Rapid #: -4495130

1

Odyssey
IP: 150.135.238.6/ILL

Status	Rapid Code	Branch Name	Start Date
Pending	AFU	Main Library	6/7/2011 8:57:26 PM

CALL #:	**QA75.5 .L42**
LOCATION:	**AFU :: Main Library :: MAIN**
TYPE:	Article CC:CCL
JOURNAL TITLE:	Lecture notes in computer science
USER JOURNAL TITLE:	Lecture Notes in Computer Science
AFU CATALOG TITLE:	Lecture notes in computer science
ARTICLE TITLE:	CAST Tools for Intelligent Control in Manufacturing Automation
ARTICLE AUTHOR:	
VOLUME:	
ISSUE:	763
MONTH:	
YEAR:	1994
PAGES:	203-219
ISSN:	0302-9743
OCLC #:	AFU OCLC #: 3719235
CROSS REFERENCE ID:	[TN:951125][ODYSSEY:150.135.238.6/ILL]
VERIFIED:	

BORROWER:	**AZU :: Main Library**
PATRON:	**Liana Son Suantak**
PATRON ID:	lianason
PATRON ADDRESS:	
PATRON PHONE:	
PATRON FAX:	
PATRON E-MAIL:	
PATRON DEPT:	
PATRON STATUS:	
PATRON NOTES:	

RAPIDILL

CAST Tools for Intelligent Control in Manufacturing Automation

Witold Jacak[1] and Jerzy Rozenblit[2]

[1] Institute of Systems Science
Johannes Kepler University Linz
A-4040 Linz, Austria
[2] Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721, U.S.A.

1 Introduction

In recent years, the use of programmable and flexible systems has enabled partial or complete automation of machining and assembly of products. Such the cellular flexible manufacturing systems (FMS) are data-intensive. They consist of three main components: a *production system, a material handling system,* and a *hierarchical computer assisted control system.*

A cellular manufacturing system is intelligent if it can *self-determine choices in its decisions* based upon the simulation of a needed solution in a virtual world or upon the experience gained in the past from both failures and successful solutions stored in the form of rules in the system's knowledge base. The intelligent manufacturing system discussed here is a *computer integrated cellular system* consisting of fully automated robotic cells. Planning and control within a cell is carried out off-line and on-line by the hierarchical controller which itself is regarded as an integral part of the cell. We call such a cell **Computer Assisted Workcell** (CAW).

Given a technological task, an intelligent Computer Assisted Workcell should be able to determine control algorithms so that: a) a task is realized, b) deadlocks are avoided, c) the flow time is minimal, d) the work-in-process is minimal, e) geometric constraints are satisfied, and f) collisions are avoided.

To synthesize an autonomous, or semi-autonomous, Computer Assisted Workcell we use artificial intelligence and general system theory concepts, methods, and tools such that as *hierarchical decomposition of control problems, hierarchy of models specification, discrete and continuous simulation* from system theory ideas and *action planning methods, model's state-graph searching methods* from artificial intelligence concepts.

The control laws which govern the operation of CAW are structured hierarchically. We distinguish three basic levels: a) *the workstation (execution) level,* b) *the cell (coordination) level,* and c) *the organization level.*

• The organizer accepts and interprets related feedback from the lower levels, defines the strategy of task sequencing to be executed in real-time, and processes large amounts of knowledge with little or no precision. Its functions are: reasoning, decision making, learning, feedback, and long term memory exchange.

• The coordination level defines part routing in logical and geometric aspects and coordinates the activities of workstations and robots. The robots coordinate the activities of the equipment in the workstation. This level is concerned with the formulation of the actual control task to be executed by the lowest level.
• The execution level consists of device controllers. It executes the programs generated by the coordinator.

Consider the CAW with a hierarchical structure shown in Figure 1. It is composed of three interactive levels of organization, coordination and execution, modeled with the aid the theory of intelligent system [3]. All planning and decision making actions are performed within the higher levels. In general, performance of such systems is improved through self-planning with different planning methods and through self-modification with learning algorithms and schemes interpreted as interactive procedures for the determination of the best possible cell action.

There are two major problems in synthesis of such complex control systems: The first depends on the *coordination and integration* at all levels in the system, from that of the cell, where a number of machines must cooperate, to that of the whole manufacturing workshop, where all cells must be coordinated. The second problem is that of *automatic programming* of the elements of the system. Thus we distinguish two major levels of the control problem: the *logical (operational)* and the *geometric control*.

The intelligent control of CAW is synthesized and executed in two phases, namely:
- **planning and off-line simulation phase** and
- **on-line simulation based monitoring and control phase.**

In the first phase, a hierarchical simulation model of a robotic workcell called **virtual cell** is created. Workcell components such as NC-machine tools, robots conveyors, etc., are modelled as elementary DEVS systems [13]. This type of simulation is used for verification and testing of different variants of task realizations (processes) obtained from the route planner. To simulate variants of a process, the system must have the knowledge of how individual robots actions are carried out. For detailed modeling of cell components continuous simulation and motion planning methods are used. The geometrical interpretations of cell-actions are obtained from the motion planner and are tested in a geometric cell-simulator. This allows us to select the optimal task realizations which establish logical control of the system.

In the second phase, a real-time discrete event simulator of CAW is used to generate a sequence of future events of the virtual cell in a given time-window. These events are compared with current states of the real cell and are used to predict motion commands of robots and to monitor the process flow. Since a CAW has to make many subjective decisions based on deterministic programming methods, knowledge bases and knowledge based decision support should be available as an advisory layer. Such knowledge bases are part of the so-called virtual workcell.

