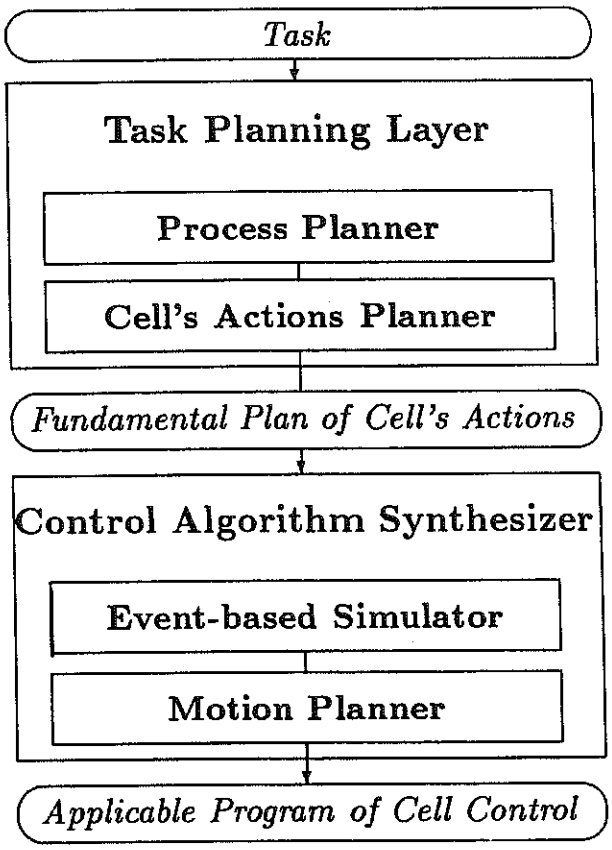

Figure 2. Structure of the cell-control synthesizer

The first level is a *Discrete-Event Simulator* of the manufacturing process. The simulator uses the Discrete-Event System Specification (DEVS) [19, 20] formalism to model actions and technological devices (we define the basic tenets of DEVS in Section 4). The second level is the *Motion Planner* [25, 26, 28] employed to specify each individual robot action. The planner creates variants of collision-free time trajectories of the manipulator which executes each action. It uses robot-dependent planning techniques and the discrete dynamical system formalism [28]. The structure of the multilayer system of cell-control synthesis is shown in Figure 2.

In the ensuing section, we define the two layers of the task planning system.

3. Task Planning Layer

The *task planner* is responsible for the first phase of the cell's program synthesis, namely, the generation of the fundamental plan of task realization. The basic problem at this level is the derivation of an ordered sequence of robot and machine actions which perform a task. Here, the robot's model is reduced to a fundamental action:

Action_ robot(r) = Transfer **part** From **a** To **b**

which denotes a transfer of a part from a workstation or a store **a** to a workstation or a store **b**. The machine's activity is represented by:

Action_ machine(d) = Execute **o** On **b**

which is interpreted as the beginning of the execution of an operation o from the set O on the machine **b**.

The sequence of robot and machine actions depends directly on the order of the technological operations of *Task*. Hence, the *task planning* problem can be reduced to the problem of finding a feasible, ordered sequence of technological operations that minimizes the number of possible deadlock occurrences.

To derive the sequence of operations and, then, the sequence of actions, we decompose the task planning problem into two subproblems: *the technological process planning* problem and the *synthesis of fundamental plan of cell actions* problem. These subproblems are solved by the Task Planning Layer.

3.1 Technological Process Planning

In the process planning phase, we must find an ordered sequence of technological operations from **Task**, called *process*, with a minimum number of deadlock cases. This problem is related to operations scheduling [10,14]. To explain it in more detail, we introduce some additional notions.

Definition 3.1

An ordered sequence of operations is called a **pipeline, sequential machining process** *and is denoted as*:

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

if the following conditions hold:

A. *if for two operations o_i and o_j from Task, $o_i < o_j$, then $i < j$*

B. *for each $i = 1, ..., L-1$, there exists a robot which can transfer a part from machine d_i or store m_i (assigned to operation o_i) to machine d_{i+1} assigned to operation o_{i+1}, i.e.:*

$$(\exists r \in R)(((o_i, o_{i+1}), r) \in \beta)$$

A process can be realized by different devices (called *resources*) required by the successive operations from the list *Process*. The set of device orderings is called *production routes* and is denoted by **P**. The production route $\mathbf{p} \in \mathbf{P}$ is an ordered list of resources which has $2L + 1$ stages, where L denotes the length of the list *Process*. The production route is created online during the execution of the machining operations.

Definition 3.2

The production route is defined as follows:

$$\mathbf{p} = (p(i) \mid i = 0, 1, ..., 2L) \tag{5}$$

Such a production route has always $2L + 1$ elements. Some of them may be equal to zero. The "minimal" production route has a minimum number of production stores m. We denote it as:

$$\mathbf{p}_{\min} = (m_0, device(o_1), device(o_2), ..., device(o_L))$$

where $device(o_i) = \alpha_1(o_i) = d_i$ if robot $r = \beta(o_i, o_{i+1})$ can transfer parts directly from d_i to d_{i+1} (i.e., $\{d_i, d_{i+1}\} \subset Range(r)$) and $device(o_i) = (\alpha_1(o_i), \alpha_2(o_i)) = (d_i, m_i)$ if the robot r can transfer parts from m_i to d_{i+1} only (i.e., $\{m_i, d_j\} \subset Range(r) \wedge d_i \notin Range(r)$).

Each execution of a process is called a *job*. A job J is characterized by a production route **p**, its *start time*, and two *status*, i.e., a state that the job is (for example, allocated, processing, completed, waiting for a resource in the production route).

The production route **p** has L stages. During the stage i, the operation o_i is executed and thus the resource (machine) $d_i = \alpha_1(o_i)$ is required for a finite period of time. After this interval, if it is not possible to allocate the store $m_i = \alpha_2(o_i)$ to the process for some waiting time units, the next machine $d_{i+1} = \alpha_1(o_{i+1})$ must be assigned to the process. This allocation strategy places a higher priority on the allocation of machines than on the allocation of stores. The process continues to hold the resource d_i or m_i and cannot advance to stage $i + 1$ until the machine d_{i+1} is allocated to it. This creates a potential for deadlocks among a set of successive executions of processes in the cell.

As we have noted before, the process being planned should minimize the number of deadlock cases. Avoiding deadlock causes the increase in job waiting time and, consequently, the increase in the makespan. Therefore, in designing the cell planning and control algorithm we should carefully examine the deadlock avoidance conditions and their effects on the entire process. A quality criterion of process planning should be defined to assess how well the algorithm performs. Before we define such a criterion, a deadlock avoidance procedure is presented [12].

In the workcell considered in this paper, the so-called *circular wait-deadlock* among pipeline processes can occur.

Definition 3.3

Circular wait occurs if there is a closed chain of jobs in which each job is waiting for a machine held by the next job in the chain [11].

To avoid it, the minimal production route, $\mathbf{p}_{min}$, is partitioned into a unique set of Z sublists called *zones*.

$$\mathbf{p}_{min} = (z_k \mid k = 1, ..., Z) \tag{6}$$

where every zone has the form $z_k = (s_k, u_k)$ and:

$u_k = (u^i_k \mid i = 1, ..., I(k))$ is the sublist of resources which appear only once in a production route, i.e., the number of machines in an unshared zone k. Such resources are called unshared resources. In addition, if $p(i) = p(i+1)$ and there exists no $p(k) = p(i)(k \neq i, i + 1)$, then $p(i)$ and $p(i + 1)$ are unshared resources, too.

