Engineering Applications of Artificial Intelligence 22 (2009) 505–521

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

An implementing framework for holonic manufacturing control with multiple robot-vision stations

Theodor Borangiu [a,*], Pascal Gilbert [b], Nick-Andrei Ivanescu [a], Andrei Rosu [a]

[a] University Politehnica of Bucharest, Faculty of Automatic Control and Computer Science, Department of Applied Informatics, 313, Spl. Independentei, Sector 6, RO-060032 Bucharest, Romania

[b] Ecole Polytechnique Fédérale de Lausanne, Laboratoire LSRO, STI SMT-GE, Bâtiment 3141, Station 17, CH 1015 Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 14 April 2008
Received in revised form
2 March 2009
Accepted 5 March 2009
Available online 17 April 2009

Keywords:
Holonic manufacturing
Semi-heterarchical control
Robotics
Real-time vision
Applied AI

ABSTRACT

The paper describes a holonic control architecture and implementing issues for agile job shop assembly with networked intelligent robots, based on the dynamic simulation of material processing and transportation. The holarchy was defined considering the PROSA reference architecture relative to which in-line vision-based quality control was added by help of feature-based descriptions of the material flow. Two solutions for production planning are proposed: a knowledge-based algorithm using production rules, and an OO resolved scheduling rate planner (RSRP) based on variable-timing simulation. Failure- and recovery-management are developed as generic scenarios embedding the CNP mechanism into production self-rescheduling. Aggregate Order Holon execution is realized by OPC-based PLC software integration and event-driven product transportation. The holonic control of multiple networked robot-vision stations also features tolerance to station computer- (IBM PC-type), station controller- (robot controller), quality control- (machine vision) and communication- (LAN) failure. Fault tolerance and high availability at shop-floor level are provided due to the multiple physical communication capabilities of the robot controllers, to their multiple-axis multitasking operating capability, and to hardware redundancy of single points of failure (SPOF). Implementing solutions and experiments are reported for a 6-station robot-vision assembly cell with twin-track closed-loop pallet transportation system and product-racking RD/WR devices. Future developments will consider manufacturing integration at enterprise level.

1. Introduction

A *networked robotized job shop assembly structure* is composed by a number or robotic resources, linked by a closed-loop transportation system (closed-loop conveyor). The final products result by executing a number of mounting, joining and fixing operations by one or several of the networked robots. The set of specific assembling operations is extended to on-line part conditioning (locating, tracking, qualifying, handling) and checking of relative positioning of components and geometry features. These functional extensions are supported by artificial vision-merging motion control tasks (*Guiding Vision for Robots*—GVR) and quality control tasks (*Automated Visual Inspection*—AVI). Real-time machine vision is used to adjust robot paths for component mounting or fastening, to check for proper geometry and pose of assembly components, and to inspect the assembly in various execution stages (Borangiu, 2004).

Traditional networked assembly structures have either a *hybrid* or *heterarchical* architecture. The first one, derived from the hierarchical architecture, allows cooperation and sharing of information between lower-level (robot) controllers; a supervisor initiates all the activities and then the subordinates cooperate to perform them. The second is formed by a group of independent entities called *agents* that bid for orders based on their status and future workload. There is no master–slave relationship; all the agents including the manager of a particular order are bidding for it. Due to the decentralized architecture, the agents have full local autonomy and the system can react promptly to any change made to the system. However, because the behaviour of an order depends on the number and characteristics of other orders, it is impossible to seek global batch optimization and the system's performance is unpredictable. In order to face resource breakdowns, networked assembly structures should use robot controllers with multiple-network communication facilities allowing for fault tolerance: data saving and task redistribution.

There is currently a new trend in manufacturing control to apply the principle of *holons* in industrial-networked robotics. The interpretation of the holon as a *whole particle* proposes an entity which is entirely stand-alone or supreme as is (a whole),

* Corresponding author. Tel.: +40 21 402 93 14; fax: +40 21 317 09 12.
E-mail address: borangiu@cimr.pub.ro (T. Borangiu).