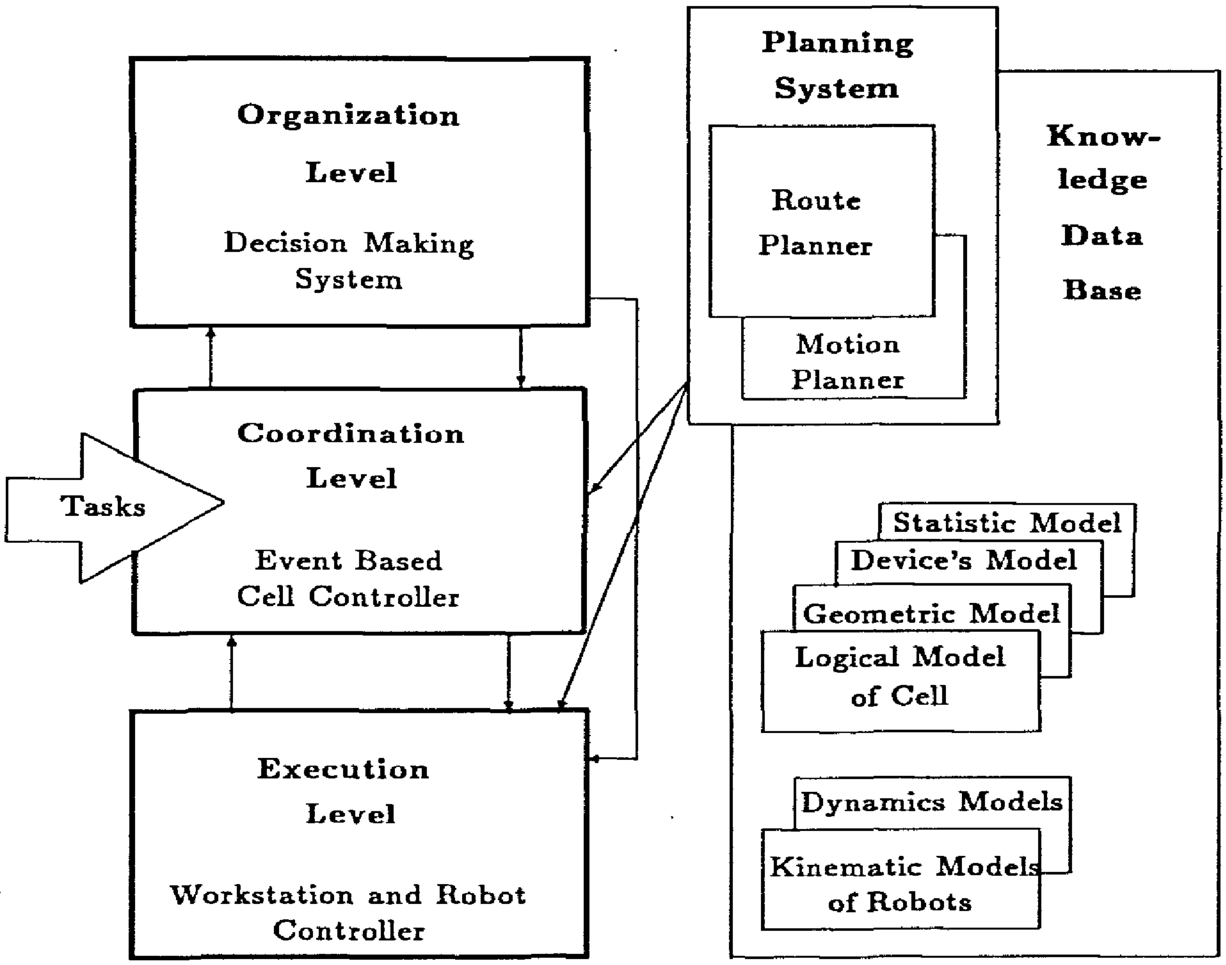

Fig. 1. Hierarchical structure of CAW control system

2 Virtual Workcell and Technological Task

A real cell is a fixed, physical group of machines D (or stores M), and robots R. A *virtual cell* is a formal representation (computer model) of a workcell. To synthesize CAW's control, we first specify the family of technological tasks realized in the cell.

2.1 Technological Task

The *technological task* realized by the robotic cell is represented by a triple:

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

where: O is a finite set of *technological operations* (machine, test, etc.) required to process the parts, $\prec \subset O \times O$ is the partial order (*precedence relation*) on the set O, and $\alpha \subset O \times (D \cup M)$ is a relation of device or store assignment.

The partial order represents an operational precedence i.e.; $q \prec o$ means that the operation q is to be completed before the operation o can begin. $(o, d) \in \alpha$ means that the operation o can be performed on the workstation d, and if $(o, m) \in \alpha$, then m is the production store from the set M where the parts can be stored after the operation o has been completed.
The technological task described above can be realized in a virtual cell. The virtual robotic cell has a hierarchical structure which comprises models. The models represent the knowledge about a real production environment.

