SOFHICOR - A Service-Oriented Architecture for Holonic Manufacturing Control

Summary

1. System architecture
    - Open Control, Intelligent Embedded Devices/Intelligent Product, Semi-heterarchical control, Holarchy
    - Resource Holons
    - Product Holons
    - Order Holons
    - Expertise Holons
2. Managing Resource Breakdown/Recovery
3. Management of Rush Orders
4. Experimental results. The pilot platform

1. System architecture

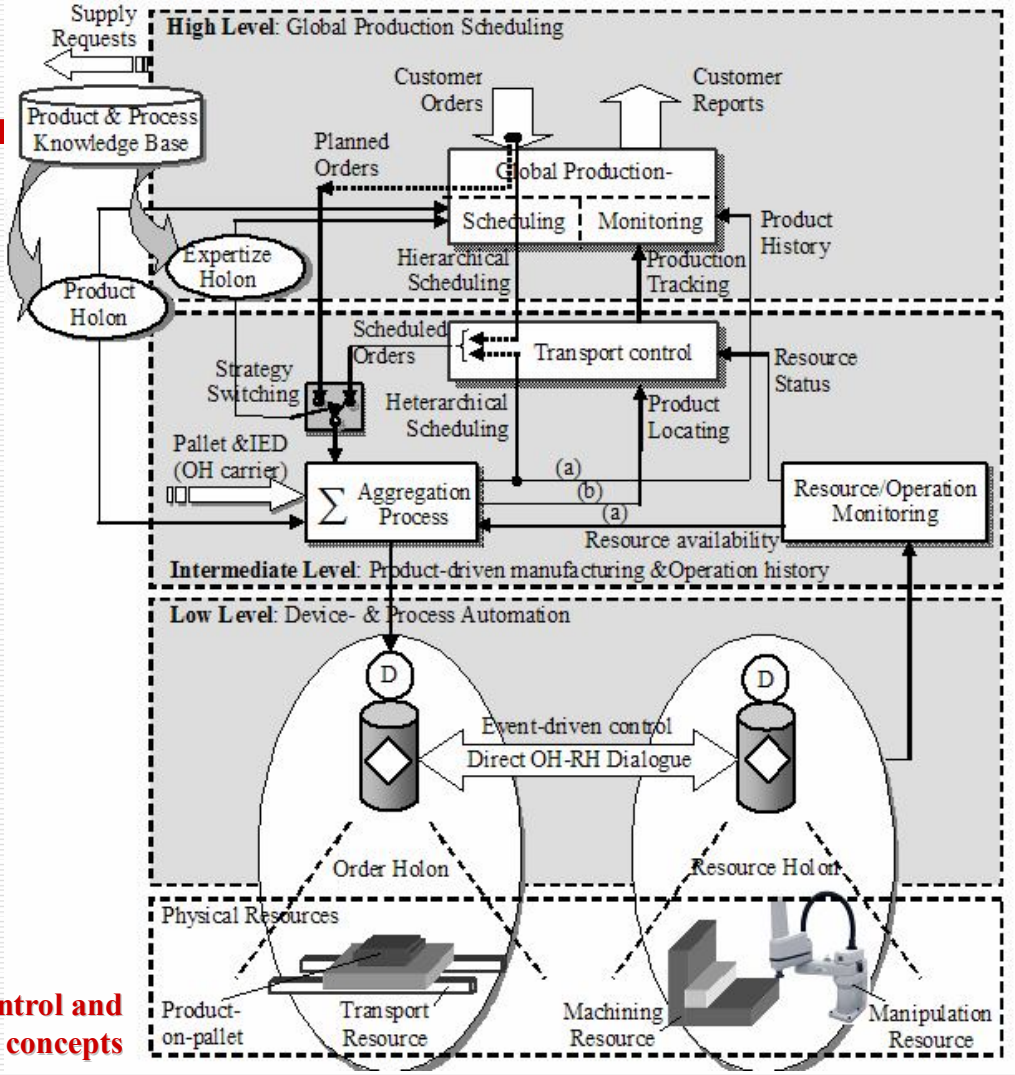
The Open Control (OC) Concept

A global control paradigm, in which traditional (explicit) control is augmented by a new type of control (implicit): *entities can be strictly controlled hierarchically and, at the same time, they can be influenced heterarchically by their environment (environmental) and/or by other entities (societal).*

Effect: allows designing distributed control systems that are both agile and globally optimized, thus reducing the myopic behaviour of self-organized architectures and increasing the agility of traditional architectures.

The Open Control uses IED to implement the **Intelligent Product** (IP) functionality (based on the "*Client-Server*" model (IP -the Server); **strategies**:

- Non-negotiated heterarchical;
- Negotiated heterarchical;
- *Semi-heterarchical*