0952-1976/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2009.03.001

but belongs to a higher order system as a basic individual part (a particle). If a limited number of parts (holons) fail, the higher order system should still be able to proceed with its main task by diverting the lost functionality to other holons (Ramos, 1996; Deen, 2003).

Based on Koestler's concept, the following definitions, established by the Holonic Manufacturing Systems (HMS) consortium (Van Brussel et al., 1998) were accepted and used in the present project:

- *Holon*: An autonomous and co-operative building block of a manufacturing system for transforming, transporting, storing and/or validating information and physical objects. It consists of an information- and physical-processing part. A holon can be part of another holon.
- *Autonomy*: The capability of an entity to create and control the execution of its own strategies.
- *Cooperation*: A process whereby a set of entities develops and executes mutually acceptable plans.
- *Holarchy*: A system of holons that can cooperate to achieve a goal or objective. The holarchy defines the basic rules for cooperation of holons and thereby limits their autonomy (Wyns et al., 1997).
- *Holonic manufacturing execution system (HMES)*: A holarchy integrating in (custom-designed) software architecture the entire range of manufacturing tasks from ordering to design, production.
- *Holonic attributes*: Attributes of an entity that make it a holon. The minimum set is thus autonomy and cooperativeness (Bongaerts et al., 1996, 1998; Markus et al., 1996; Morel et al., 2003).

Based on the PROSA reference architecture, several research groups developed holonic control frameworks to operate parts of a manufacturing system (e.g. part processing on multiple machine tools), but only a few considered material-handling tasks (Liu and Layland, 1973) and transportation. The negotiation scenario, proposed by Usher and Wang (2000), for the cooperation between intelligent agents in manufacturing control, or the "*n* products on *m* machines" KB scheduling algorithms, proposed by Kusiak (1990), are limited to production planning and job scheduling, and do not consider: (a) the constraints imposed by the transportation system (e.g. cell conveyor); (b) the need to qualify (recognize, locate, check for collision-free robot access and correct robot points for part mounting) assembly components; and (c) verify the assembly in different execution stages.

The proposed holonic control framework faces the difficulties arising when moving from control theory to practice, because the (particular) cell conveyor is modelled, parameterized and integrated in the generic job scheduling, and the material components (parts, assembly) are described by task-dependent features that are extracted through image processing at execution time in view of material qualifying and product inspection. In order to face resource break-downs, the job shop assembly structure using networked robot controllers with multiple-LAN communication is able to replicate data for single-product execution and batch production planning and tracking (Cheng et al., 2006).

The holonic implementing framework will be exemplified on a discrete, repetitive production system with part machining, robotized assembling and visual quality control capabilities. The management of changes is imposed at resource breakdown, storage depletion and occurrence of rush orders. The expected performances of the system are: high productivity (selectable cost functions: throughput, machine/robot loading, time), high accuracy of operations, adaptability to material flow variations and shop-floor agility.

The functionalities below were imposed in the development of the holonic control system:

- adaptability and quick reaction in face of production changes (rush orders);
- assistance to mounting (GVR) and product quality control (AVI) by real-time machine vision; visual robot guidance during precision assembly and visual in-line geometry control of products are technical requirements imposed to the framework to increase the quality of services performed;
- efficient (optimal) use of available resources (robot, CNC machine tools) in normal operating;
- stability in face of disturbances (resource failures, storage depletion).

The paper describes: (i) the design of a distributed control architecture for networked assembly robots, automatically switching between a hierarchical and heterarchical operating mode; (ii) the definition of the holarchy and set up of the holon structures; (iii) the design and software implementing of operation scheduling algorithms and HMES integration; (iv) the solution adopted for fault tolerance to robot and CNC breakdown (dynamic job reconfiguring instead of reprogramming) and high availability (redundancy in SPOF hardware and inter-device communication paths); (v) the definition of generic part qualifying operations as framework features applying AI through real-time, high-speed image processing.

In the first part of the paper a general framework is proposed to implement a holonic manufacturing control solution. The second part of the paper exemplifies the framework by a case study using a particular industrial technology (Bosch-Rexroth conveyor, Adept Technology robots and vision, etc.).