2.2 Geometric Model of Workcell

The lower level of virtual cell representation describes the geometry of a virtual cell. Formally, the geometry of the cell is defined as follows [15]:

$$Cell_{Geometry} = (\mathrm{G}, \mathrm{H}) \tag{2}$$

The first component of the cell geometry description

$$\mathrm{G} = \{\mathcal{G}_d = (E_d, V_d) | d \in D \cup M\} \tag{3}$$

represents the set of geometric models of the cell's objects. E_d is the coordinate frame of object (device) d and V_d is the polyhedral approximation of the d-th object geometry in E_i.

$$\mathrm{H} = \{\mathcal{H}_d : \ E_d \rightarrow E_0 | d \in D \cup M\} \tag{4}$$

represents the cell layout as the set of transformations between an object's coordinate frames E_d and the base coordinate frame E_0 [18].
Consequently, a geometric model has two components: (1) workcell objects models and (2) workcell layout model.

Workcell Object Model: The geometry model of each object is created by using solid modeling [9]. Solid modeling incorporates the design and analysis of virtual objects created from primitives of solids stored in a geometric database. The complex virtual objects of a workcell V_d (such as technological devices, robots, auxiliary devices or static obstacles) are composed of solid primitives such as cuboid, pyramids, regular polyhedrons, prisms, and cylinders.

Workcell Layout The workcell's objects can be placed in a robot's workscene at any position and in any orientation. The virtual objects (devices, stores) are loaded from a library (model base) into the Cartesian base frame. They can be located anywhere in the cell using translation and rotation operations in the base coordinate frame [14].

2.3 Logical Model of Workcell

Based on a geometric model of the workcell, a logical structure of the cell must be created. The group of machines is divided into subgroups, called *machining centers* serviced by separate robots.

Parts are transferred between machines by the robots from set R, which service the cell. A robot $r \in R$ can service only those machines that are within its service space $Serv\_Sp(r) \subset E_0$ (E_0 - Cartesian base frame). The set of devices which lie in the r-th robot service space is denoted by $Group(r) \subset D \cup M$. More specifically, a device belongs to group serviced by robot r (i.e., $d \in Group(r)$) if all positions of its buffer lie in the service space of robot r. Consequently, the logical model of a workcell is represented by:

$$Cell_{Logic} = (D \cup M,\ R,\ \{Group(r) \mid r \in R\}) \tag{5}$$

Based on the sets $Group(r)$ and the description of the task, we can define the relation β which describes the transfer of parts after each technological operation of the task.

$$\beta \subset (O \times O) \times R \tag{6}$$

where: $((o_i, o_j), r) \in \beta) \Leftrightarrow (\{d_i, d_j\} \subset Group(r) \vee \{m_i, d_j\} \subset Group(r))$ and $\alpha(o_i) = \{d_i, m_i\}$ and $\alpha(o_j) = \{d_j, m_j\}$. In a special case β is a partial function; i.e.: $\beta : O \times O \rightarrow R$.
The virtual cell concept is the basis for designing the CAW planners.

3 Logical and Geometric Control Planners

The realization of a technological task depends on the sequencing of its operations. The execution sequence can be represented as a list of machines, called a *production route* (or *logical control of process*), along which parts flow during the manufacturing process. Each route determines a different topology of robot motion trajectories, different deadlock avoidance conditions, and a different job flow time. Therefore, route planning is a critical issue in all manufacturing problems.

3.1 Technological Route Planner

Searching for the most efficient route requires that we define route comparison criteria. One such criterion is that the sequence of operations and robot/machine actions related to it minimize the mean flow time of parts [6]. The flow time of every job is the sum of *processing* time, *waiting* time, and the time of inter-operational moves called *transfer* times.

The route planner should find an ordered execution sequence of L operations called a *sequential machining process*:

$$p = (o_1, o_2, ..., o_L) \tag{7}$$

with the following constraints:
(i) if for two operations o_i and o_j from $Task$, $o_i \prec o_j$, then $i < j$
(ii) 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 sequences of technological devices (called *resources*) required by successive operations from the list $\wp$ when they are being executed. This set, denoted by **P**, is called production routes. The route planning algorithm should take into account the conditions for deadlock avoidance. In order to formulate a quality criterion of process planning, we use the procedure of deadlock avoidance presented in [8], [7].

The quality criterion used to evaluate the technological route being planned is the probability of waiting for a resource when deadlock avoidance conditions are in force. The problem solved by the route planner is to find an ordered sequence of operations from the technological task which is feasible and which minimizes this quality criterion. This is a permutation problem with potentially more than one solution. To solve the planning problem under consideration, the route planner uses a backtracking graph search algorithm [14].

Each route from the output of the task planner determines a different logical sequence of robot and machine actions and a different input data for the geometric control of cell actions.

3.2 Motion Planner

The production route **p** for a machining task determines the parameters of the robot's movements and manipulations (such as initial and final positions) to carry out this task. The set of all robot's motions between devices and stores needed to carry out a given process is called a *geometric route* or *geometric control*.

First, based on the sequence of operations *Process* and its production route **p**, the positions table (Frames_Table) for all motions of each robot is created. The Frames_Table determines the initial and final geometric positions and orientations of the robot's effector for each robot movement. The robot's motion trajectory planning process is performed in two stages: a) planning of the geometric track of motion, and b) planning of the motion dynamics along a computed track.

The Collision-free Path Planner In order to apply fast methods of geometric path planning, the robot's kinematics model should facilitate a direct analysis of the robot's location with respect to objects (virtual devices) in its environment. Moreover, such a model should facilitate 3D graphic simulation of robot movements. One possible description of the manipulator's kinematics is a discrete dynamic system [16],[17],[18].