The Open Control and Intelligent Product concepts

1. System architecture

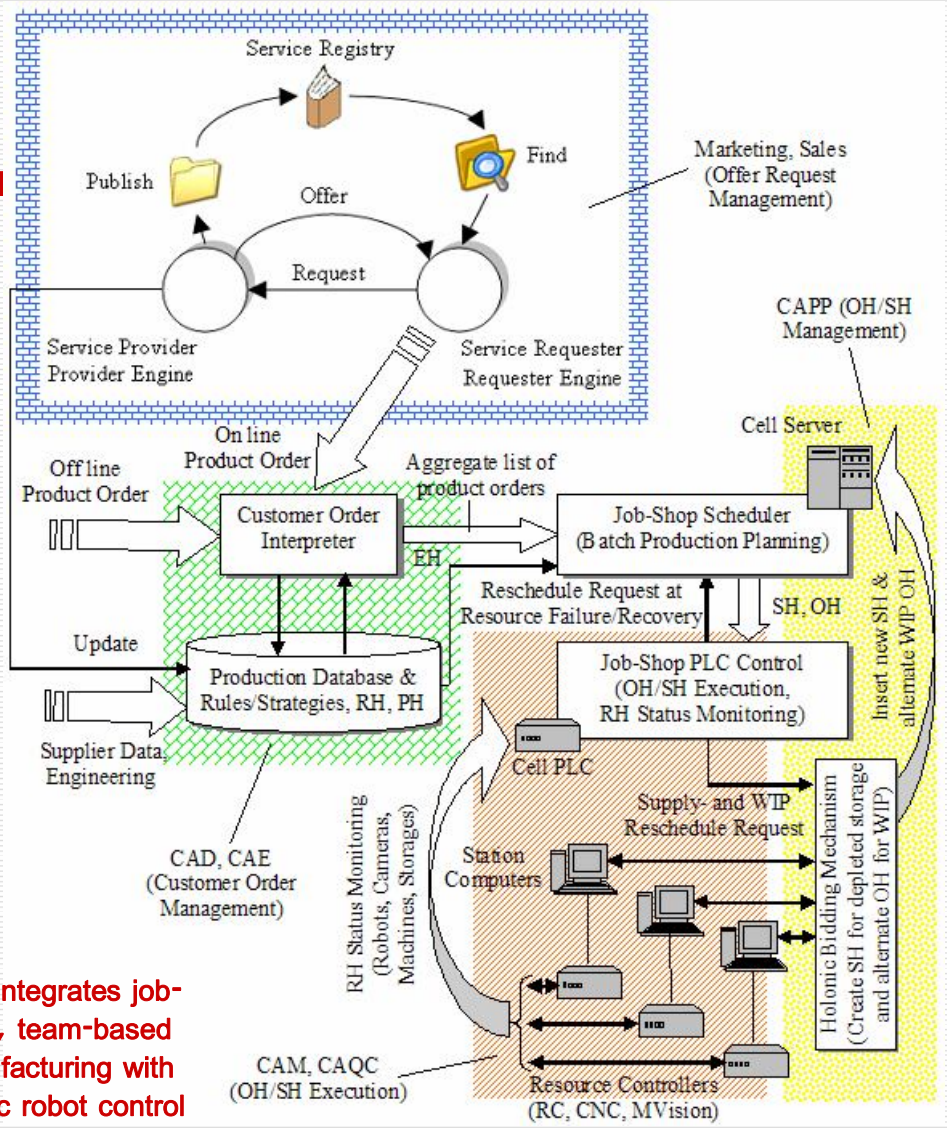
Key features of holonic manufacturing control:

1. Controlled process: networked robot assembly with in-line part- machining and supply
2. Manufacturing structure: job-shop type, transportation by closed-loop conveyor:
3. Use real-time, high-speed machine vision to condition materials/parts (GVR, AVI):
   - ✓ Visually Qualify, Recognize, Locate items in the workplace foreground
   - ✓ Feature-based product- and component measuring: in-line CAQC
   - ✓ Authorize part access based on clear fingerprint check: avoid collision
4. Physical infrastructure addressed: industrial robots, CNC machine tools, machine vision, material storages, intelligent feeding devices, tool holders
5. Semi-heterarchical control architecture, designed as HMES
6. PROSA reference architecture taken as model for HMES development
7. Holons:
   - ✓ Are autonomous and cooperating agents
   - ✓ Encapsulate an information part and a physical part
   - ✓ Holon type: Product- (PH), Resource- (RH), Order- (OH), and Expertise (EH) Holons
8. A Service Oriented Architecture (SOA) integrates 4 areas: (i) **Offer Request Management**; (ii) **Client Order Management**; (iii) **OH & SH Management**; (iv) **OH Execution** (Tracking)
9. Fault-tolerance and disaster recovery provided

1. System architecture

SOA of job shop manufacturing with Holonic control of RC, CNC and MV

- Closed-loop conveyor
- Networked robot-vision (RV) workstations
- Dual-camera RV stations: stationary (*down looking*) & mobile (*arm-mounted*)
- CNC machine tools (3D, 4D)
- Magnetic RD/WR devices for pallet (product) traceability
- On-line created Supply Holons
- KB Management of Client Orders
- Fault tolerance and disaster recovery