$s_k = (s^j_k \mid j = 1, ..., J(k))$ is a sublist of resources which are used more than once in a route, i.e., the number of machines in a shared zone k.

The sublists u_k and s_k of the zone z_k are referred to as the unshared and shared subzones of z_k, respectively.

We now consider a deadlock avoidance algorithm based on the method described in [12]. The deadlock avoidance algorithms proposed for computer operating systems [11, 12, 13] are not efficient for pipeline processes. They do not incorporate the production route information which indicates the specific order in which resources of a cell must be allocated and deallocated to jobs.

A *resource allocation policy* is a rule for allocating a resource to a job. In general, unrestricted allocation of an available resource required by a job can lead to a deadlock among a

set of jobs in the system. In other words, a set of waiting jobs are in an unrestricted deadlock when each is waiting for an unavailable resource which will never become available under any resource allocation policy. To avoid potential deadlocks, we use the so-called *restricted allocation policy* proposed in [12, 13].

A *restricted policy* is a rule for defining a subset of waiting jobs for which the resource allocation decision is made. Clearly, a restriction policy could lead to a condition in which a set of jobs can never progress even though they are not deadlocked in the unrestricted sense.

The restricted allocation policy is defined as follows:

Assume that the workstation requested by the process is free. Then, the policy allows it to be loaded:

A. If the requested workstation is unshared and there is at least an additional free workstation in the entire unshared zone, and

B. None of the workstations is shared and the capacity of each shared workstation is not at its maximum (i.e., at least one job can be allocated to it).

This allocation policy leads to following fact:

Fact 3.1

The circular-wait deadlock can never occur under the restricted allocation policy described in rules A and B.

We use the above conditions to formulate the *planning quality criterion*. Our goal is to optimize the production rate with respect to job waiting times.

Based on [12], another fact (3.2) can be derived. Let zone z_k have $n(k) = J(k) + 1$ elements. $J(k)$ is the number of shared machines and 1 denotes all the unshared devices. Assume that the probability of a machine being completely full is equal to w.

Fact 3.2

The probability that the job will have to wait for processing is $w(1-(1-w)^{n(k)-1})$.

The more elements (machines) belong to a zone (the subzone u_k is treated as one element), the higher the probability that a job will wait for resource allocation. Thus, the production routes should contain zones with a minimum number of elements. The problem of production route synthesis with a minimum number of elements in the zones is equivalent to the problem of route synthesis with a maximum number of zones.

Now, we can introduce the measure of route quality. Let $\mathbf{p}_{min} = (z_k \mid k = 1, ..., Z)$ be the minimal production route of *Process*, and $z_k = (s_k, u_k)$ where $u_k = (u^i_k \mid i = 1,...I(k))$ is the unshared subzone and $s_k = (s^j_k \mid j = 1, ... J(k))$ is the shared subzone. In some cases, the first zone can contain only an unshared subzone (i.e., $z_1 = u_1$) and the last subzone can contain only a shared subzone (i.e., $z_Z = s_Z$). By $n(k)$, we denote the number of elements of the z_k zone. If $z_k = (s_k, u_k)$, then $n(k) = J(k) + 1$; if $z_k = u_k$, then $n(k) = 1$ and if $z_k = s_k$, then $n(k) = J(k)$. In addition, if there exist s^i_k and s^{i+1}_k in subzone s_k such that $s^i_k = s^{i+1}_k$, then $n(k) = (J(k) - 1) + 1$.

The measure of route quality is defined as follows:

$$v(\mathbf{p}_{min}) = \max_k \{n(k) \mid k = 1, ... Z\} \qquad (7)$$

The function $v(\mathbf{p}_{min})$ is used to evaluate the technological process being planned. Now, we must find a feasible, ordered sequence of operations from *Task* which minimizes the function $v(\mathbf{p}_{min})$. This is a permutation problem which may have more than one solution. To solve it, we proceed as follows: *Task* is represented by a directed acyclic graph. Let $Process^K = (o_1, o_2, .. o_K)$ $(K < L)$ be a subprocess of *Process* and $\mathbf{p}^K_{min} = (z_k \mid k = 1, .., Z^K)$ be a subroute of minimal route $\mathbf{p}_{min}$. Then, a backtracking graph search algorithm is applied. The process planning algorithm (PPA) is defined below:

```
Form a list START of operations with no predecessors,
  i = 1, m = 0, v^opt = ∞
while ( START ≠ ∅ ) {
  Remove an operation from START and put it on PATH
  BACK_i = START
  while ( PATH ≠ ∅ ) {
    if (PATH = O (O is the set of operations)) {
      m = m + 1
      Process_m = PATH, p^m = p
      Put Process_m on PROCESSES
      v^opt = v ( p^m_min )
(**)    Remove nonoptimal sequences Process from
          PROCESSES
    } else {
      i = j for o_j last operation in PATH
      Generate Suc(o_j)
      if ( Suc(o_j) = ∅ ) {
        remove o_j from PATH
        remove o_j from Suc (o_j-1)
        i = j - 1
        continue
      }
      if ((∃q ∈ Suc(o_j) & ∃o_l ∈ PATH)( q < o_l)) {
        remove all elements of PATH beginning with o_l
        remove o_l from Suc(o_l-1)
        i = l - 1
        continue
      }
      i = i + 1
      BACK_i = { q ∈ Suc(o_j) | f(q) = min_j f(o_j) }
      remove an operation o_i from BACK_i
      if ( f(o_i) ≤ v^opt ) {
        insert o_i at end of PATH
        continue
      }
    }
    Find max index i for which BACK_i ≠ ∅
    Remove all elements of PATH beginning with o_i
  }
}
```

The evaluation function $f(q)$ is calculated as follows: Let PATH be a list $(o_1,..o_j)$. Let $q \in Suc(o_j)$.

- Create temporarily the subprocess $Process^{j+1} = (o_1, o_2, .., o_j, q)$. Calculate its production subroute $\mathbf{p}^{j+1}_{min}$ and decompose it into zones $(z_k \mid k = 1, .., Z^{j+1})$.
- Calculate the measure of route quality $v(\mathbf{p}^{j+1}_{min}) = \max_k \{n(k) \mid k = 1, ... Z^{j+1}\}$ and set $f(q) = v(\mathbf{p}^{j+1}_{min})$.

The output list PROCESSES in the above procedure contains only feasible, ordered sequences of technological operations which realize the *Task*. The PPA algorithm is a graph search. It starts with a set of operations that have no predecessors in the task precedence graph. It then expands a node from this set to generate partial routes. The best path continues to be expanded. Should a node be reached on an incomplete path that has no successor, backtracking occurs.

Fact 3.3

If there exists a production route for a given Task, then the PPA procedure finds optimal ordered sequences of operations, computed by the function V.

Proof:

Let $Process^j = (o_1, o_2, .., o_j)$ $(j < L)$ be a subprocess *Process* and $\mathbf{p}^j_{min} = (z_k \mid k = 1, .., Z^j)$ be a subroute of $\mathbf{p}_{min}$. Notice that $V(\mathbf{p}^j_{min}) \le V(\mathbf{p}_{min})$. We eliminate the subprocess $Process^{j+1} = (o_1, o_2, .., o_j, q)$ if $V(\mathbf{p}^{j+1}_{min}) > V(\mathbf{p}^*_{min})$, where p* is a previously obtained optimal route. It is clear that in this case:

$$V(\mathbf{p}_{min}) \ge V(\mathbf{p}^{j+1}_{min}) > V(\mathbf{p}^*_{min})$$

In step (**), the suboptimal solutions are eliminated. □

Each *Process* from the list PROCESSES determines a different fundamental plan of robot and machine actions, and, respectively, a different law of cell control. To minimize the job flow time, all variants of the fundamental plan should be tested.