The discrete kinematics model of robots with n degrees of freedom has the following form:

$$Robot = (C, U, \delta) \tag{8}$$

where: C denote the set of robot configurations (manipulator states) in the Cartesian base frame; $U = \{-1, 0, +1\}^n$ is the discrete input signal set, and $\delta : C \times U \longrightarrow C$ is the one-step state transition function of a dynamic system of the form $c(k+1) = \delta(c(k), u(k))$ and is defined recursively [16],[18].

The complete formal explanation of the discrete model of robot kinematics is presented in [16],[18].

Motion Trajectory Planning The trajectory planner receives the geometric tracks as inputs and determines a time history of position, velocity, acceleration, and input torques which can be then fed to the trajectory tracker. In the trajectory planner, the robot is represented by the model of path parameterized manipulator dynamics [19]:

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

$$\dot{s} = \mu \qquad (10)$$

where pseudo-velocity μ is the time derivative of the parameter s.
To optimize the time trajectory of motion the following cost function is chosen:

$$I = \int_0^{s_{final}} \left(\frac{1}{\mu(s)}\, \lambda_1 \; + \; \lambda_2 \sum_{i=1}^{n} ||f_i||\right) ds \;\text{ and }\; \lambda_1 + \lambda_2 = 1. \qquad (11)$$

The trajectory planning problem is then reduced to cost minimization, subject to the dynamic model specification and the constraints of torque f. This is a classical dynamical optimization problem for a nonlinear system and a effective bidirectional search method is used [20], [21].

The Motion Planner generates the set of optimal trajectories of parts transfer movements. This set is establishes the geometric control laws of a CAW.

4 Discrete Event, Real-Time Cell Controller

The process planning is based on the description of task operations and their precedence relation. As a result of this stage, the fundamental plan of cell-actions, i.e., an ordered sequence of technological operations, is created.

In the second phase, the real time event-based simulator of CAW is synthesized. The simulator generates a sequence of future events of the virtual cell in a given time-window. These events are compared with current states of the real cell and are used to predict motion commands of robots and to monitor the process flow. The simulation is event oriented and is based on the DEVS system concept introduced by Zeigler [13]. The structure of CAW is shown in Figure 2.

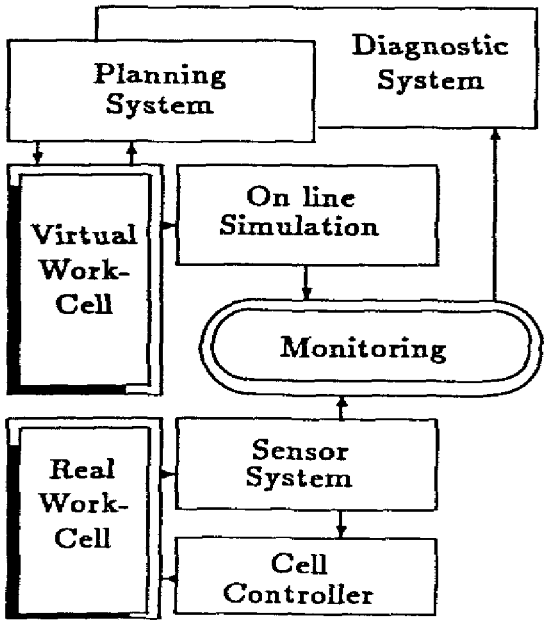

Fig. 2. Structure of Computer Assisted Workcell

4.1 Synchronized, Event-Based Model of Workstation

Each workstation $d \in D$ is modelled by two coupled atomic discrete event systems (DEVS) called active and passive models of a device, i.e., $DEVS_d = (Dev_d^A, Dev_d^P)$, where Dev_d^A is the active model and Dev_d^P is the passive model.

The active atomic DEVS performs the simulation process and a passive atomic DEVS represents the register of real states of the workstation. It acts as a synchronizer between the real and the simulated processes. This model is a modified version of the specification proposed by Zeigler and has following structure:

The active model of workstation:

$$Dev_d^A = (X_V(d), S_V(d), \delta_{int}^d, \delta_{ext}^d, \mathrm{ta}^d)$$

$X_V(d)$ is a set, the external input virtual event types, $S_V(d)$ is a set, the sequential virtual states, δ_{int}^d is a set of functions, the internal transition specification, δ_{ext}^d is a function, the external transition specification, ta^d is a function, the time advance function, with the following constraints:

(a) The total virtual event-set of the system specified by Dev_d^A is
$E_V(d) = \{(s,t) | s \in S_V(d), 0 \leq t \leq \mathrm{ta}^d(s)\}$;

(b) δ_{int} is a set of parameterized internal state-transition functions:
$\delta_{int} = \{\delta_{int}^u | u \in U\}$ and
$\delta_{int}^u : S_V \rightarrow S_V$;

(c) δ_{ext} is an external state-transition function:
$$\delta_{ext} : E_V \times X_V \rightarrow S_V;$$
(d) ta is a mapping from S_V to the non-negative reals with infinity:
$$\text{ta} : S_V \rightarrow R,$$

We now describe each component of an active model. Each workstation $d \in D$ can have a buffer. The capacity of this buffer is denoted by $C(d)$. If a workstation has no buffer, then $C(d) = 1$. Let NC(d) (NC program register) denote the set of operations performed on the workstation d, i.e., $\text{NC}(d) = \{o \in O | (o, d) \in \alpha\}$.