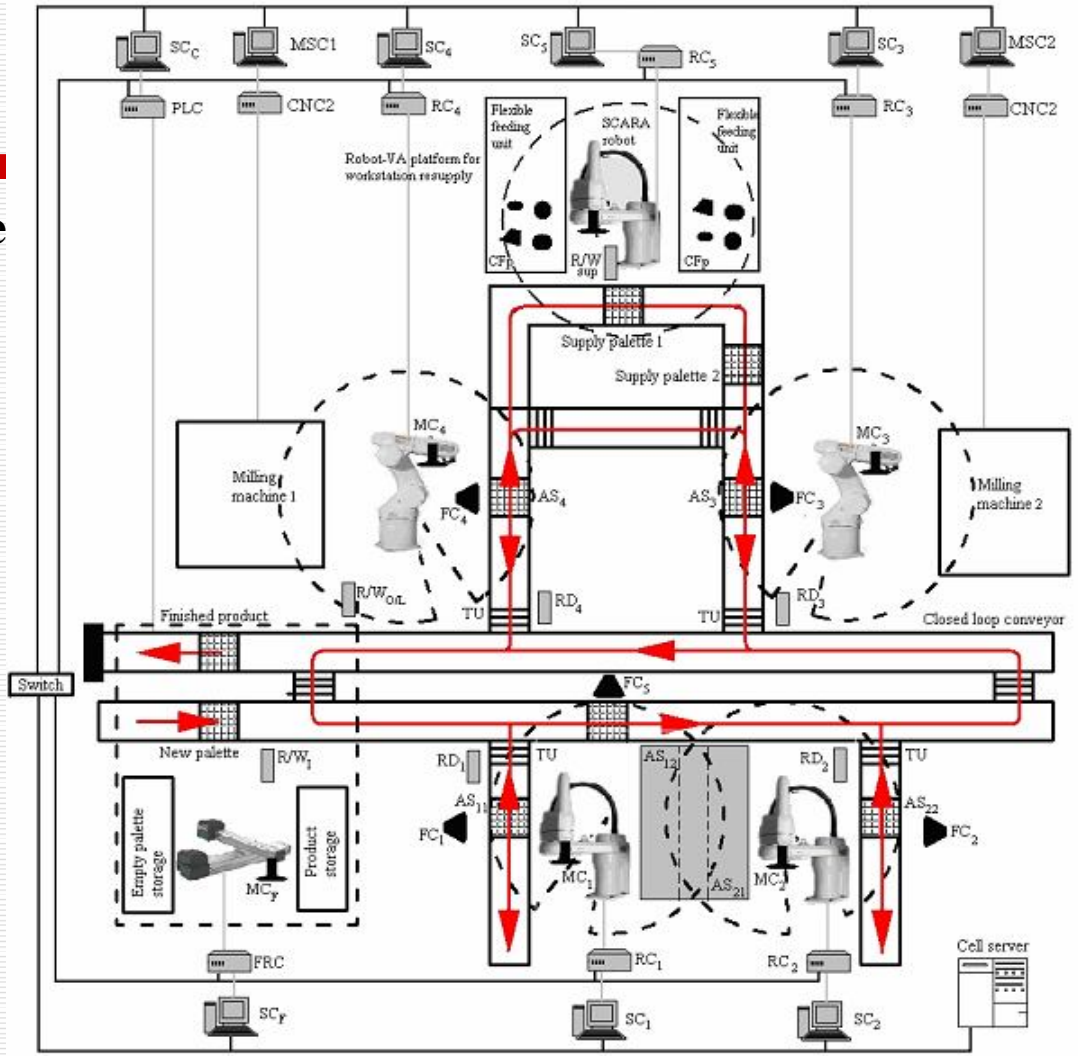


SOA integrates job-shop, team-based manufacturing with holonic robot control

1. System architecture

Distributed control architecture

- Distributed architecture:
  information entities (holons)
  with physical counterparts:
  Cell- & St- Server, Comp, Ctrl
- Multiple-LAN communication:
  Ethernet, serial, ring, I/O
- Cell Server (IBM xSeries) for:
  - Client order management
  - Job-shop batch scheduler
- PC Station Computers for:
  - CNP data provider
  - Data and pg replication
  - St_Ctrl terminal: edit, track
- PLC for:
  - OH execution
  - Product traceability