3.2 Fundamental Plan Synthesis

Based on the sequence of operations *Process* obtained through task planning, we create a fundamental plan of actions for the cell's components, i.e.:

$$Plan = (Action_i \mid i = 1, ..., L)$$

Each $Action_i$ consists of three parts which determine:

a. The preconditions of an action,

b. The robot's motion parameters to execute the i^{th} operation of *Process*, and

c. The action execution parameters.

The i^{th} action has the following form:

$$Action_i = [\text{Cond } (o_i), \text{Transfer } (o_{i-1}, o_i), \text{Execute } (o_i)]$$

The Cond (o_i) segment describes the preconditions which must be satisfied in order for the operation o_i to be executed. It also establishes the geometric parameters for the robot's motion trajectories. The preconditions are formulated as a function of both the workcell's and job's state.

Let $i\text{–}Br(d)$ denote the state of the i^{th} position of the workstation d's buffer $B(d)$, where $1 \le i \le C_b(d)$. This state can be characterized as follows:

- $i\text{–}Br(d) = (0,0)$—the position is free,
- $i\text{–}Br(d) = (j,0)$—the position is occupied by a part before the operation o_j,
- $i\text{–}Br(d) = (j,1)$ The position is occupied by a part after the operation o_j.

Using the above specification, the Cond(o_i) segment can be defined by the following instructions of TORPL:

```
LOOP FOR(i=1,...,L)
  IF there exists j-Buffer(d(i))=(0,0)

    IF there exists k-Buffer(d(i-1))=(i-1,1)
      SET Pickup_position=Frame_k-Buffer(d(i-1))
    ELSE IF there exists k-Store(m(i-1))=(i-1,1)
       SET Pickup_position=Frame_k-Store(m(i-1))
    ELSE continue LOOP

    SET Place_position =Frame_j-Buffer(d(i))

    IF PROCEDURE(Deadlock_Av(d(i))=True
      BEGIN Transfer & Execution

  ELSE IF there exists k-Buffer(d(i-1))=(i-1,1)
       IF there exists j-Store(m(i-1))=(0,0)
         SET Pickup_position=Frame_k-Buffer(d(i-1))
         SET Place_position =Frame_j-Store(m(i-1))
       ELSE continue LOOP

  ELSE continue LOOP

END LOOP
```

By $d(i)$ and $m(i)$ we denote the workstation and store assigned to the operation o_i, i.e., $d(i) = \alpha_1(o_i)$ and $m(i) = \alpha_2(o_i)$. The call PROCEDURE(Deadlock_Av(d(i)) checks to see if the deadlock avoidance conditions for device $d(i)$ are satisfied. If they are satisfied, then the sub-actions Transfer and Execution are activated. In addition, the geometrical parameters of the Transfer sub-action are established. The parameters indicate the objects between which Transfer has to be performed. This sub-action is realized by a robot servicing the workstation $d(i)$.

The elementary robot actions can be expressed as TORPL instructions. The instructions may be interpreted in many various ways. The basic macro-instructions of TORPL are: MOVE (EMPTY, HOLDING) TO *position*, GRASP, PICKUP AT *position*, PLACE ON *position*, WAIT FOR *sensor input signal*, INITIALIZE *output signal*, OPEN GRIPPER, and CLOSE GRIPPER [17, 22, 23, 25]. The basic instructions may be combined into higher level macros, e.g., the PICK-AND-PLACE instruction [25].

In this set of instructions, the action Transfer (o_{i-1}, o_i) is interpreted as the PICK-AND-PLACE part FROM x TO y macro, where x denotes geometrical data of a workstation's buffer (store), and y is the position and orientation of the buffer of the next workstation. The parameters x and y are determined by segment Cond(o_i), namely, `x=Pickup_frame` and `y= Place_frame`. This macro-instruction is decomposed into a sequence of more primitive instructions as illustrated below [23, 25, 26]:

```
[PICK-AND-PLACE part FROM x TO y :=]
   MOVE EMPTY TO x+v (x=position of buffer of machine d
            and v is the approach vector)
      PICKUP AT x :=
         [begin PICKUP]
            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 the next machine buffer)
PLACE ON y
[end PICK-AND-PLACE]
```

The above sequence of instructions is used to synthesize the robot's program. The instructions for the action Execute can be translated into TORPL in a similar manner.

This completes the definition of the fundamental plan of cell's actions specified based on the chain of technological operations *Process*. Given the set *Actions* we can define the control process for the robots and workstations. We can also specify the time trajectories of robot motions for each operation. Each chain of technological operations from *Process* generates a different fundamental plan of cell actions and different cell control algorithms. Each route determines a specific topology of robot motions tracks, distinct deadlock avoidance conditions, and a certain job flow time. Therefore, route planning is critical for problems such as the maximum rate and minimum jobs-in-process optimization problems.

PART II: Modeling and Simulation

This part demonstrates the application of simulation modeling concepts to the synthesis of the task level controller. More specifically, we show how the DEVS specification is used to model the fundamental cell components and how the overall system model is built. Then, an example that illustrates the theory-based concepts is presented.

4. Task Programming Layer—Control Algorithm Synthesizer

The control algorithm synthesis requires that we introduce conditional instructions which depend on: (a) the states of each machine d_i of the cell (sub-action Cond), and (b) the operational instructions that realize the actions (segments Transfer and Execute). To minimize the flow time, variants of the fundamental plan and motion interpretation should be tested by a simulator.

4.1 Workcell Modeling and Simulation

To model the cell, the Discrete-Event System specification (DEVS) formalism is used [20]. DEVS is a hierarchical, modular formalism developed by Zeigler [20]. In this formalism one must specify basic models from which larger ones are built, and how these models are connected together in a hierarchical fashion.

Each workstation $d \in D$ is modeled as an atomic DEVS [24]. Such a model has the following structure:

$$Dev_d = (X(d), S(d), Y(d), \delta^d_{int}, \delta^d_{ext}, \mathrm{ta}^d) \qquad (8)$$

where:

$X(d)$ is a set, the external input events
$S(d)$ is a set, the sequential states
$Y(d)$ is the output value set
δ^d_{int} is a function, the internal transition specification
δ^d_{ext} is a function, the external transition specification
ta^d is a function, the time advance function

with the following constraints:

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

Further explanation of DEVS and its semantics is presented in [20].

Each workstation $d \in D$ can have a buffer. The capacity of this buffer is denoted by $C_b(d)$. The capacity of a workstation is equal to $C(d) = C_b(d) + 1$. If a workstation has no buffer, then $C(d) = 1$.

Let NC_$Reg(d)$ (NC programs register) denote the set of operations performed on the workstation d, i.e., NC_$Reg(d) = \{o \in O \mid \alpha_1(o) = d\}$.