The virtual state set of workstation: The state set of d is defined by $S_V(d) = SD_d \times SB_d$, where SD_d denotes the state set of the machine and SB_d denotes the state set of its buffer. The state set SD_d is defined as $SD_d = S_{ready} \cup S_{busy} \cup S_{done}$ where:

- $S_{ready} = \{ready\}$ signifies that machine is free
- $S_{busy} = \{(busy, q) | q \in \text{NC}(d)\}$ and $(busy, q)$ signifies that machine is busy processing q-operation
- $S_{done} = \{(done, q) | q \in \text{NC}(d)\}$ and $(done, q)$ signifies that machine has completed q-operation and is not free

The state set of a workstation's buffer SB_d is specified as
$$SB_d = (O \times \{0, 1, \#\})^{C(d)}.$$
Let $C(d) = K$ and $b_d = (b_i | i = 1, ..., K) \in SB_d$, then
$$b_i = \begin{cases} (0,0) & \Leftrightarrow \text{i -- th position of the buffer is free} \\ (0,\#) & \Leftrightarrow \text{i -- th position of the buffer is reserved for a part being currently} \\ & \quad \text{processed} \\ (q,0) & \Leftrightarrow \text{i -- th position of the buffer is occupied by a part before operation q} \\ (q,1) & \Leftrightarrow \text{i -- th position of the buffer is occupied by a part after operation q} \end{cases}$$
We assume that the i-th position denotes a location at which a part is placed in the buffer.

Model of Workstation's Controller: Given the virtual state set, we define the internal state-transition functions δ_{int} to model the workstation. They are parameterized by an external parameter u which is loaded into the workstation from a higher level of control called the workcell management system. The parameter $u \in U$ is the operation's choice function. It represents the priority strategy of workstation, i.e.,
$$u : 2^{\text{NC}(d)} \rightarrow \{o\} \subset \text{NC}(d)$$
and $u(\emptyset) = \emptyset$, $u(\{o\}) = \{o\}$.

The choice function u defines a priority rule under which the operations are chosen from the device's buffer.

Let $s(d) = (s_d, (b_1, b_2, ..., b_K)) = (s_d, b) \in S_V(d)$ and
$$Wait(d) = \{q | ((\exists b_j)((o, 0) = b_j\}$$

be the set of operations waiting to be processed.

Then the internal transition function

$$\delta^u_{int} : S_V(d) \rightarrow S_V(d)$$

is specified as follows:

$$\delta^u_{int}(s(d)) = \begin{cases} ((done, q), (b_1, b_2, ..., b_K)) & \Leftrightarrow s_d = (busy, q) \\ ((busy, q), (b_1, ., b_{i-1}, (0, \#), b_{i+1}, ., b_K)) & \Leftrightarrow s_d = ready \wedge \\ & u(Wait(d)) = \{q\} \ \wedge \ (q, 0) = b_i \\ (ready, (b_1, b_2, ..., b_K)) & \Leftrightarrow s_d = ready \ \wedge \ Wait(d) = \emptyset \\ (ready, (b_1, ., b_{i-1}, (q, 1), b_{i+1}, ., b_K)) & \Leftrightarrow s_d = (done, q) \wedge b_i = (0, \#) \end{cases}$$

Corollary 1. *It is clear that if $s_d = (done, q) \ \vee \ s_d = (busy, q)$ than $(\exists! i)(b_i = (0, \#)$ because the workstation can not process two parts simultaneously.*

Model of Workstation-Robot Interaction: The interaction between the robots and workstation is modelled by the external state transition function.

The set of external virtual events for the workstation's model Dev^A_d is defined as follows:

$$X(d) = \{e^1_d(q, i), e^2_d(q, i), e^0 | q \in NC(d) \wedge i = 1, ..., K\}$$

where:

$e^1_d(q, i)$ = 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

$e^2_d(q, i)$ = 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^0 = DO NOTHING (empty event)

The external transition function δ^d_{ext} for each workstation d is defined below. Let $s(d) = (s_d, (b_i | i = 1, .., K)) \in S_V(d)$ then

$\delta_{ext}((s(d), t), e^0) = s(d)$

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

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

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

Time Model The time advance function for Dev^A_d determines the time needed to process a part on the d-th workstation. It is defined as follows:

Let $s(d) = (s_d, (b_i | i = 1, .., K))$. Then

$$\mathrm{ta}^d(s(d)) = \begin{cases} \tau_{process}(q) & \Leftrightarrow s_d = (busy, q) \\ \tau_{load} + \tau_{setup}(q) & \Leftrightarrow s_d = ready \ \wedge \ u(Wait(d)) = \{q\} \\ \tau_{unload} & \Leftrightarrow s_d = (done, q) \\ \infty & \text{otherwise} \end{cases}$$

$\tau_{process}(q)$ denotes the tooling/assembly time of operation q for the workstation d. $\tau_{load} + \tau_{setup}(q)$, τ_{unload} denote the loading, setup, and unloading times for d, respectively.

At time moment t, let the workstation be in the active state s which has begun at time moment t_o. After this time, the workstation transfers its state from s into $\delta^u_{int}(s)$, which will be active in the next time interval $[t_o + \mathrm{ta}(s), t_o + \mathrm{ta}(s) + \mathrm{ta}(\delta^u_{int}(s))]$.