The paper is structured as follows: in Section 2 the proposed system architecture at process level, the process simulation tool, and integration at enterprise level are presented; Section 3 describes the semi-heterarchical control solution and the adopted holarchy; Section 4 discusses failure/change and recovery operating modes in which production is dynamically rescheduled by pipelining a resource bidding mechanism of CNP type (at shop-floor horizon) with a global scheduler (at aggregate batch horizon); Section 5 explains how ordered assembly plans (Order Holons) are mapped into conveyor routing PLC commands for product execution and tracking; Section 6 reports implementing issues, high availability solutions and experimental results.

2. System architecture and dynamic simulation tool

To be competitive, manufacturing should adapt to changing conditions imposed by the market. The greater variety of products, the possible large fluctuations in demand, the shorter lifecycle of products expressed by a higher dynamics of new products, and the increased customer expectations in terms of quality and delivery time are challenges that manufacturing companies have to deal with to remain competitive. Besides these market-based challenges, manufacturing firms also need constantly to be flexible and adapt to newly developed processes and technologies and to rapidly changing environmental protection regulations. A proposed solution to this problem is a distributed architecture, in which information entities, having manufacturing counterparts, cooperate to solve together the assigned tasks. Such a type of holonic manufacturing control architecture is presented in this paper.

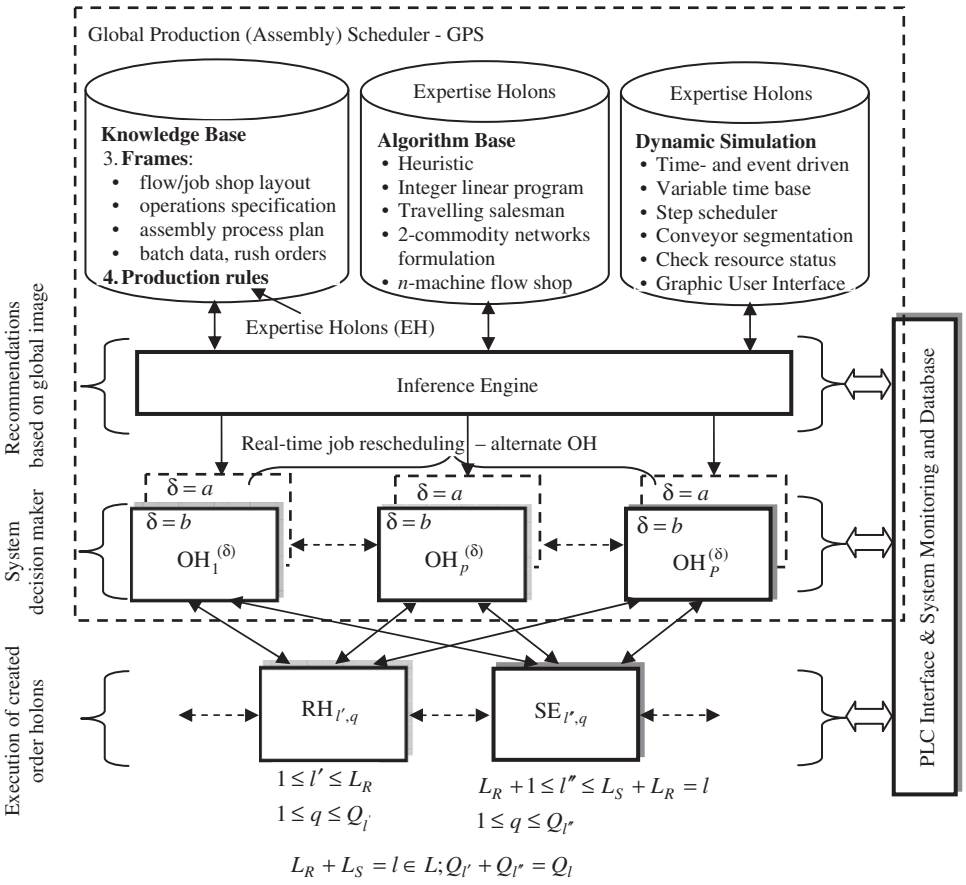

Fig. 1. Entities and their functions in the proposed knowledge-based holonic assembly system.