The state set of d is defined by:

$$S(d) = SD_d \times SB_d$$

where SD_d denotes the state set of a machine d and SB_d denotes the state set of its buffer. The state set SD_d is defined as:

$$SD_d = S_{free} \cup S_{proc} \cup S_{compl}$$

where:

- S_{free} = {free} signifies that **machine is free**
- $S_{proc} = \{(prc, q) \mid q \in \mathrm{NC\_}Reg(d)\}$ and (prc, q) signifies that **machine is busy processing q-operation**
- $S_{compl} = \{(com, q) \mid q \in \mathrm{NC\_}Reg(d)\}$ and (com, q) signifies that **machine has completed q-operation and is not free**

The state set of the workstation's buffer SB_d is defined as:

$$SB_d = (O \times \{0, 1, \#\})^{C_b(d)}$$

Let $C_b(d) = K$ and $sb_d = (b_i \mid i = 1, \ldots, K) \in SB_d$, then:

$$b_i = \begin{cases} (0,0) \Leftrightarrow & i^{\mathrm{th}} \text{ position of the buffer is free} \\ (0,\#) \Leftrightarrow & i^{\mathrm{th}} \text{ position of the buffer is reserved for a part being currently processed} \\ (q,0) \Leftrightarrow & i^{\mathrm{th}} \text{ position of the buffer is occupied by a part before operation q} \\ (q,1) \Leftrightarrow & i^{\mathrm{th}} \text{ position of the buffer is occupied by a part after operation q} \end{cases}$$

The state of the workstation's buffer is described by a vector whose coordinates specify the current state of each position in the buffer. We assume that the i^{th} position denotes a location at which a part is placed in the buffer.

The set of external events for the workstation's model Dev_d is defined as follows:

$$X(d) = \{(e_d^1(q, i), e_d^2(q, i), e^o) \mid q \in \text{NC\_Reg}(d) \wedge i = 1, ..., K\}$$

where:

$e_d^1(q, i)$ = PLACE $(q, 0)$ ON i^{th} *position*—signifies that a part before q operation is placed on i^{th} position of d-workstation's buffer

$e_d^2(q, i)$ = PICKUP $(q, 1)$ AT i^{th} *position*—signifies that a part after q operation is removed from i^{th} position of d-workstation's buffer

e^o = DO NOTHING

Given the state and external event set, we now define the transition functions. Let $s(d) = (s_d, (b_i \mid i = 1, .., K)) \in SD_d \times SB_d$. Then the internal transition function:

$$\delta_{int}^{d} : S(d) \rightarrow S(d)$$

is specified as shown in Table 1.

The external transition function δ_{ext}^{d} for each workstation d is defined in Table 2.

The time advance function for Dev_d determines the time needed to process a part on the d^{th} workstation. It is defined as follows:

Let $s(d) = (s_d, (b_i \mid i = 1, .., K))$. Then:

$$ta^d(s(d)) = \begin{cases} \tau_p^d(q) & \Leftrightarrow \; s_d = (prc, q) \in S_{proc} \\ \tau_l^d & \Leftrightarrow \; s_d = free \in S_{free} \\ & \quad \wedge (\exists i)\ (i = \min_j \{j \mid b_j = (q, 0)\} \\ \tau_s^d & \Leftrightarrow \; s_d = (com, q) \in S_{compl} \wedge b_i = (0, \#) \\ \propto & \text{otherwise} \end{cases}$$

$\tau_p^d(q)$ denotes the tooling/assembly time of operation q for the workstation d. τ_l^d, τ_s^d denotes the loading and unloading times for d.

The above is a complete model of a cell's workstation.

We can define a model of a production store in a similar manner. The state set of a store m is specified as a vector of states of each store position, i.e.:

$$S(m) = (O \cup \{0\})^{C(m)}$$

where: if $s_m = (s_i \mid i = 1, ..., C(m)) \in S(m)$, then:

$$s_i = \begin{cases} 0 & \Leftrightarrow \text{ i}^{th} \text{ position of store is free} \\ q & \Leftrightarrow \text{ i}^{th} \text{ position of store is occupied by a part after operation } q \end{cases}$$

The set of external events for m is similar to the set of external events of a workstation.

$$X(m) = \{(e_m^1(q, i), e_m^2(q, i), e^o) \mid q \in O \wedge i = 1,., C(m)\}$$

where:

$e_m^1(q, i)$ = PLACE q ON i^{th} *position*—signifies that a part after q operation is placed on i^{th} position of m store

$e_m^2(q, i)$ = PICKUP q AT i^{th} *position*—signifies that a part after q operation is removed from i^{th} position of m store

e^o = DO NOTHING

The internal transition δ_{int}^{m} is an identity function which can be omitted. The external transition function δ_{ext}^{m} for each store m can be defined in the same manner as the function δ_{ext}^{d}. The time advance function for Dev_m is equal to ∞. Hence, the event model of a store is given by:

$$Dev_m = (X(m), S(m), \delta_{ext}^{m}) \qquad (9)$$

The activation of each machine is caused by an external event generated by the model of a robot. Such a model is realized by a generator associated with the cell's model. The events generated by the cell's robots depend on the states of the workstations

Table 1.

$$\delta_{int}^{d}(s(d)) = \begin{cases} ((com, q), (b_1, b_2, ..., b_K)) & \Leftrightarrow \; s_d = (prc, q) \in S_{proc} \\ ((prc, q), (b_1, ., b_{i-1}, (0, \#), b_{i+1}, ., b_K)) & \Leftrightarrow \; s_d = free \in S_{free} \wedge \\ & \quad (\exists i)\ (i = \min\{j \mid b_j = (q, 0)\} \\ (free, (b_1, ., b_{i-1}, (q, 1), b_{i+1}, ., b_K)) & \Leftrightarrow \; s_d = (com, q) \wedge b_i = (0, \#) \end{cases}$$

Table 2.

Let $s(d) = (s_d, (b_i \mid i = 1, .., K)) \in SD_d \times SB_d$, then

$\delta_{ext}^{d}((s(d), t), e^o) = s(d)$

$\delta_{ext}^{d}((s(d), t), e_d^1(q, i)) = (s_d, (b_1, ., b_{i-1}, (q, 0), b_{i+1}, ., b_K)) \Leftrightarrow b_i = (0, 0)$

$\delta_{ext}^{d}((s(d), t), e_d^2(q, i)) = (s_d, (b_1, ., b_{i-1}, (0, 0), b_{i+1}, ., b_K)) \Leftrightarrow b_i = (q, 1)$

$\delta_{ext}^{d}((.,.),.)$ = *failure* for all other states

d_i and a given fundamental plan *Process*. Thus, we define a device (called an *acceptor*) which observes the states of each workstation. The acceptor is defined as the following discrete event system [20]:

$$A = (X_A, PROCESSES, Y_A, \lambda^A) \tag{10}$$

where:

$$X_A = \{(s, t) \mid s \in S_{cell} \wedge t \in Time\}$$

$$S_{cell} = S(d_1) \times S(d_2) \times ... \times S(d_D) \times S(m_1) \times ... \times S(m_M)$$

and *Time* is the time base.

The role of the acceptor is to select the events which will invoke the robots to service the workstations. Let us define the set of acceptor outputs as set of subsets of operations, robots, and position indexes, i.e.:

$$Y_A \subset 2^{ORP}$$

where: set $ORP = O \times R \times ((D \cup M) \times N)^2$ and N denotes the set of natural numbers.

Then, the acceptor's output function is given by:

$$\lambda^A : X_A \times PROCESSES \rightarrow Y_A$$

and

$$\lambda^a ((s,t), Process) = \{(q, r, (w, i), (v, j)) = x \mid Activ(x, s) = \text{TRUE} \wedge q \in Process\}$$

where $s = (s(d_1), s(d_2), ..., s(d_D), s(m_1), .., s(m_M))$.

