Management of changes in a holonic manufacturing system with dual-horizon dynamic rescheduling of production orders

Theodor Borangiu*, Silviu Raileanu*, Andrei Rosu*, Mihai Parlea*, Florin D. Anton*

**University Politehnica of Bucharest, Dept. of Automation and Applied Informatics, Bucharest, RO 060032, ROMANIA (Tel: +40 21 402 9314; e-mail: borangiu@cimr.pub.ro).*

Abstract: The paper describes a solution and implementing framework for the management of changes which may occur in a holonic manufacturing system. This solution is part of the semi-heterarchical control architecture developed for agile job shop assembly with intelligent robots-vision workstations. Two categories of changes in the manufacturing system are considered: (i) changes occurring in resource status at process level: resource breakdown, failure of (vision-based) in-line inspection operation, and depletion of local robot storages; (ii) changes in production orders at business (ERP) level: rush orders. All these situations trigger production plan update and rescheduling (redefine the list of Order Holons) by pipelining CNP-type resource bidding at shop-floor horizon with global product scheduling at aggregate batch horizon. Failure- and recovery management are developed as generic scenarios embedding the CNP mechanism into production self-rescheduling. Implementing solutions and experimental results are reported for a 6-station robot-vision assembly cell with twin-track closed-loop pallet transportation system and product tracking RD/WR devices. Future developments will consider manufacturing integration at enterprise level.

Keywords: holonic manufacturing, distributed control, reconfigurable systems, robotics, applied AI

1. INTRODUCTION

Some of the problems that discrete, repetitive manufacturing industry faces are: resource availability (unexpected failure or recovery of a resource and insertion or removal of resources from the production process) and treatment of "rush orders". To cope with these problems some new concepts have been developed such as Flexible Manufacturing Systems (Groover, 1987; Upton, 1992), Holonic Manufacturing Systems (Van Brussel *et al.*, 1998; Leitao, 2006) Product-Driven Control for Manufacturing. The first, FMS, deals with the physical composition of a manufacturing cell which has a minimal degree of flexibility allowing easy reconfiguration and also facing disturbances like resource breakdowns. The second concept, HMS, deals with the control part of a manufacturing cell, structuring it into basic building blocks characterized by autonomy and cooperation. Manufacturing tasks are solved by cooperation between these entities and to the exterior the system is seen as a single entity, making it easier to integrate such structures with the upper levels (ERP) of an enterprise. The last concept, of "intelligent product", assumes that "local" intelligence is provided to the product (e.g. moving on a pallet carrier) integrated via RFID devices in an Enhanced Information Management System (IMS-RFID) which is used to retrieve process-, resource- and cell- data for product routing.

The paper deals with one main aspect in the design of distributed control systems for holonic manufacturing – that of reallocating already scheduled production orders (Order Holons) in a perturbed environment.

This environment is a FMS for *job-shop* tasks executed at batch level on various types of products. The control architecture is distributed, of semi-heterarchical type, in which the organizational control is arranged on two levels, referred to as *global* and *local*.

The global level assumes the responsibility for planning and coordination of shop-floor level activities (cell/line/factory) and the resolution of conflicts between local objectives, whereas the local level possesses the autonomy over the planning and control of internal activities within a subsystem (e.g. the robot assembly team).

In semi-heterarchical control architectures, there is an entity placed on a superior decisional level – the Global Production Scheduler (GPS) – which sends aggregate product orders, optimally scheduled – Order Holons (OH), to entities on inferior levels – Device (e.g. Robot, Machine) Controllers – which cooperate to accomplish these orders. In the proposed holonic system the schedules delivered by the GPS are not imposed to any of the individual resources (conveyor, robot, vision, machine tool); instead, they are only recommended to the decision-making entities – the Order Holons (OH). These recommendations will be followed as long as failures or changes do not occur in the system (*hierarchical operating mode*); they will be ignored at failure/change and recovery moments, being replaced by alternate schedules created from resource (Robot Controllers) offers mutually agreed by a cooperation mechanism (*heterarchical operating mode*). The holonic manufacturing control automatically switches between these two modes.