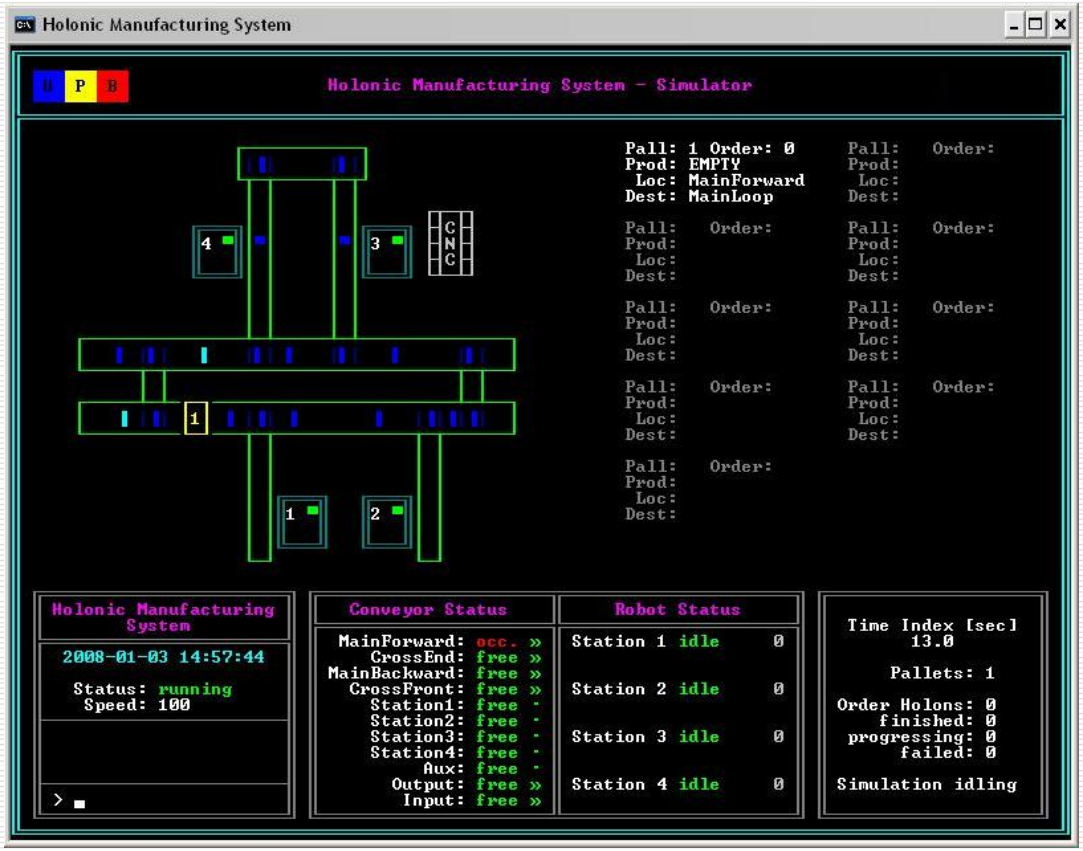


Holonic manufacturing system with self-supply of assembly parts

1. System architecture

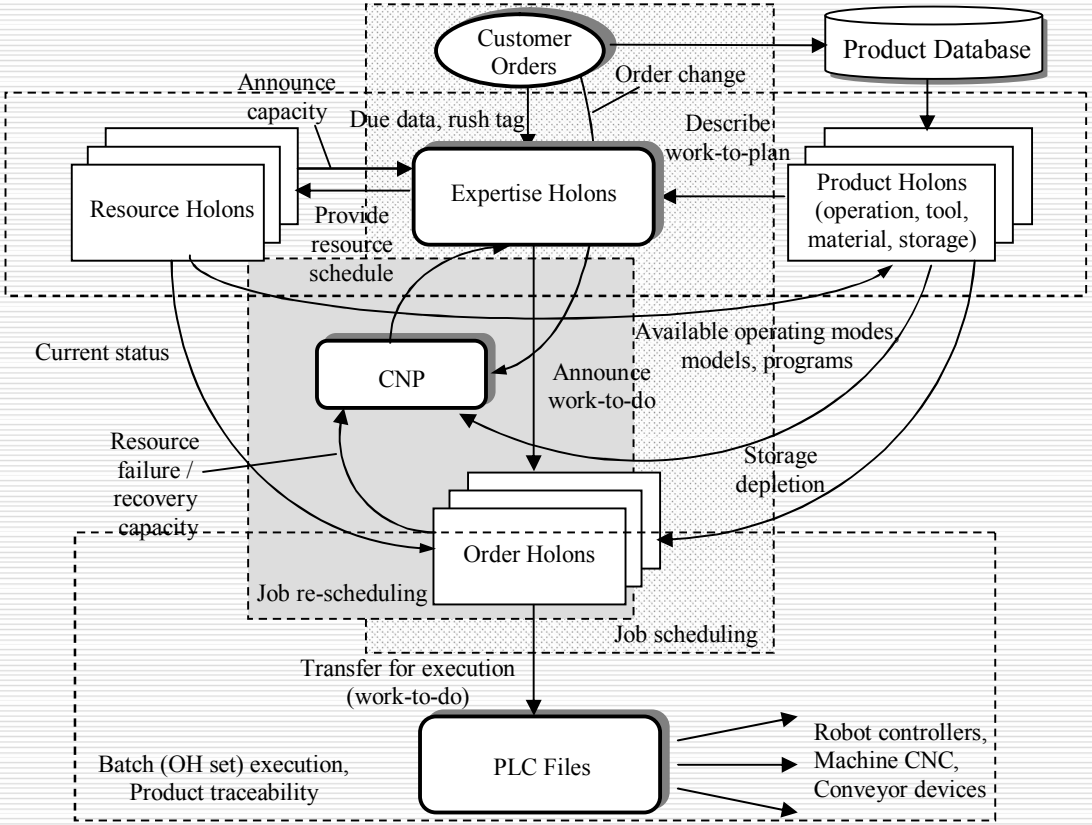
Dynamic simulation tool (DST)

- DST assists & stepwise validates:
  - the creation of OH (off-line)
  - production tracking (on-line)
- DST simulates product transport:
  - A *transportation time matrix* (TTM) is created by measuring the time used to move a pallet between points (stoppers)
  - Smallest time index (transport time slice) *t.s* = 0.5 seconds
- Usage of DST core routines with *variable time base*:
  - Visual simulation: timed mode run, event-driven TTM (0.5 s)
  - OH (re) scheduling: computed mode run, instant-driven TTM
- GUI - conveyor divided in sectors



The dynamic simulator is based on a process-oriented (*product_on_pallet* transport) Graphic User Interface

1. System architecture



Basic holon cooperation and communication structure in the semi-heterarchical control architecture

The Holarchy

- Holon types (PROSA reference):
  - *Basic*: Product-, Resource-, Order-
  - *Staff*: Expertise-
- Holarchy types (set of basic rules for holon cooperation for management & control of all manufacturing tasks):
  - Hierarchical (optimal planning)
  - Heterarchical (flexibility, quick response to changes)
- Automatic switch between the two holarchies:
  - *triggered* by: resource failure / recovery, operation failure, supply request and order change
  - based on *CNP mechanism*
- OO holon design: a class containing data fields and functionalities
- HolonManager: coordinates data exchange between holons

1. System architecture

Holon structure (selection, examples):

Product Holon (PH):

Product code: #IDpr*k*;

Operation set (operation vector): [o_*k*1, o_*k*2, ... , o_*k*f]

Operation order (predecessor/operation vector): [p(o_*k*1), p(o_*k*2), ... , p(o_*k*f)]

where:

p(o_*ki*) {Ø, ... , (f – 1)} are predecessor operations

Materials (components): vector of materials [type; number.] / operation):

[m (t, no.m)] (o_*k*1), [m (t, no.m)] (o_*k*2), ... , [m(t, no.m)] (o_*k*f),

where

[m (t, no.m)] (o_*ki*) = [t_1*i* | no.m_1*i*, t_2*i* | no.m_2*i*, ... , t_fi | no.m_f*i*]

Tools (instruments): vector of necessary tools [type; number] / operation):

[s (t, no.s)] (o_*k*1), [s (t, no.s)] (o_*k*2), ... , [s (t, no.s)] (o_*k*f),

where

[s [t, (no.s)] (o_*k*i) = [t_1*i* | no.s_1*i*, t_2*i* | no.s_2*i*, ... , t_u*i* | no.s_u*i*]

Programs: Vector of programs (vector of #ID programs / operation):