Corollary 2. *It is easy to observe that the external events cannot change the advance time for each active state. Let $x = (e, t)$ be an external event different from e^0 which occurs at time moment $t \in [t_o, t_o + \mathrm{ta}(s)]$. The workstation changes its state to state $s' = \delta_{ext}((s,t), e)$ but the advance time of the new state s' is equal to $ta(s') = t_o + \mathrm{ta}(s) - t$.*

The passive model of workstation: The passive model is represented by a finite state machine (FSM) :

$$Dev^P_d = (X_R(d), Q_R(d), \varphi^d_{ext}, \lambda^d)$$

where:

$X_R(d)$ is a set, the external input real event types
$Q_R(d)$ is a set, the sequential real states
φ^d_{ext} is a function, the external real-state transition specification
λ^d is a function, the updating function

with the following constraints:

(a) φ^d_{ext} is a real-state transition function (one-step-transition function):
$$\varphi^d_{ext} : Q_R(d) \times X_R(d) \rightarrow Q_R(d);$$
(b) λ^d is a updating function (output function):
$$\lambda : UP \times Q_R(d) \times S_V(d) \rightarrow S_V(d);$$
where: $UP = \{0, 1\}$ and 0 denotes that the updating process is stopped and 1 denotes that updating should take place.

The interaction between external sensors and the workstation is modelled by the state transition function φ_{ext}. The external sensor system generates real events, which can be used to synchronize the simulated technological process with the real technological process.

The set of real events for the workstation's model Dev^P_d can be defined as follows:

$$X_R(d) = \{re^1_d(q,i), re^2_d(q,i), re^3_d(q,i), re^4_d(q,i) | q \in \mathrm{NC}(d) \wedge i = 1, ..., K\}$$

where:

$re^1_d(q,i)$ - signifies that a part before q operation is placed on i-th position of d-workstation's buffer
$re^2_d(q,i)$ - signifies that a part after q operation is removed from i-th position of d-workstation's buffer,
$re^3_d(q,i)$ - signifies that a part before q operation is loaded into machine and processing is began,

$re^4_d(q,i)$ - signifies that machine has completed the q operation, machine is unloaded and part is placed on i-th position of d-workstation's buffer.

We assume that the state set $Q_R(d)$ of the passive model is the same as that of the state set of the active one, i.e., $Q_R(d) = S_V(d)$.
Then, the state transition function φ^d_{ext} for the workstation d can be defined in a similar manner as the external transition function δ_{ext}. This transition function reflects real state changes of the workstation. The output function λ produces the real state in the active model of workstation when updating is required, i.e.,

$$\lambda(q,s,1) = q \in S_V(d) \quad \& \quad \lambda(q,s,0) = s \in S_V(d).$$

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

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

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

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

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

The internal transition δ^m_{int} is an identity function which can be omitted. The external transition function δ^m_{ext} for each store m can be defined in the same manner as the function δ^d_{ext} . The time advance function for Dev_m is equal to ∞.

The model of robot: Each robot $r \in R$ is modelled by two coupled systems, again called active and passive models.

$$REVS_r = (Rob^A_r, Reg^P_r)$$

where Rob^A_r is the active discrete event model and Reg^P_r is the passive model. The active model is used for simulation while the passive one represents the register of the robot's real states. It acts as a synchronizer between the real and simulated processes.

The virtual workstations and robots (DEVS models) together with the real devices and robots represent the execution level of CAW control system.

The model of cell controller: The activation of each machine is caused by an external event generated by the model of a robot. The events generated by the cell's robots depend on the states of the workstations d_i and a given fundamental plan $p \in Processes$. Thus, we define a system (called an *acceptor*) which observes the states of each workstation. The acceptor is defined as the following discrete event system [13]:

$$A = (X_A, Processes, Y_A, \lambda_A) \tag{12}$$

where:

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

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

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. The output of the acceptor can be an empty set or it can contain 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 discrete event system called the *cell controller*. First, this system eliminates all unrealizable operations from the set operations given as the acceptor's output function. Thus, a set of *executable operations* is created.

Then, from such a set, operations with the highest priority are selected. The priorities are established by the *workcell organization layer* as shown in Figure 3.

Each function δ_{int} is parameterized by an external parameter g which is loaded into the cell controller from the workcell organizer. The parameter $g \in Strategies$ is the operation's choice function, and represents the priority strategy of CAW, i.e.,

$$g : 2^{OP} \rightarrow \Re$$

The choice function g defines a type of priority rule under which the operations will be chosen for processing.

5 Real Time Control and Monitoring

The external events generated sequentially by the cell controller and robots activate the workcell's devices and coordinate the transfer's actions.

The discrete event model of workcell generates a sequence of future events of the virtual cell in a given time-window which we called a *trace*. A trace of the behaviour of a device d is a finite sequence of events (state changes) in which the process has been engaged up to some moment in time. A trace in a time window is denoted as a sequence of pairs (state,time) , i.e.,

$$tr_{[t_0,t_0+T]} =< (s_1,t_1),(s_2,t_2),(s_3,t_3),..(s_n,t_n) >$$

where $t_1 \geq t_0 \wedge t_n \leq t_0 + T$.