The holonic control strategy follows the key features of the PROSA reference architecture (Van Brussel *et al.*, 1998; Valkaenars *et al.*, 1994), implemented as a HMES extended with:

- Automatic switching between *hierarchical* (efficient use of resources and global production optimization) and *heterarchical* (agility to order changes, e.g. rush orders, and fault tolerance to resource breakdowns) production control modes.
- Automatic planning and execution via Supply Holons (SH) of part supply; automatic generation of self-supply tasks upon detecting local storage depletion.
- In-line vision-based part qualification and inspection of products in user-definable execution stages.
- Robotized processing (e.g. assembling, CNC machine tending, fastening, and assembling) under visual control / guidance.

2. SYSTEM ARCHITECTURE

As suggested by the PROSA abstract, the manufacturing system was broken down into three basic holons:

1. **Resource Holons** (RH): they hold information about cell resources. Any resource may have a number of sub-resources, which are also seen as holons. The hardware part of this type of holon is the physical manipulator and controller with its functions.
2. **Product Holons** (PH): they hold information about a product type. The product information is more than a theoretical description of the physical counterpart but not directly associated with one individual physical item, unlike the resource holon (Leitao and Restivo, 2006).
3. **Order Holons** (OH): represent all information necessary to produce one item of a certain product type. This holon is directly associated with the emerging item, it holds information about the status of this very item at all times reaching from *assembly not started yet* through *order progressing* to *order completed*. OHs are created by a Global Production Scheduler (GPS) from an Aggregate List of Product Orders (APO) generated at ERP level. Alternate OH are automatically created in response to changes in product batches (*rush orders*) and to failures occurring during execution (resource breakdown, storage depletion).

A holon designs a class containing data fields and functionalities. Beside the information part, holons possess a physical part too, like the *product_on_pallet* for OH.

The way in which different types of holons communicate which each other and the type of information they exchange depends on the functionalities imposed to the manufacturing cell. Fig. 1 shows the interaction diagram of the basic holon classes as they were implemented into software to solve scheduling and failure / recovery management problems. A separate software module, the **HolonManager**, hosts all holons in form of arrays of certain types of holons and coordinates the data exchange among them.

The HolonManager entity is responsible with the planning (by help of **Expertise Holons** – EH) and management of OH as Staff Holons in the PROSA architecture do; in addition, the HolonManager interfaces the application with the exterior (maps the OH list in standard PLC files and tracks OH execution).