IDprg(o_*k*1), IDprg(o_*k*2), ... , IDprg(o_*k*f),

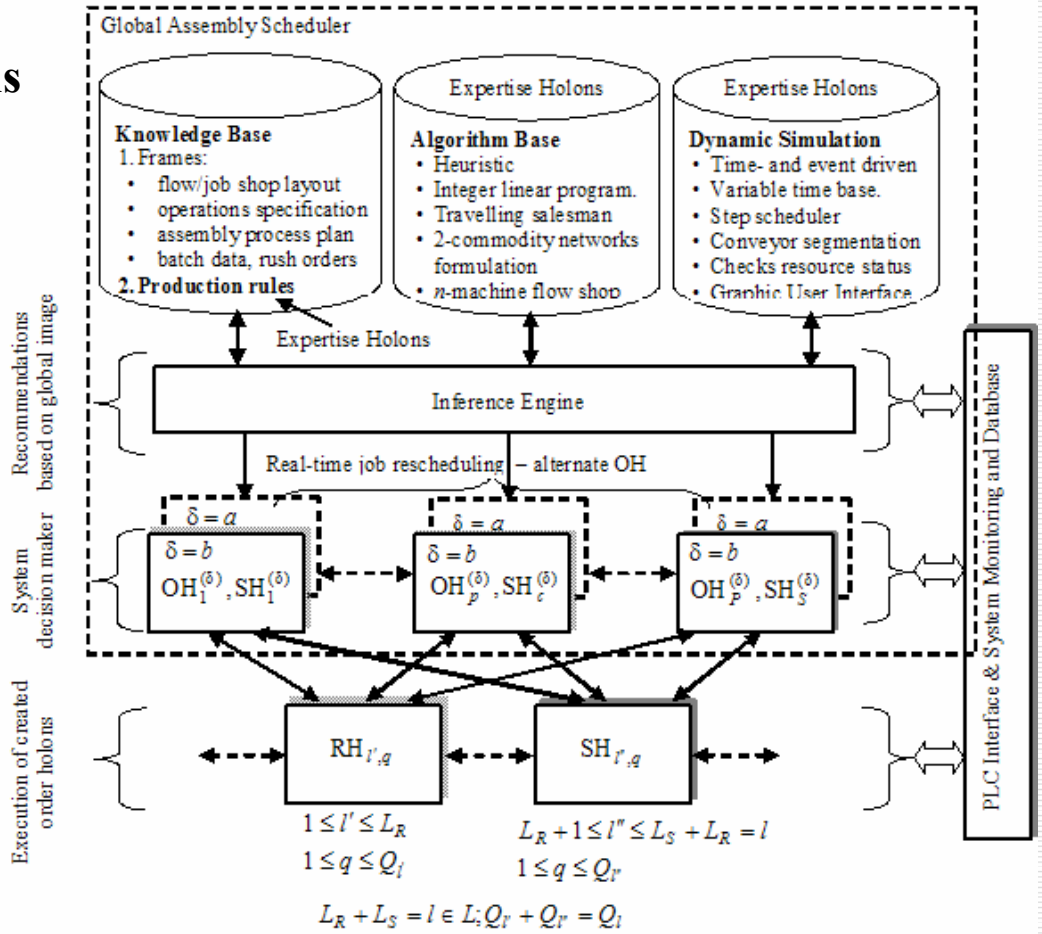
where

IDprg(o_*ki*) = IDprg_1*i*, IDprg_2*i*, ... , IDprg_v*i*

1. System architecture

Job scheduling with Expertise Holons

- A Global Production Scheduler (GPS) maps *production plans* for batch products in the form of Order- (OH) and Supply- Holons (SH); an algorithm base is embedded into a KB-system.
- OH, SH created by the GPS: treated as *recommendations* (hierarchical scheduling mode)
- *Alternate* OH, SH are created based on a task bidding mechanism of CNP type (heterarchical scheduling mode)
- Expertise Holons (EH) are used by the GPS as *scheduling agents*; they optimize a batch cost function: time of work, throughput, machine load
- Two GPS methods were developed:
  - KB-scheduling;
  - Step Scheduler

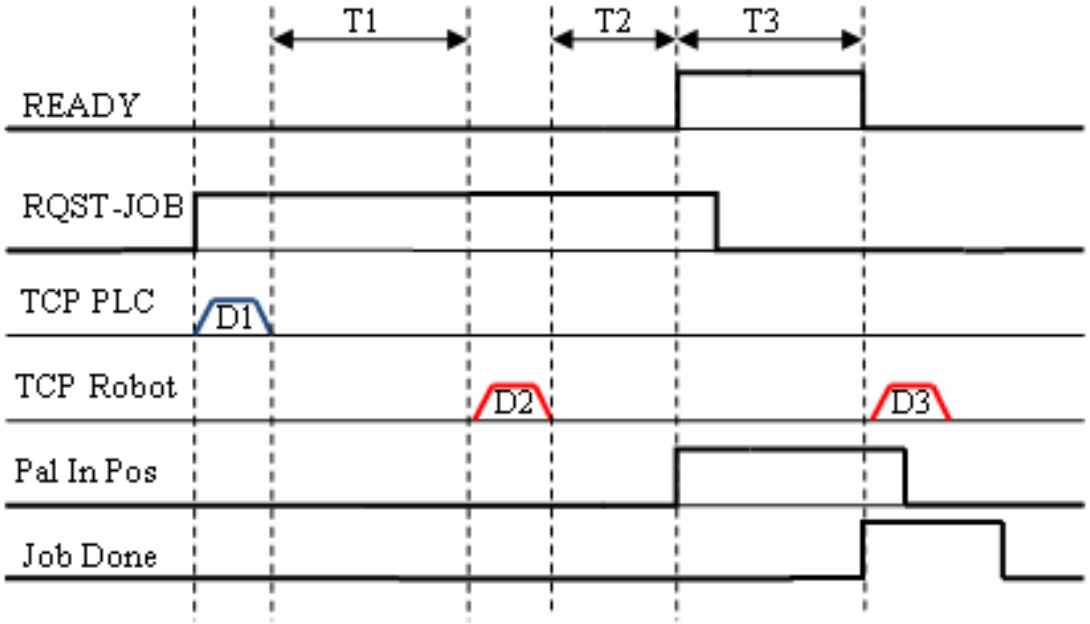


KB batch scheduling and creation of Order and Supply Holons

1. System architecture

PLC – RC communication

- PLC checks OH execution start
- {rqst_status | ack_status} initiated periodically:
  - checks operational state of R(S)H
  - period established function of the duration of R(S)H diagnosis tasks
  - watchdog timer included
- Job data transfers Product Holons:
  - data sent via Ethernet\
  - job_type sends PH requirements to resource holons RH, SH
  - end_status updates the PH and includes job termination reports
- Timeouts:
  - timeout_T1: part supply and other equipment interrogation
  - timeout_T2: pallet movement from main conveyor to workstation
  - timeout_T3: job execution time



Communication protocol between the PLC and a Robot Controller for authorizing an OH operation execution