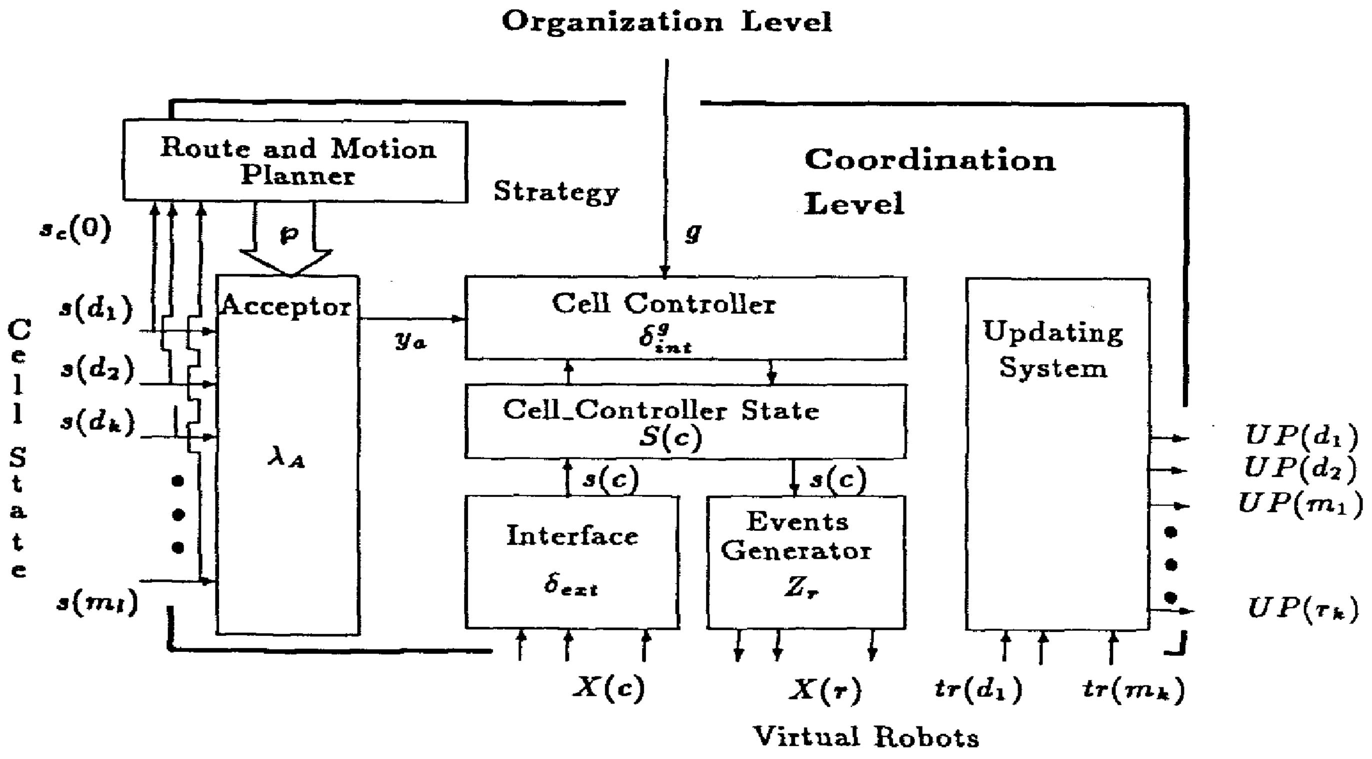

Fig. 3. *The coordination level of CAW*

The events from a trace are compared with current states of the real cell and are used to predict motion commands of robots and to monitor the process flow.

The simulation model is modified if the states of the real cell change and the current real states are introduced to the model.

5.1 Monitoring

Let $tr_T(d)$ be a trace of virtual events of device d on time-interval $T = [t_o, t_o + T]$ where t_o is the moment of the last updating:

$$tr_T(d) = < (s_o, t_o), (s_1, t_1)...., (s_k, t_k) >$$

where $t_k \leq t_o + T$ and s_i is the virtual state of DEVS model of d-device.
The monitoring process is performed as follows:
Let $q(t)$ be the current real state of d-device, modelled by passive Dev^P_d automaton and $t \geq t_o$.

If
$q(t) \neq s_j$ for $t \in [t_j - \tau_0, t_j + \tau_0]$

```
then
call diagnosis
else
call updating
```

Updating: Let $q_d(t)$ be the current real state of d-device in time t, registered by passive Dev_d^P automaton and $s_d(t') \in S_V(d)$ be a virtual state of d-device simulator in time t' . If $q(t) = s(t')$ and $t' \neq t$ but $t \in [t' - \tau_0, t' + \tau_0]$ where τ_0 is tolerance time-window. then the synchronization between real and virtual cell should be performed. Easy method for updating process is to synchronize every device and robot of the workcell.

The device d generate external signal for updating module of workcell controller and controller perform the so called **global updating** process, namely:

$$(\forall x \in D \cup M \cup R)(up(x) = 1 \Rightarrow \lambda_x(s_x, q_x, 1) = q_x \in S(x))$$

Such the global updating process is not necessary for each device. Only the part of all devices should be updating. To specify such *local updating* process we introduce the causality relation between events.
Let predicate $Occ(e)$ denotes that the event e has occurred. The causality relation is defined as follows:

$$e \rightsquigarrow e' \Leftrightarrow (\neg\, Occ(e) \Rightarrow \neg\, Occ(e'))$$

and expresses that event e is one of causes of event e'.
The causality relation is reflexive, asymmetrical and transitive relation.
Let $Trace_{[t',t]} = \bigcup\{tr_{[t',t]}(x) | x \in D \cup M \cup R\}$ be the union of all devices and robots traces.
Based on above definition we can construct the set of devices and robots for which the updating the virtual process is needed.

$$UpDate(s(t')) = \{x \in D \cup M \cup R | (\exists e \in tr_{[t',t]}(x))((s(t'), t') \rightsquigarrow e)\}$$