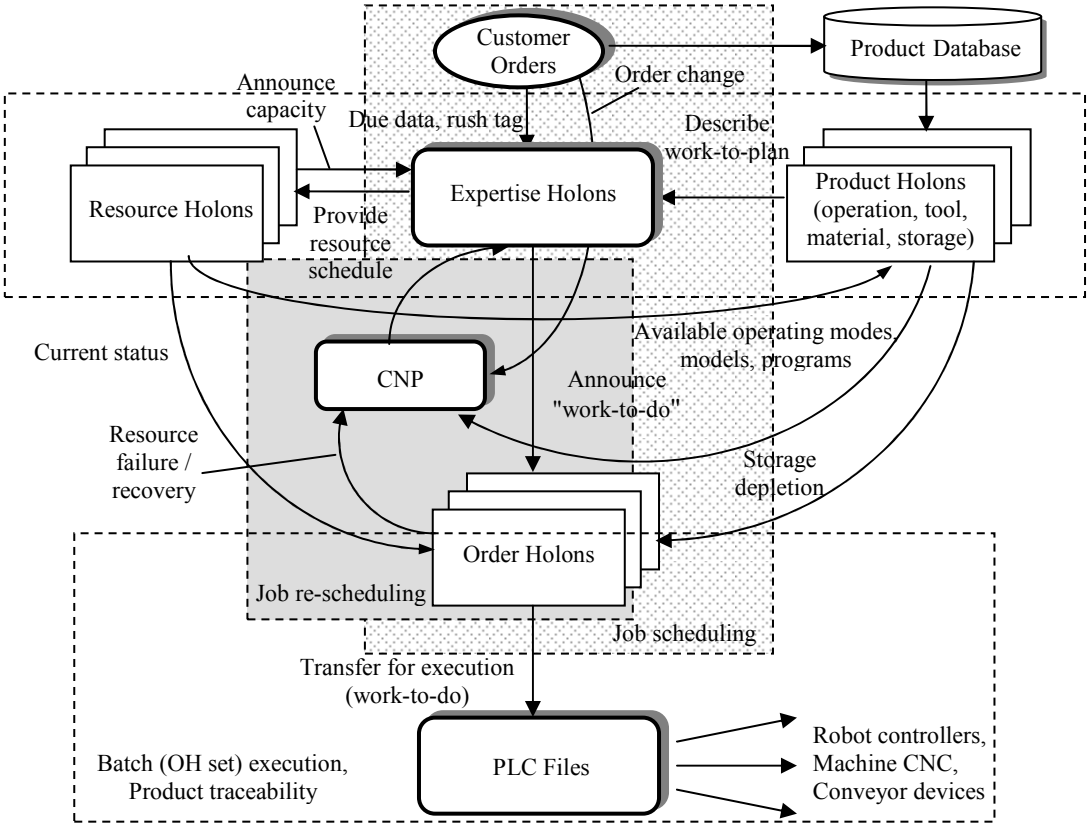

Fig. 1. Basic holon cooperation and communication structure in the semi-heterarchical control architecture

A **basic process plan** (quasi optimal) is generated initially, upon receiving from the ERP level an aggregate production order (APO or raw orders), based on a method which can be selected from the next two: (i) Knowledge-based scheduling (KBS) inspired by (Kusiak, 1990) research; (ii) Resolved Scheduling Rate Planner (Borangiu, 2008). This basic process plan is computed at the global horizon of *P* products of the aggregate batches, and consists from a list of Supply Holons (SH) responsible for feeding the local robot storages and a list of Order Holons (OH) driving product execution. The OH list is mapped into PLC files for batch execution.

Alternative process plans, triggered by resource failure, local storage depletion or occurrence of rush orders, are pipelined automatically: (a) at the horizon of p_E products in course of execution in the system, based on heterarchical contract negotiation schemes (CNP-type) between valid resources; (b) at the global horizon of $P - p_T - p_E$ remaining products, p_T = number of terminated products, based on hierarchical GSP. Two categories of changes are considered:

1. Change occurring in the resource status at shop floor level: (i) breakdown of one resource (e.g. robot, machine tool); (ii) failure of one inspection operation (e.g. visual measurement of a component/assembly); (iii) depletion of one robot workstation storage.
2. Change occurring in production orders, i.e. the system receives a *rush order* as a new batch request (new APO).

All these situations trigger a fail-safe mechanism which manages the changes, providing respectively fault-tolerance at critical events in the first category, and agility in reacting (via ERP) to high-priority batch orders. A *FailureManager* was created for managing changes in resource status. A virtually identical counterpart, the *RecoveryManager*, takes care of the complementary event (resource recovery).

Upon monitoring the processing resources (robots), their status may be at run time: *available* – the resource can process products; *failed* – the resource doesn't respond to the interrogation of the PLC (the entity responsible for Order Holon execution), and consequently cannot be used in production; *no stock* – similar to failed but handled different (the resource cannot be used in production during its re-supply, but it does respond to PLC status interrogations.

There are two types of information exchanges between the PLC (master over OH execution) and the resource controllers for estimation of their status during production execution:

- *Background interrogation*: periodic polling of I/O lines RQST_STATUS and ACK_STATUS between the PLC – OH coordinator and the Resource Controllers (robot, machine tool, ASRS).
- *Ultimate interrogation*: just before taking the decision to direct a pallet (already scheduled to a robot station) to the corresponding robot workplace, a TCP/IP communication between the PLC and the robot controller takes place (according to the protocol in Fig. 2). This communication validates the execution of the current OH operation on the particular resource (robot).

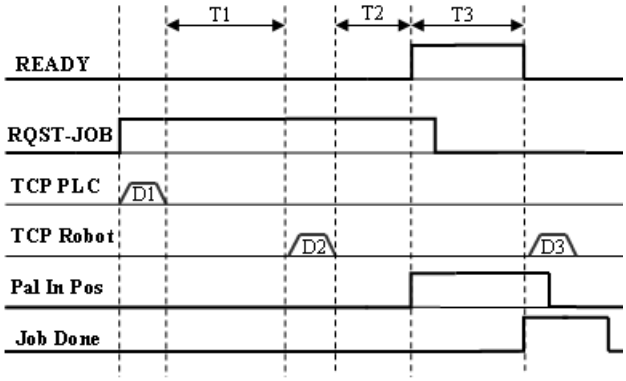

Fig. 2. Communication protocol between the PLC and a Robot Controller for authorizing an OH operation execution