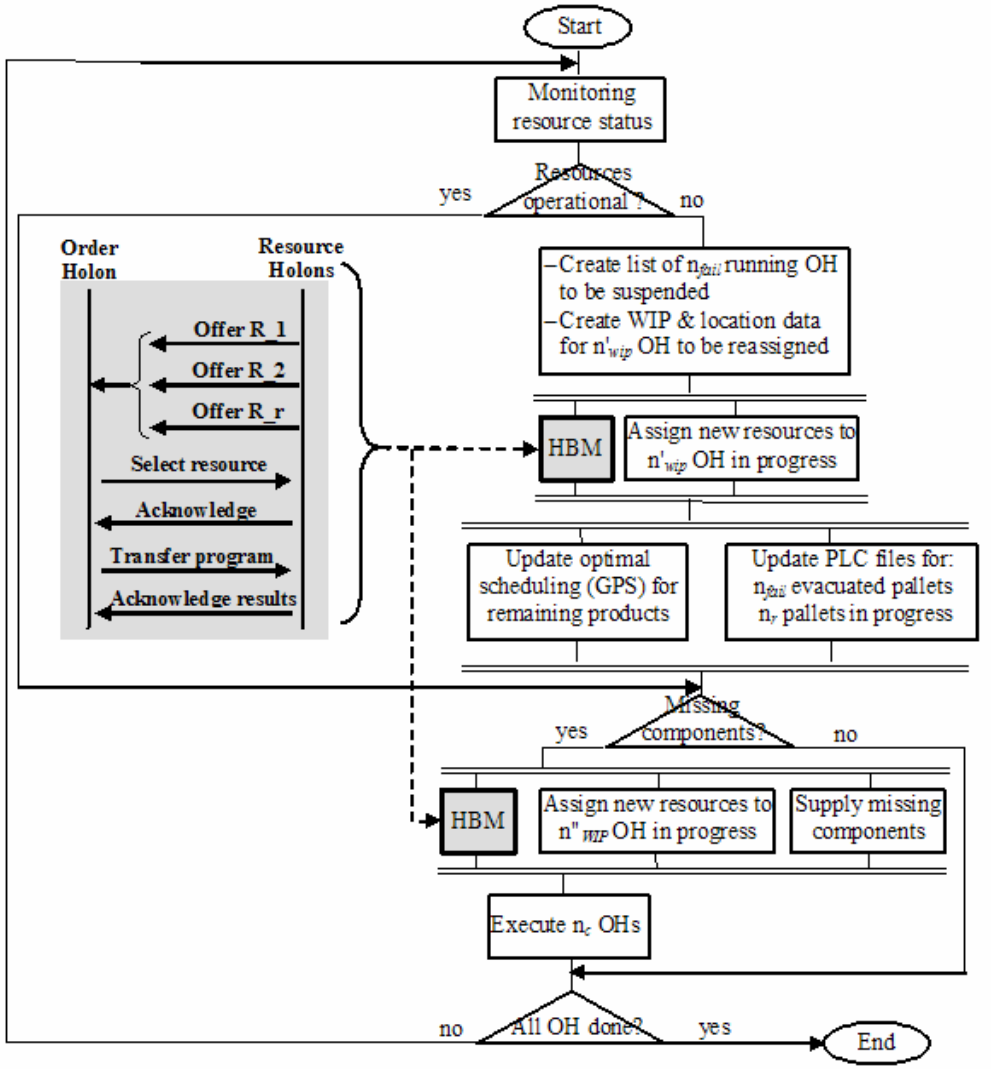
2. Managing Resource Breakdown/Recovery

- **Fail-safe mechanism for production control**:
  - Triggered whenever: a resource is down or the result of an operation is negative, or components are missing
  - A *FailureManager* and the *RecoveryManager* control changers (failures & recoveries)
- The **status** of the system's resources is periodically checked by the PLC responsible for OH, SH execution Any robot can assume a number of states, stored in the corresponding resource holon *Status* field:
  - off-line: the resource is not available at the current time
  - failed**:** the resource encountered a failure not yet acknowledged by the *FailureManager*, once dealt with, the status is set to off-line
  - suspend**:** the resource is executing an operation which has been suspended for the time being
  - waiting: the resource has been allocated and is waiting for the item to arrive before it may carry out the task
  - busy: the resource is currently executing an operation
  - recover: the resource is recovering but not yet acknowledged by the *RecoveryManager*, once the situation recognized, the status is set to idle
  - idle: the resource is fully ready to receive a task
- **Global image**: the states *failed* and *off-line* describe the robot as non-functional, whereas the states *suspend*, *waiting*, *busy*, *recover*, and *idle* are used for a fully operational resource

2. Managing Resource Breakdown/Recovery

Failure Manager flowchart:

- HBM: Holonic Bidding Mechanism of CNP type, assigns operations to resources
- N: all scheduled OHs
- n_{WIP}: OH currently in execution
- n_{fin}: finished OHs
- n_d: OHs delayed due to part missing
- n_{fail}: OHs failed (in execution, cannot be finished)
- n_e: OHs not yet started, can no more be executed
- $n'_{WIP} = n_{wip} - n_{fail}$: OHs in execution, can be continued without rescheduling
- $n''_{WIP} = n_{wip} - n_d$: OHs that can be done by re scheduling to resources disposing of parts
- $n_c = N - n_{fin} - n_{WIP} - n_e$: OHs not yet started, can be processed



Dynamic OH rescheduling at resource failure/storage depletion with embedded CNP job negotiation (monex)

2. Managing Resource Breakdown/Recovery

Failure Manager actions (OH):

1. Stop production immediately
2. Update RH with the new states of all robots
3. Read orders from the system.
4. Evaluate all products if they can still be completed on the system, by checking the status of each order
   - if the order has been in the failing robot station, mark it as failed and evacuate it;
   - if the order is in the system, but cannot be completed anymore due to the resource failure mark it as failed and evacuate it;
   - if the order is not yet in the system, but cannot be completed due to the resource failure mark it as failed.
5. For the remaining orders in the system, locate them and initialize the transport simulation. The Contract Net Protocol is embedded in this stage.
6. Run the scheduling algorithm taking the finished and failed orders (that do not need any scheduling)
7. Delete the orders stored in the system and transfer the updated orders to the system
8. Resume production

Failure Manager actions (SH):

1. One SH is created by specifying the *parts to be retrieved* from 1 or 2 flexible feeding devices by a visually guided SCARA robot, and the *path of the supply pallet* to the depleted storage
2. From the OH currently in execution, n_d will be delayed until the empty storage is fed, and $n''_{WIP} = n_{wip} - n_d$ OH will be re scheduled by the holonic bidding mechanism to robots disposing of all necessary assembly parts.