The predicate $Activ(q, s)$ determines which operation from $Process \in PROCESSES$ requires service when the cell's state is equal to s, namely:

For $q = o_k \in Process$ and $s \in S_{cell}$ the function $Activ(q, s)$ = TRUE iff one of following conditions holds:

Variant a: $\alpha_1(o_k) = v$ and for $s(v) = (s_v, b_v)$ the j[th] coordinate $b_{vj} = (0, 0)$ and $\alpha_1(o_{k-1}) = w$ and for $s(w) = (s_w, b_w)$ the i[th] coordinate $b_{wi} = (o_{k-1}, 1)$ and $\beta(o_{k-1}, o_k) = r$ and conditions of Fact 3.1 for the workstation v are satisfied.

Variant b: $\alpha_1(o_k) = v$ and for $s(v) = (s_{vd}, b_v)$ the j[th] coordinate $b_{vj} = (0, 0)$ and $\alpha_1(o_{k-1}) = u$ and for all coordinates $b_{ui} \neq (o_{k-1}, 1)$ and $\alpha_2(o_{k-1}) = w$ and for $s(w)$ the i[th] coordinate $s_{wi} = o_{k-1}$ and $\beta(o_{k-1}, o_k) = r$ and conditions of Fact 3.1 for the workstation v are satisfied.

Variant c: $\alpha_1(o_k) = w$ and for $s(w) = (s_w, b_w)$ the i[th] coordinate $b_{wi} = (o_k, 1)$ and $\alpha_1(o_{k+1}) = u$ and for all coordinates $b_{uj} \neq (0, 0)$ and $\alpha_2(o_k) = v$ and for $s(v)$ the j[th] coordinate $s_{vj} = 0$ and $\beta(o_k, o_{k+1}) = r$.

$Activ(q,s)$ = FALSE for all other cases.

The output of the acceptor can be an empty set or it contains indexes of only those operations which can execute without deadlock. Not all the operations can be performed simultaneously. Therefore, to select the operations which can be executed concurrently, we introduce a device called *selector*. This system is defined as follows:

$$SEL = (Y_A, RULES, \lambda^{Sel}) \tag{11}$$

where:

- Y_A is the acceptor output set,
- RULES is the set of strategies for operation selection,
- $\lambda^{Sel} : Y_A \times \text{RULES} \rightarrow Y_A$ is the selector output function.

The output function λ^{Sel} acts on the acceptor outputs in following manner: Let $S_a = \lambda^A ((s, t), Process)$ be the current output and:

$$S_{ar} = \{(q, x, (w, i), (v, j)) \in S_a \mid x = r\} \text{ for } r \in R$$

denote the set of operations processed by the robot r which are ready to process. The function λ^{Sel} is defined in two steps:

Internal Selection Rule: Collision Prevention

From the set of operations S_a choose a subset for which robot actions do not result in a collision, i.e.:

$$S_a/_{IntSel} = \lambda^{Sel} (\lambda^A ((s, t), Process), Rule\_Int) = \cup \{S_{ar} \mid r \in Robot\_Activ\}$$

where *Robot_Activ* denotes a set of robots whose simultaneous actions do not lead to collisions. The set is created as follows:

$$r \in Robot\_Activ \Leftrightarrow S_{ar} \neq \emptyset \wedge \neg (<r' \in Robot\_Active) (r \neq r' \wedge Srv\_Sp(r) \cap Srv\_Sp(r') \neq \emptyset)$$

By this definition, more than one set of active robots can be generated. From all possible elements in *Robot_Activ*, we select one that has the greatest number of elements. This ensures that a maximum number of operations will be executed simultaneously.

External Selection Rule: Priority of Operations

For each set S_{ar}, where $r \in Robot\_Activ$, choose only one operation which has the highest priority, i.e.:

$$S_a/_{ExtSel} = \lambda^{Sel} (S_a/_{IntSel}, Rule\_Ext) = \{oper_{ar} \mid r \in Robot\_Activ\}$$

where Priority($oper_{ar}$) = max{Priority(x) | $x \in S_{ar}$}.

This rule represents an external selection strategy which applies to the set of all operations waiting to execute. For example, the rule can be formulated as "*select the operation with maximum/minimum index in the sequence Process.*" A simulator should be able to test different external strategies.

As the selector's output is the set S_a which contains only one element for each robot from *Robot_Activ*, i.e., $S_{ar}/ExtSel = \{oper_{ar}\}$.

To model a robot, we use a *generator*. The generator receives the outputs of the acceptor and selector in order to determine its state. The model of a robot, r, is defined by the following DEVS:

$$Robot_r = (S(r), \delta^r, \text{ta}^r, \{Z^{rd}\}) \tag{12}$$

The DEVS-model of each robot contains the state set $S(r) = S_r \times POSITIONS \times HS$, where:

- $S_r \subset 2^{Y_A}$ is the set of subsets of the acceptor output such that $S_r \subset 2^{U(r)}$, where $U(r) = \{(q, x, (w, i), (v, j)) \in \lambda^A((s, t), Process) \mid x = r\}$,
- *POSITIONS* is the set of positions and orientations of the robot's effector-end in the base-Cartesian space (effector's frames),
- *HS* is the set of states of the effector, i.e., *HS* = {Empty, Holding}

The internal transition function:

$$\delta^r : S(r) \rightarrow S(r)$$

is defined as follows: Let $s(r) = (s_r, \text{k–}position, \text{Empty})$. Then:

$$\delta^r(s_r) = \begin{cases} s(r) & \text{if } U(r) = \emptyset \\ (s_r \cup s_a, \text{i–pos. of } w, \text{Holding}) & \text{if } \lambda^{Sel}(\lambda^A(s,t), Process)) = \\ & = \{(o_k, r, (w, i), (v, j))\} = s_a \end{cases}$$

Let $s(r) = (s_r, \text{i-}position, \text{Holding})$. Then:

$$\delta r(s(r)) = (s_r - s_a, \text{j–pos. of } v, \text{Empty})$$

where $\lambda^{Sel}(\lambda^A((s, t), Process)) = \{(q, r, (w, i), (v, j)\} = s_a$.

Notice that an internal transition can be easily represented by a sequence of commands in TORPL [22], for example:

```
MOVE EMPTY FROM frame_k-pos
   TO frame_i-pos              [of buffer of device w]
PICKUP part AT
   frame_i-position            [of buffer of device w]
MOVE HOLDING FROM frame_i-pos
   TO frame_j-pos              [of buffer of device v]
PLACE part ON
   frame_j-position            [of buffer of device v]
```

The time advance functions determine (a) the sum of the time of motion to position i and the time of the pickup operation, $\tau^r_{motion} + \tau^r_{pick}$, and (b) the sum of motion time from the position i to the position j and the time of the place operation on the v^{th} workstation, $\tau^r_{motion} + \tau^r_{place}$, for Empty and Holding states, respectively.

The last component of $Robot_r$ is the set of functions which generate external events for workstations and stores.

$$\{Z_{rd}\} = \{Z_{rd} : S(r) \rightarrow X_d \mid d \in Device(r)\}$$

where $Device(r) = \{\alpha_1(o_i) \cup \alpha_2(o_i) \mid \beta(o_i, o_i + 1) = r \wedge o_i \in Process\}$ is the set of devices serviced by robot r in process *Process*.