In this protocol, READY is a signal generated by the Robot Controller indicating the *idle* or *busy* state of the resource (robot). The PLC requests through its digital output line RQST-JOB to use the robot for an assigned OH operation upon the product placed on the pallet waiting to enter the robot workstation. D1 details the scheduled job via the TCP PLC transmission line from the PLC to the Robot Controller. The Robot Controller indicates in D2 job acceptance or denial via the TCP Robot transmission line. When the job is accepted, the pallet is directed towards the robot's workplace, where its arrival is signalled to the Robot Controller by the Pal In Pos digital output signal of the PLC. Job Done is a signal indicating job termination (D3 details the way the job terminated: success, failure). T1 is the decision time on job acceptance (storage evaluation etc), T2 is the transport time to move the pallet from the main conveyor loop to the robot workplace, and T3 is the time for job execution.

Upon periodic interrogation, the entity coordinating OH execution – the PLC – checks the status of all resources, which acknowledge being *available* or *failed*. The ultimate interrogation checks only the state of one resource – the one for which a current operation of an OH was scheduled; during this exchange of information, the PLC is informed whether the resource is *available*, *failed* or valid yet unable to execute the requested OH operation upon the product due to components missing in its storage (*no stock* status).

3. MANAGING RESOURCE BREAKDOWN/RECOVERY

When the **failure** status of a resource is detected, the *FailureManager* is called, executing a number of actions according to the procedure given below (Fig. 3):

1. Stop immediately the transitions of executing OH, i.e. the circulation of *products_on_pallets* in the cell; production continues at the remaining valid resources (robots).
2. Update the resource holons with the new states of all robots.
3. Read Order Holons currently in execution in the cell.
4. Evaluate all products if they can still be finished, by checking the status of each planned OH:
   - if the OH was in the failing robot station, mark it as failed and evacuate its *product_on_pallet*;
   - if the OH is in the system, but cannot be completed anymore because the failed resource was critical for

this product, mark it as failed and evacuate its *product_on_pallet*;

- if the OH is not yet in the system, but cannot be completed due to the failure of the resource which is critical for that product, mark it as failed (n_e .is the total number of such OH).

5. For the remaining $n'_{wip} = n_{wip} - n_{fail}$ schedulable OH in the system, locate their *products_on_pallets* and initialize the transport simulation associated to the current operational configuration of the system. Authorise the n'_{wip} OH to launch Contract Net Protocol-based negotiations (HBM) with the remaining available Resource Holons for re-scheduling of their associated operations. n_{wip} are the OH currently introduced in the system (in the present implementation, $n_{wip} \le 5$), and n_{fail} is the total number of OH currently in the system, which cannot be finished because they need the failed resource at some moment of the execution (Hsieh, 2008).
6. Run the Global Production Scheduling algorithm for the $N - n_{fin} - n_{wip} - n_e$ OH not yet introduced in the system, where a number of N OH was scheduled in total and n_{fin} OH were finished.
7. Delete the orders stored on the system and transfer the updated orders to the system
8. Resume *product_on_pallet* transfer within the transport system (allow OH transitions in the system).

It might happen that a failed robot gets fixed before the current manufacturing cycle is finished. In this **recovery** case, the cell regained the ability to run at full capacity but the lined up orders do not make use of this fact, as they are managed by the system in a degraded mode.

The procedure of rescheduling back the Order Holons is virtually identical to the one used in case of failure; the main difference is that none of *the products_on_pallets* being currently processed need to be evacuated since there is no reason to assume they could not be completed. Any orders that were marked as failed due to temporary resource unavailability are now untagged and included in the APO list for scheduling at the horizon of the rest of batch, as they may be manufactured again due to resource recovery (Lastra and Delamerm, 2006).