3. Management of Rush Orders

- The system is agile to changes occurring in production orders too, i.e. manages **rush order**s received as *new batch requests* from the ERP level while executing an already scheduled batch production (a sequence of OH).
- Because of the similarities between a task run on a processor and a batch of orders executed in a manufacturing cell, the Earliest Deadline First (EDF) procedure was used.
- *Feasibility test* for applying EDF:
  1. All tasks are periodic, independent and fully pre-emptive
  2. All tasks are released at the beginning of the period and have deadlines equal to their period
  3. All tasks have a fixed computation time or a fixed upper bound which is less or equal to their period
  4. No task can voluntarily stop itself
  5. All overheads are assumed to be 0
  6. There is only one processor, then a set of n periodic tasks can be scheduled if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, \text{ where } n = \text{number of tasks}, C_i = \text{execution time}, T_i = \text{cycle time}$$

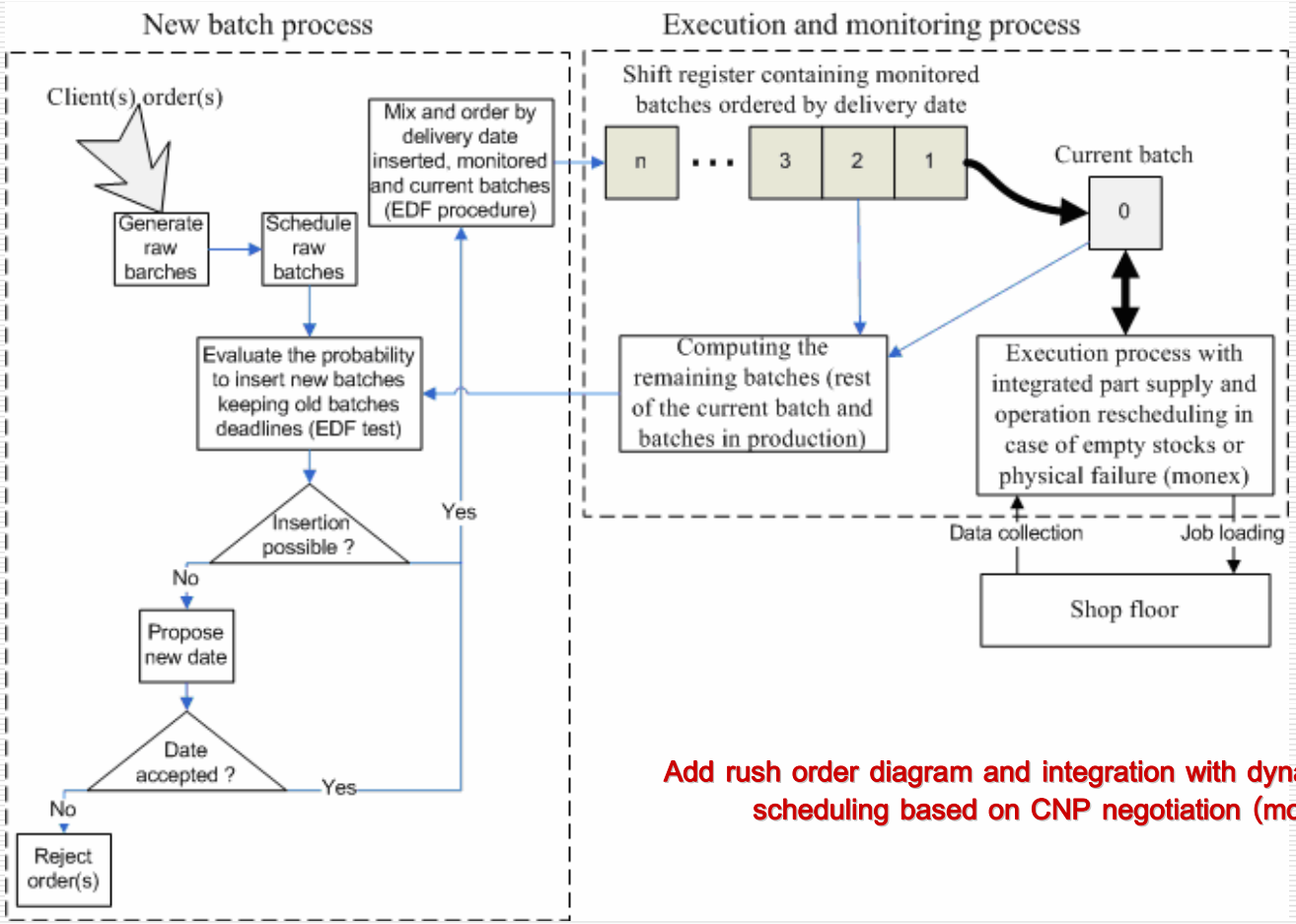
or, in other words, if the utilization of the processor (resource) is less than 100%

3. Management of Rush Orders

- The EDF approach used to insert **rush orders** in a production already scheduled:

1. Compute the remaining time for finishing the rest of the current batch (if necessary).
2. Insert new production data: product types, quantities, delivery dates.
3. Separate products according to their delivery date.
4. Form the entities "production batches" (a *production batch* is composed of all the products having the same delivery date).
5. Generate raw orders inside the production batches (APO lists).
6. Schedule the raw orders (using a GPS algorithm, e.g. KBS or Step Scheduler), compute the makespan and test if the inserted batch can be done (the makespan is smaller than the time interval to delivery date if production starts now).
7. Analyse the possibility of allocating the batches to the cell using the EDF and second equation for feasibility test.
8. Allocate batches to the system according to EDF.
9. Resume execution process with new scheduled OH.