Now, we define **local updating** process as follows:

$$(\forall x \in UpDate(s(t')))(up(x) = 1 \Rightarrow \lambda_x(s_x, q_x, 1) = q_x \in S_V(x))$$

Corollary 3. *It is easy to proof that so defined local updating process is equivalent to global one, i.e., synchronization of devices and robots from the set UpDate is equivalent to the synchronization of all devices and robots of CAW.*

Moreover

Corollary 4. *If $t' > t$ then $UpDate = \{d\}$ and only for the d-device the synchronization is necessary to performed.*

Pre-Diagnosis: Let $q_d(t)$ be the current real state of d-device in time t, registered by passive Dev_d^P automaton and $s_d(t') \in S_V(d)$ be a virtual state of d-device simulator in time t' . Let $q(t) = s(t')$ and $t' \neq t$ and $t < t' - \tau_0$ or $t > t' + \tau_0$ where τ_0 is tolerance time-window.

In this case the diagnosis of real cell should be performed. To reduce the complexity of such process we use these same causality relation $\rightsquigarrow$ in order to eliminate the devices which do not need the diagnostics process. By $CON(s(t'))$ we denote the set of events which are direct causes of event $(s(t'), t')$, i.e.

$$CON(s(t')) = \{e \in Trace_{[t',t]} | e \rightsquigarrow (s(t'), t') \wedge (\neg \exists e')(e \rightsquigarrow e' \wedge e' \rightsquigarrow (s(t'), t'))\}$$

From the set $CON(s(t'))$ we eliminate these events which ware previously monitored, i.e.

$$Con(s(t')) = CON(s(t')) - Monitored\_Events$$

Based on the set $Con(s(t'))$ we can find the devices or robots for which diagnosis is needed as follows:

$$Diag(s(t')) = \{x \in D \cup M \cup R | (\exists e \in tr_{[t',t]}(x))(e \in Con(s(t')))\}$$

Additionally the taxonomy of failure type can be performed.

6 Summary

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 real-time discrete event simulator is used next to generate a sequence of future events of the virtual cell in a given time-window. These events are compared with current states of the real cell and are used to predict motion commands of robots and to monitor the process flow. The architecture, called Computer Assisted Workcell, offers support methods and tools at the following layers of control: organization, coordination, and execution.

References

1. E.D. Sacerdot, *Planning in a hierarchy of abstraction spaces*, *Artificial Intelligence*, vol. 15, no. 2, 1981
2. D. McDermott, *A temporal logic for reasoning about processes and plans*, *Cognitive Science*, vol. 6, 1982
3. G. N. Saridis, "Intelligent robotic control", *IEEE Trans. Autom. Contr.* vol. AC-28,5,1983
4. G.N. Saridis,"Analytical formulation of the principle of increasing precision with decreasing intelligence for intelligent machines", *Automatica*, 1989

5. A. Meystel, "Intelligent control in robotics",*J. Robotic Syst.*, vol.5, 5, 1988
6. J.A. Buzacott "Modelling manufacturing systems", *Robotics and Comp. Integr. Manufac.*,vol. 2, 1, 1985
7. E.G. Coffman, M. Elphick, A. Shoshani. System Deadlock.*Computing Surveys*, 3(2),67-78, 1971
8. Z.Banaszak, E.Roszkowska. Deadlock Avoidance in Concurrent Processes. *Foundations of Control*, 18(1-2),3-17, 1988
9. P.G.Ranky, C.Y.Ho *Robot Modeling. Control and Applications with Software*, Springer Verlag. 1985
10. A. Kusiak *Intelligent Manufacturing Systems.* Prentice Hall. 1990
11. J.E. Lenz. *Flexible Manufacturing.* Marcel Dekker, Inc., 1989
12. A.C. Sanderson, L.S. Homem De Mello and H. Zhang. Assembly Sequence Planning. *AI Magazine*, 11(1), Spring 1990
13. B.P. Zeigler. *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, 1984
14. W. Jacak, ICARS Simulation based CAPP/CAM System for Intelligent Control Design in Manufacturing Automation, in *The International Handbook on Robot Simulation System*, ed. D.W. Wloka, John Wiley, 1993
15. W. Jacak and J.W. Rozenblit. Automatic Simulation of a Robot Program for a Sequential Manufacturing Process, *Robotica* 10/3, 45-56, 1992.
16. W. Jacak. Strategies for Searching Collision-Free Manipulator Motions: Automata Theory Approach. *Robotica*, 7, 129-138, 1989.
17. W. Jacak. A Discrete Kinematic Model of Robot in the Cartesian Space. *IEEE Trans. on Robotics and Automation*, 5(4), 435-446, 1989
18. W. Jacak. Discrete Kinematic Modeling Techniques in Cartesian Space for Robotic System. in: *Advances in Control and Dynamics Systems*, ed. C.T. Leondes, Academic Press, 1991
19. K.Shin, N.McKay, A Dynamic Programming Approach to Trajectory Planning of Robotic Manipulators. *IEEE Trans. on Automatic Control*, 31(6), 491-500, 1986
20. K.Shin, N.McKay, Minimum Time Control of Robotic Manipulator with Geometric Path Constrains. *IEEE Trans. on Automatic Control*, 30(6), 531-541, 1985
21. W.Jacak, I.Dulęba, P.Rogaliński. A Graph- Searching Approach to Trajectory Planning of Robot's Manipulator, *Robotica*, 10/6, 38-47, 1992