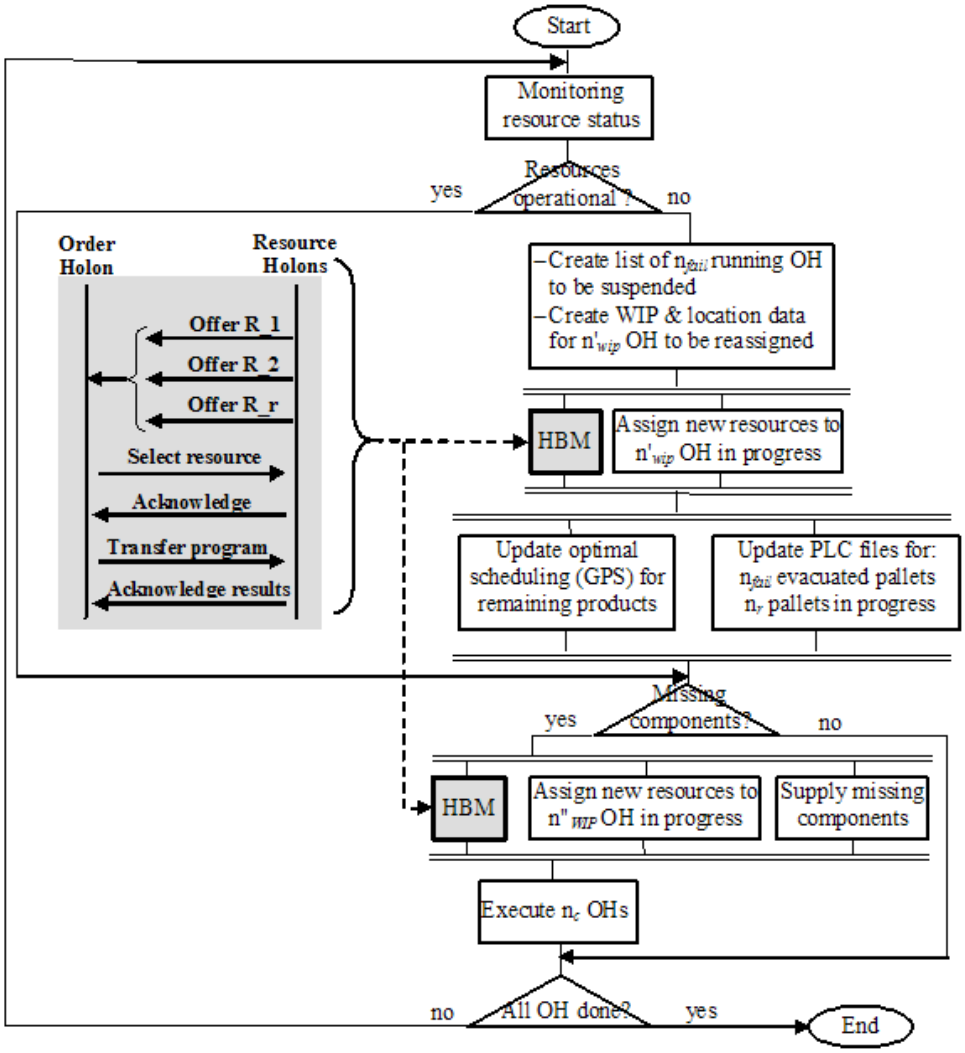

Fig.3. Dynamic OH rescheduling at resource failure/storage depletion with embedded CNP job negotiation (monex)

4. 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 similaritys between a task run on a processor and a batch of orders executed in a manufacturing cell (both are preemptive, independent of other tasks or batches, have a release, a delivery date and an fixed or limited interval in which they are processed), the Earliest Deadline First (EDF) procedure was used to schedule new batches (rush orders) for the robotized assembly cell.

EDF is a dynamic scheduling algorithm generally used in real-time operating systems for scheduling periodic tasks on resources, e.g. processors (Sha et al., 2004). It works by assigning a unique priority to each task, the priority being inversely proportional to its absolute deadline and then placing the task in an ordered queue. Whenever a scheduling event occurs the queue is searched for the task closest to its deadline. A *feasibility test* for the analisys of EDF scheduling was presented in (Liu and Layland, 1973); the test shows that if: (1) all tasks are periodic, independent, fully preemptive; (2) all tasks are released at the begining 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

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

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

A *batch* or Aggregate Product Order list (APO) is composed of raw orders (list of products to be manufactured); this is why two different batches are independent. Nevertheless, there is a difference between a task and a batch of products: a task is periodic while a batch is generally aperiodic. This means that insted of testing the feasibility of assigning batches to the production system considering the equation above, one can use the following test: "for an ordered queue (based on delivery date) of n batches with computed makespans, if $\sum_{j=1}^{i} \text{makespan}_j \le \text{delivery\_date}_i, i = \overline{1,n}$, then the batches can be assigned to the production cell using EDF without passing over the delivery dates".