The function Z_{rd} generates external events for the model of the workstation d. More specifically, this function is defined as follows:

For $s(r) = (s_r, k\text{–}position, \text{Empty})$:

$$Z_{rd}(s(r)) = \begin{cases} e^0 & \text{if } s_r = \emptyset \vee d \neq w \\ e^2_d(q, i) & \text{if } d = w \end{cases}$$

and for $s(r) = (s_r, \text{i–}position, \text{Holding})$:

$$Z_{rd}(s(r)) = \begin{cases} e^0 & \text{if } d \neq v \\ e^1_d(q, j) & \text{if } d = v \end{cases}$$

where $\{(q, r, (w, i), (v, j))\} = \lambda^{Sel}(\lambda^A((s, t), Process))$

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

This completes the cell's discrete-event model specification. The structure of the event-based model is shown in Figure 3.

5. An Illustrative Example

In this section we introduce an example of a simple task to illustrate the modeling concepts discussed so far in a real-world manufacturing context. First, we briefly summarize the notions of experimental frame and motion planning as they support the solution to our example.

5.1 Experimental Frame of Cell Model

We separate the model description from a simulation experiment under which the model is observed. A set of circumstances under which a model is experimented with is called an experimental frame. Zeigler [20] has shown that an experimental frame can be 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 forms of the fundamental plan *Process* and selection strategy λ^{Sel} determine the cell's control law. Recall that the sequence *Process* is an external data stream of the cell-model acceptor and that the function λ^{Sel} provides input parameters for the cell-model generator.

The variants of *Process* obtained from the process planner and those given by the selection rule should be tested by a simulator. The acceptor and selector produce elementary actions of robots servicing selected technological operations. The interpretation of these actions is carried out using a motion programming approach in which detailed paths and trajectories, gross and fine motion, grasping and sensing are specified. Variant interpretations of the motion commands result in different realizations of a task. To simulate, the system must have the knowledge of how individual robot actions are carried out in the process. This knowledge must be available in order to verify all possible sequences of technological operations from the set *Processes*. The most important parameters are the time τ^o_d it takes to complete an operation o and the time the robot requires to service a workstation. The time τ^o_d depends on the type of machine on which the operation o is being executed. 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), τ^{motion}_r, depend on the geometry of the workscene and the cost function of the robot's motion. This 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 system.

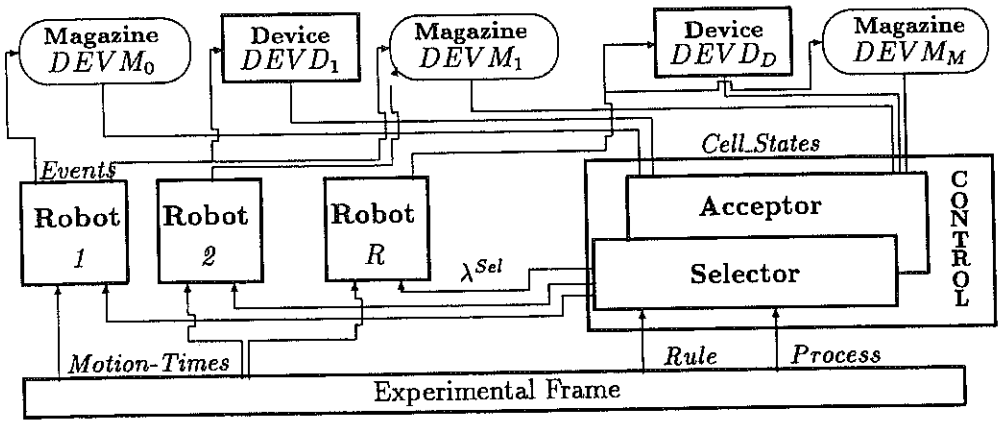

Figure 3. Structure of discrete-event model of a workcell

5.2 *Motion Planning*

To create all valid interpretations of robot commands, a motion planner is used for each individual robot action. It creates variants of collision-free time-trajectories of a manipulator. Such a planner uses robot-dependent planning techniques. The variants of motion interpretation obtained from the motion planner are tested by an event-based simulator.

The motion trajectory planning process is decomposed into two subproblems: (a) planning of collision-free geometric track of motion, and (b) planning of the motion dynamics along the computed track.

The path planner determines the collision-free track of the motion from the initial to the final effector location (defined by the acceptor and selector of the cell's model) based on the geometric and kinematic description of the robot and its environment. This problem has been addressed in various ways and is widely reported in literature [14, 25, 26, 27].

The optimal speed and acceleration of moves along the computed track are computed by the trajectory planner. The trajectory planner receives the geometrical tracks as input and determines the 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 [29, 30, 31].

Hence we obtain the optimal trajectory and the time of manipulator transfer moves. The time trajectories of motions are the basis for computing the times of each elementary action. For each motion command, we can change the geometry of movement or change the motion dynamics by selecting criteria for optimal trajectory planning. Variant interpretations obtained from the motion planning allow us to test and select the control law *Process* which minimizes the makespan. To calculate the makespan for different sequences of operations and different selection rules, we use the DEVS simulator. The structure of the experimental frame is shown in Figure 4.

5.3 *Technological Task*

Let us consider a technological task $Task = (O, <, \alpha)$, which consists of the following operations:

$$O = \{o_1, o_2, o_3, o_4,\}$$

where o_1 denotes **mill_1** operation, o_2 denotes **turn_1** operation, o_3 denotes **mill_2** operation and o_4 denotes **mill_3** operation. The precedence relation $<$ is shown in Figure 5.

The workcell on which this task is carried out consists of two CNC-millers d_1, $d_3 \in D$, and CNC-lathe $d_2 \in D$, three

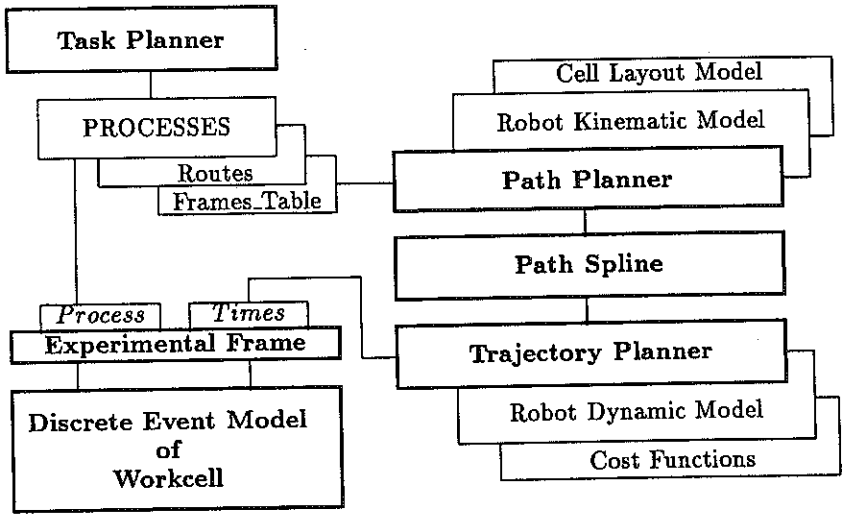

Figure 4. Structure of experimental frame

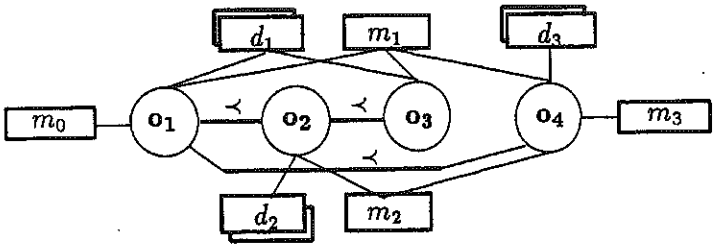

Figure 5. Precedence and assignment relations

production stores $m_1, m_2, m_3 \in M$, two robots $r_1, r_2 \in R$ and one feeder conveyor m_0. The store m_3 is also an output-conveyer. The capacity of all devices is equal to one ($C(d) = 1$). The geometrical model of the workcell is illustrated in Figure 6.