3. Management of Rush Orders



Add rush order diagram and integration with dynamic job re-scheduling based on CNP negotiation (monex)

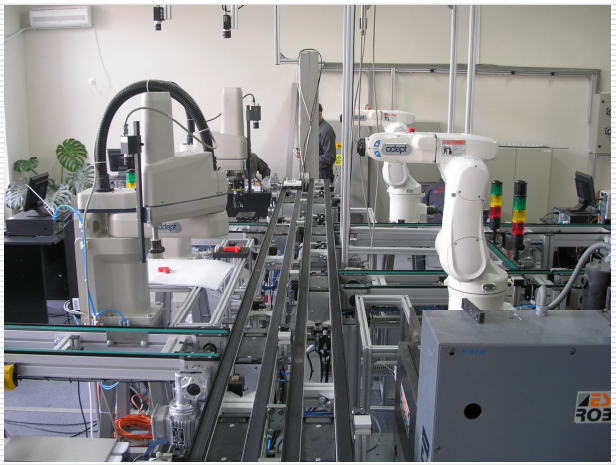
3. Management of Rush Orders

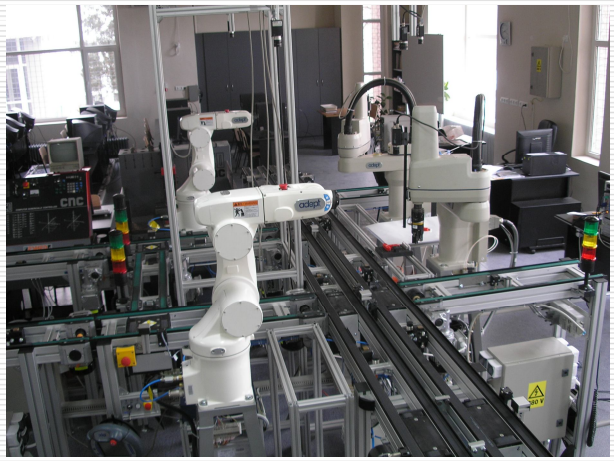
- The capability of adding rush orders to production needs a new entity, the **Batch Holon**.
- Job scheduling is done at batch level (all orders with the same delivery date are scheduled together) and then batches are assigned to the cell according to their delivery date, using the EDF procedure

The minimal structure of a *batch holon*

Type	Name	Description
string	batch_name	Name or index of the batch
Date	delivery_date	Delivery date of the orders
Product[ ]	requested_products	Vector containing the products to be executed
Resource[ ]	used_resources	Vector containing the configuration used for current batch planning
Order[ ]	orders_to_execute	Vector containing the entities OH already scheduled using a specified cell structure (defined by the variable used_resources)
int	makespan	Time interval needed for the current batch to be executed if started now and not interrupted (it is a result of scheduling)

4. Experimental results. The SOFHICOR pilot platform for holonic manufacturing control with intelligent robots

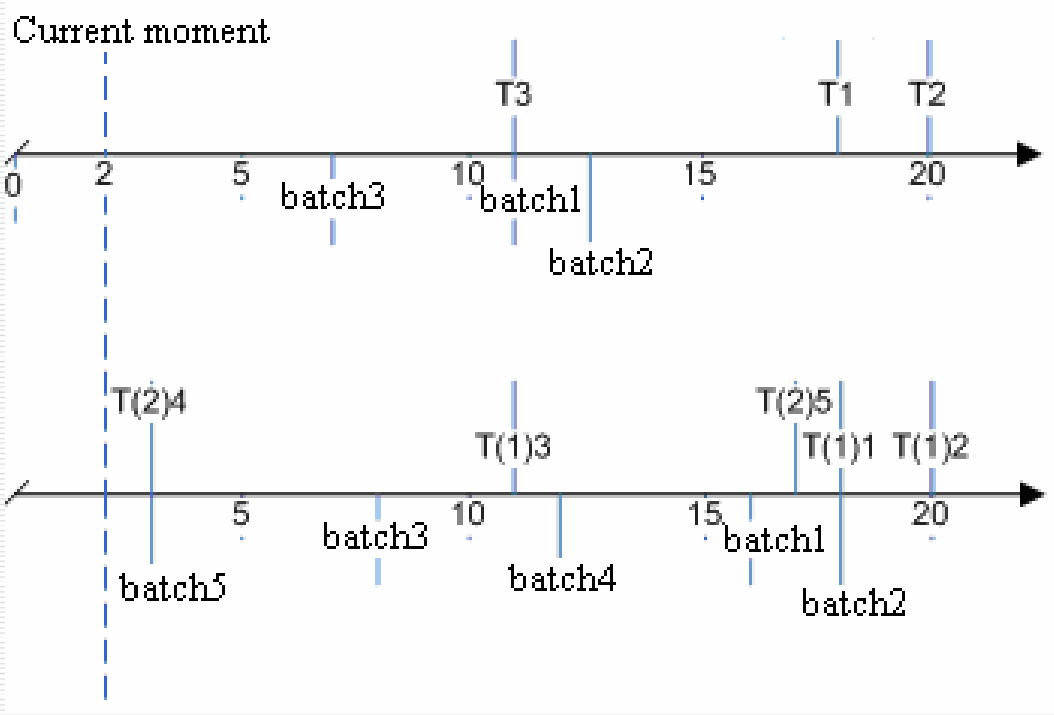




Layout of the manufacturing cell with holonic control

4. Experimental results. Batch production results

- Production scheduling at batch level was implemented and tested using the EDF method.
- The figure shows the results obtained when two new batch orders T24 = (4, 17) and T25 = (1, 3) are received at time T = 2 after the execution of three planned batches: T11 = (2, 18), T12 = (3, 20), T13 = (7, 11) started.
- Here T*ij* = (*m*, *dd*) signifies the number (*j*) of the batch for which execution was requested at date *i*; the batch has the makespan *m* and due delivery date, *dd* (both expressed in time units).



Inserting new batches among executing ones with the EDF algorithm