This EDF approach is used to insert **rush orders** in a production already scheduled by the GPS; the steps below are carried out for inserting a new production batch (rush order) during the execution of a previously created sequence of Order Holons (see Fig. 4):

0. Compute the remaining time for finishing the rest of the current batch (if necessary).
1. Insert new production data: product types, quantities, delivery dates.
2. Separate products according to their delivery date.
3. Form the entities "production batches" (a *production batch* is composed of all the products having the same delivery date).
4. Generate raw orders inside the production batches (APO lists).
5. 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).

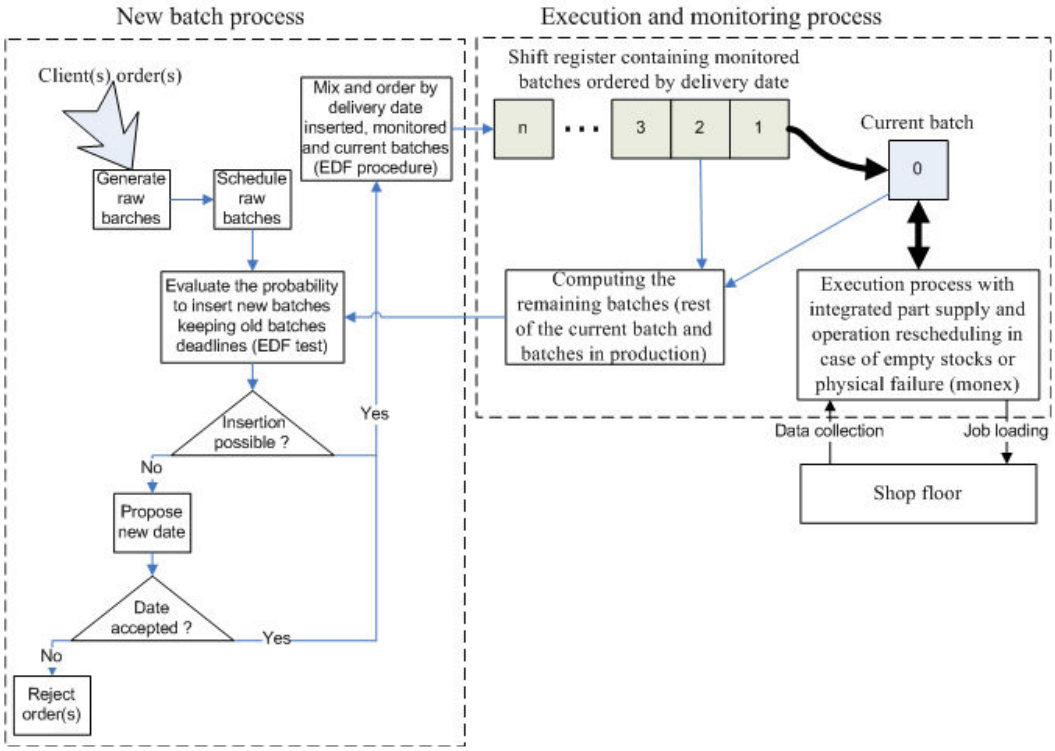

Fig. 4. Add rush order diagram and integration with dynamic job re-scheduling based on CNP negotiation (monex)

6. Analyse the possibility of allocating the batches to the cell using the EDF and second equation for feasibility test.
7. Allocate batches to the system according to EDF.
8. Resume execution process with new scheduled OH.

In this mechanism for managing the changes in production orders, an i*nserted batch* is a batch that arrives while another one is in execution. A *monitored batch* is one whose orders are scheduled and assigned to the cell (it has a priority and is waiting to enter execution). A c*urrent batch* is that executing.

The capability of adding rush orders to production needs a new entity, the **batch**. Thus, 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 (Table 1).

Table 1. 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)

Because batch execution is interruptible (preemptive system), new batches can be introduced exactly when they arrive. The insertion is triggered by "new order" events; a real-time acceptance response is provided (via ERP level) to the client if the rush order can be executed at requested delivery date.