The following devices are assigned to every operation o_i:

$$\alpha = \begin{cases} o_1 \rightarrow (\{d_1\}, \{m_0, m_1\}) \\ o_2 \rightarrow (\{d_2\}, \{m_2\}) \\ o_3 \rightarrow (\{d_1\}, \{m_1\}) \\ o_4 \rightarrow (\{d_3\}, \{m_1, m_2, m_3\}) \end{cases}$$

The robots can service the following devices:

$$\text{Range}(r_1) = \{d_1, d_2, m_0, m_1, m_2\}$$
$$\text{Range}(r_2) = \{d_3, m_1, m_2, m_3\}$$

The allocation relation α is illustrated in Figure 5.

It is easy to observe that this task can be realized by any one of the following three technological processes, namely:

$$Process_1 = (o_1, o_2, o_3, o_4),$$
$$Process_2 = (o_1, o_4, o_2, o_3),$$
$$Process_3 = (o_1, o_2, o_4, o_3).$$

The production routes for each $Process_i$ are shown in Figure 7. Moreover, Figure 7 depicts the minimal production routes and their zone decompositions.

The task planner selects $Process_1$ with the value of function $v(\mathbf{p}_{\min}^1) = 2$ (for all other processes, the function v is equal to 3). $Process_1$ is chosen by the Task Planner to determine the operational control law of the workcell.

The control program is created on the second layer of the control synthesizer, where each movement command is planned and tested. One of the planned collision-free paths of the robot motion is shown in Figure 8. This path realizes a translocation of a part from lathe d_2 to miller d_1. Figure 8 also depicts the optimal joint velocities of robot movements along this path for two quality criteria, i.e., minimum-time and minimum-energy criterion. The results of motion planning are shown in the Table 3.

The table shows the times and energies of each motion in two variants, namely the minimum-time and minimum-energy motion. The set of time-trajectories for each motion command of the control program completes an applicable plan of the workcell's operational control. The simulation results of the workcell action model are presented in Figure 9. The productivity of the workcell is equal to 12.9 parts per hour when the minimum-time motion is used and 10.92 parts per hour in the minimum-energy case. In the first case, the unit-cost of production is equal to 1641 Joules per part and in the second case only 598 Joules per part.

6. Concluding Remarks

A comprehensive framework for design of an intelligent cell-controller requires integration of several layers of support methods and tools. We have proposed an architecture that facilitates an automatic generation of different plans of sequencing operations, synthesis of action plan for robots servicing the devices, synthesis of the workcell's simulation model, and verification of control variants based on simulation of the overall cell's 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 CAM/CIM is becoming increasingly obvious due to a number of reasons: most

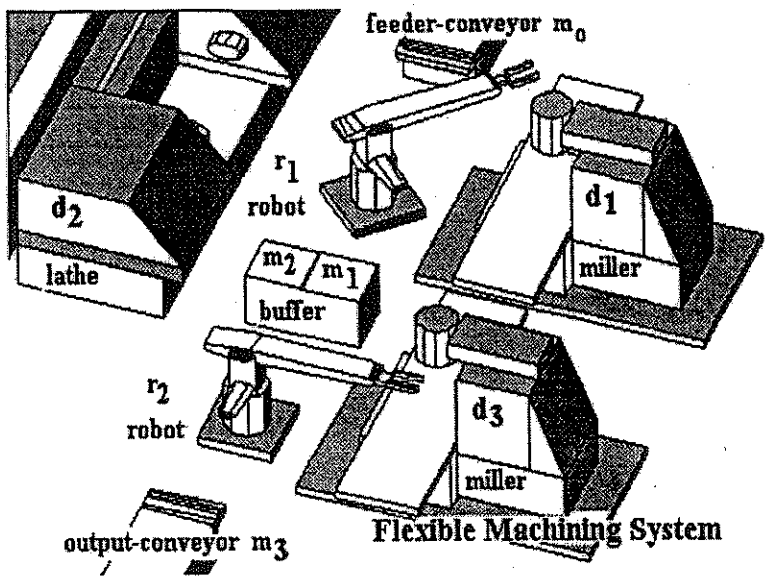

Figure 6. Geometrical model of the workcell

Table 3. Time and energy of motions [Time [Sec] / Energy[Joule]]

	m_0	d_1	m_1	d_2	m_2	d_3	m_3	
m_0		[1/214] [2.3/81]						Min. Time Min. Ener.
d_1			[0.7/112] [1.4/37]	[1.4/255] [3.1/90]	[0.8/112] [1.5/38]			Min. Time Min. Ener.
m_1	[1.8/346] [4.1/118]			[1.3/226] [2.9/72]		[0.9/175] [2.3/48]		Min. Time Min. Ener.
d_2	[3.2/634] [6.8/218]	[1.4/255] [3.1/90]			[1.2/225] [2.9/71]			Min. Time Min. Ener.
m_2	[1.9/345] [4.2/117]	[1.8/111] [1.5/37]						Min. Time Min. Ener.
d_3							[1.1/173] [2.9/76]	Min. Time Min. Ener.
m_3			[2.8/508] [5/204]					Min. Time Min. Ener.