5. CONCLUSIONS

The distributed control solution was implemented, tested and validated on a real manufacturing structure with 6 industrial assembly robots and 4-axis CNC milling machines, using the holonic approach. This development platform was recently created in the University Politehnica of Bucharest (Fig.5).

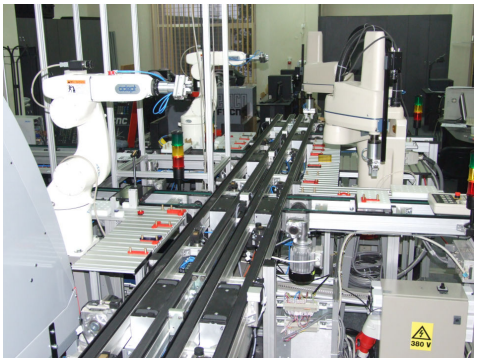

Fig. 5. Layout of the manufacturing cell with holonic control

The control is structure is fully operational, both in the normal hierarchical mode and upon switching automatically to the heterarchical one in response to discussed changes.

Production scheduling at batch level was implemented and tested using the EDF method; Fig.6 shows the results obtained when two new batch orders $T_2 4 = (4, 17)$ and $T_2 5 = (1, 3)$ are received at time T = 2 after the execution of three planned batches: $T_1 1 = (2, 18)$, $T_1 2 = (3, 20)$, $T_1 3 = (7, 11)$ 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).

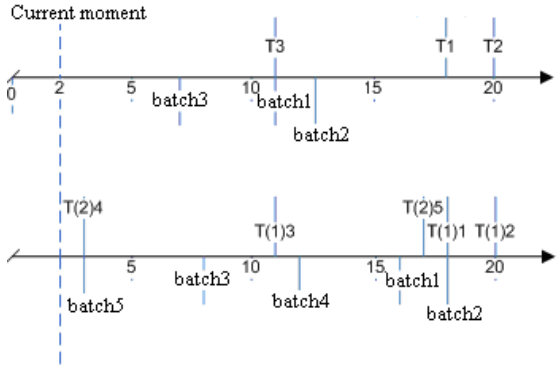

Fig .6. Inserting new batches among executing ones with the EDF algorithm

REFERENCES

Borangiu, Th., Gilbert, G., Ivanescu, N. and A. Rosu (2008). Holonic Robot Control for Job Shop Assembly by Dynamic Simulation. In: *Proc. of the 16th Mediterranean Conference on Control and Automation – MED'08*, June 25-27, 2008, Congress Centre, Ajaccio, France.

Brussel, H.V., Wyns, J., Valckenaers, P., Bongaerts, L. and P. Peeters (1998). Reference Architecture for Holonic Manufacturing Systems: PROSA, *Computers in Industry, Special Issue on Intelligent Manufacturing Systems*, **37**, 3, 255 – 276.

Fu-Shiung Hsieh (2008). Holarchy Formation and Optimization in Holonic Manufacturing Systems with Contract Net, *Automatica*, **44**. 4., 959-970.

Groover, M. (1987). *Automation, Production Systems and CIM*. Prentice-Hall.

Kusiak, A. (1990). *Intelligent Manufacturing Systems*, Prentice Hall, Englewood Cliffs, New York.

Lastra, J. and I. Delamerm (2006). Semantic web services in factory automation: Fundamental insights and research roadmap, *IEEE Trans. on Industrial Informatics*, **2**, 1-11.

Leitao, P. and F. Restivo (2006). ADACOR: A holonic architecture for agile and adaptive manufacturing control, *Computers in Industry,* **57**, 121-130.

Liu, C.L. and J.W. Layland (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of ACM* , **20**, 46-61.

Sha, L. et al. (2004). Real Time Scheduling Theory: A Historical Perspective, *Real-Time Systems*, **28**, 101-155.

Upton, D. (1992). Flexible Structure for Computer Controlled Manufacturing System. *Manufacturing Review*, **5,** 58–74.

Valckenaers, P., Van Brussel, H., Bongaerts, L. and J. Wyns (1994). Results of the holonic control system benchmark at the KULeuven, In: *Proceedings of the CIMAT Conference (CIM and Automation Technology)*, 128-133. Rensselaer Polytechnic Institute, Troy, New York.