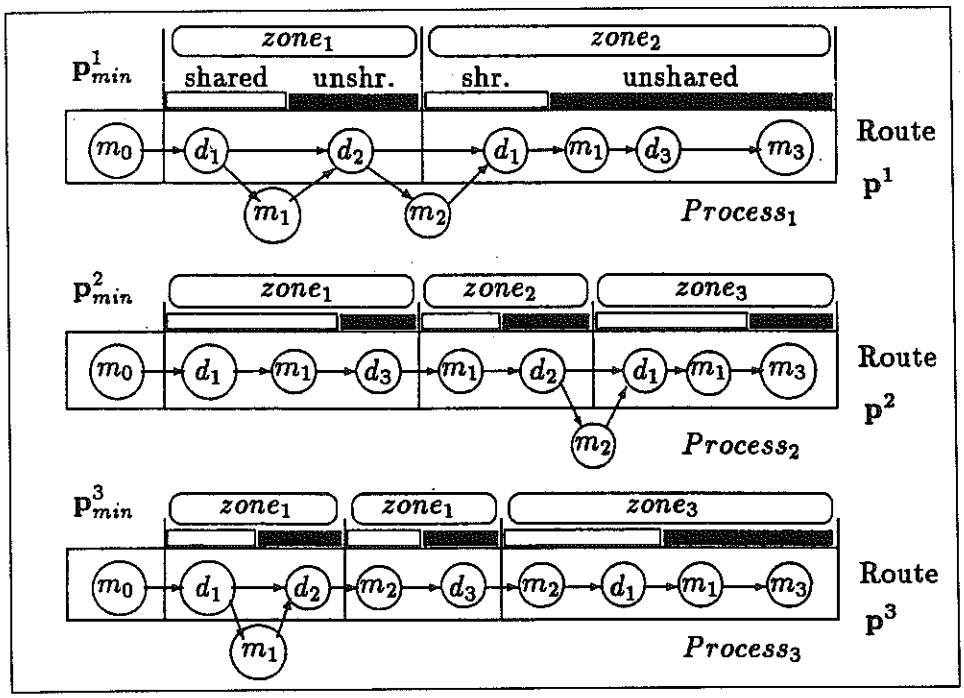

Figure 7. Production routes and zones

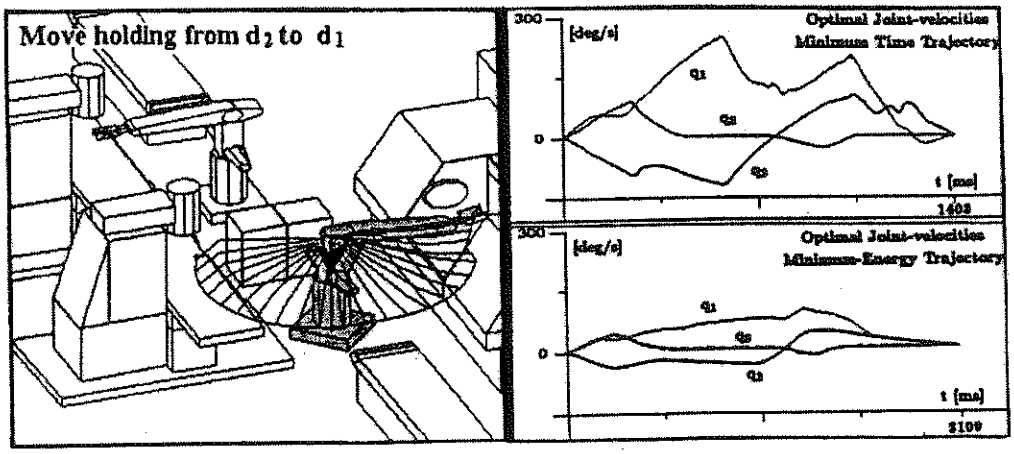

Figure 8. Optimal trajectory of robot r_1 motion

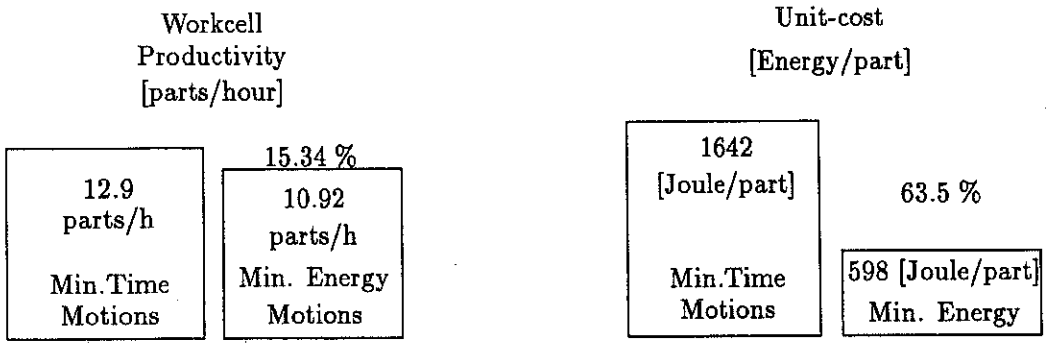

Figure 9. Productivity and unit cost of workcell

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 a workcell in order to evaluate its efficiency. They cannot emulate a programming language that would use a simulation model.

7. References

[1] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*, San Francisco, Morgan Kaufmann, 1988.

[2] Sacerdot, E.D. "Planning in a Hierarchy of Abstraction Spaces." *Artificial Intelligence*, Vol. 15, No. 2, 1981.

[3] McDermott, D. "A Temporal Logic for Reasoning about Processes and Plans." *Cognitive Science*, Vol. 6, 1982.

[4] Lifschitz, V. "On the Semantics of STRIPS." *Reasoning About Actions and Plans*, M. Georgeff, editor, Morgan Kaufmann, San Francisco, 1987.

[5] Saridis, G.N. "Intelligent Robotic Control." *IEEE Transactions of Autom. Contr.*, Vol. AC-28, No. 5, 1983.

[6] Saridis, G.N. "Analytical Formulation of the Principle of Increasing Precision with Decreasing Intelligence for Intelligent Machines." *Automatica*, 1989.

[7] Meystel, A. "Intelligent Control in Robotics." *Journal of Robotic Systems*, Vol. 5, No. 5, 1988.

[8] Buzacott, J.A. "Modelling Manufacturing Systems." *Robotics and Comp. Integr. Manufac.*,Vol. 2, No. 1, 1985.

[9] Jones, A.T., McLean, C.R. "A Proposed Hierarchical Control Model for Automated Manufacturing Systems." *Journal of Manufacturing Systems*, Vol. 5, No. 1, pp 15-25, 1986.

[10] Maimon, O. "Real-time Operational Control of Flexible Manufacturing System." *Journal of Manufacturing Systems*, Vol. 6, No. 2, pp 125-136, 1987.

[11] Coffman, E.G., Elphick, M., Shoshani, A. "System Deadlock." *Computing Surveys*, Vol. 3, No. 2, pp 67-78, 1971.

[12] Banaszak, Z., Roszkowska, E. "Deadlock Avoidance in Concurrent Processes." *Foundations of Control*, Vol. 18, No. 1–2, pp 3-17, 1988.

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

[14] Ranky, P.G., Ho, C.Y. *Robot Modeling. Control and Applications with Software*, Springer Verlag, 1985.

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

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

[17] Homem De Mello, L.S., Sanderson, A.C. "AND/OR Graph Representation of Assembly Plans." *IEEE Transactions on Robotics and Automation*, Vol. 6, No. 2, pp 188-199, 1990.

[18] Sanderson, A.C., Homem De Mello, L.S., Zhang, H. "Assembly Sequence Planning." *AI Magazine*, Vol. 11, No. 1, Spring 1990.

[19] Rozenblit, J.W., Zeigler, B.P. "Design and Modeling Concepts." *International Encyclopedia of Robotics, Applications and Automation*, R. Dorf, editor, John Wiley and Sons, New York, pp 308-322, 1988.

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

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

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

[23] Speed, R. "Off-line Programming of Industrial Robots." *Proceedings of ISIR 87*, pp 2110-2123, 1987.

[24] Jacak, W., Rozenblit, J.W. "Automatic Simulation of a Robot Program for a Sequential Manufacturing Process." *Robotica*, Vol. 10, pp 45-56, 1992.

[25] Lozano-Perez, T. "Task-Level Planning of Pick-and-Place Robot Motions." *IEEE Trans. on Computer*, Vol. 38, No. 3, pp 21-29, 1989.

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

[27] Jacak, W. "A Discrete Kinematic Model of Robot in the Cartesian Space." *IEEE Trans. on Robotics and Automation*, Vol. 5, No. 4, pp 435-446, 1989.

[28] Jacak, W. "Discrete Kinematic Modeling Techniques in Cartesian Space for Robotic System." *Advances in Control and Dynamics Systems*, C.T. Leondes, editor, Academic Press, 1991.

[29] Shin, K., McKay, N. "A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators." *IEEE Trans. On Automatic Control*, Vol. 31, No. 6, pp 491-500, 1986.

[30] Shin, K., McKay, N. "Minimum Time Control of Robotic Manipulator with Geometric Path Constraints." *IEEE Trans. On Automatic Control*, Vol. 30, No. 6, pp 531-541, 1985.

[31] Jacak, W., Duleba, I., Rogalinski, P. "A Graph-Searching Approach to Trajectory Planning of Robot's Manipulator." *Robotica*, (in print), 1992.