For manufacturing structures with networked resources (robots, machine vision, machine tools), as is the generic class of product assembling with in-line part machining and interphasic quality checking, a *semi-heterarchical distributed control architecture* is proposed in which the organizational control is arranged on two levels, referred to as global and local (Rahimifard, 2004). The global level assumes the responsibility for planning and coordination of the 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 *assembling robot team*). The *team-based manufacturing* paradigm (based on a wide range of criteria: similarity of activities, definition of business processes, planning and control requirements (Leitao and Restivo, 2006, Barata, 2005)) was adopted to provide the flexibility, agility and responsiveness required to cope with volatility of production demands.

In this semi-heterarchical control architecture, 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) 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 treated as recommendations for the decision-making entities—the OHs. 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, being replaced by alternate schedules created from resource (Robot Controllers) offers mutually agreed by cooperation mechanisms (*heterarchical operating mode*). The holonic manufacturing control automatically switches between these two modes.

The flow of information between control units in the designed system is always bi-directional due to the decentralized nature of the architecture and the application of the holonic concept of cooperation. This production control structure is derived from a generic manufacturing one, formed by four types of entities, as shown in Fig. 1 (Babiceanu et al., 2004; Borangiu et al., 2007; www.rvholon.cimr.pub.ro):

1. A *Global Production Scheduler*, generating production plans for all batch products in the form of OH. An algorithm base is embedded into a knowledge-based system (KBS). A dynamic production simulator acts as time-driven validation tool for job scheduling. An inference engine in the KBS controls, according to a forward chaining control strategy, the procedures of: triggering production rules, switching between algorithmic and simulation stages during planning, and production updating at failure/change occurrence (Bongaerts et al., 1996). The GPS uses in its production rules, algorithmic base, respectively variable time step scheduler classes of Expertise Holons (EH) to generate OHs. EH have embedded expertise on process organization (products), execution (production rules, resources), and optimization (machine/robot utilization, resource load balancing, flow time, a.o.).
2. A *PLC interface* for Production Execution & Monitoring and *Database* (PLCID) *access* entity, responsible for managing the execution and monitoring the production jobs (sequence of

OH) and availability of resources in the system, and for keeping track of products moving between stations.

3. A layer of *Order Holons* ($OH_p^{(\delta)}$, $1 \leqslant p \leqslant P$) of variable depth (basic OHs, $\delta = b$ i.e. production plans off-line computed for the P final products and alternate OHs, $\delta = a$ i.e. production plans rescheduled in real time at system failure/production change and recovery.
4. Two types of Resource Holons:
   - *Robot/machine tool* (or material conditioning) *holons* ($RH_{l',q}$), formed by all robots, grippers, and tools together with their controllers, responsible for machining, mounting, and fastening parts, and for moving their arm-mounted cameras in picture-taking points where the products are visually inspected, respectively machines, clamping devices and tools (Iwamura et al., 2005).
   - *Sensory* (material tracking and checking) *holons* ($SE_{l'',q}$), formed by all machine vision systems (*stationary* – down looking and *mobile* – arm mounted), and magnetic code RD/WR devices, respectively used for component position and geometry control and product tracking on pallets.

In this representation, the following notations were adopted: O = set of all operations (machining, assembly, conditioning); P = set of all assemblies (final products); OA_p = set of operations for assembly A_p, $p \in P$; L_R = set of all RH; L_S = set of all SE; L = set of all resource types; $Q_{l'}$ = set of resources of type $l', l' \in L_R$; $Q_{l''}$ = set of resources of type $l'', l'' \in L_S$, Q_l = set of resources of type $l, l \in L$.

Fig. 2 shows the integration of this holonic assembly system at enterprise level, in a SOA concept.

Service-oriented architectures (SOA) support the development of more powerful reconfiguring and interoperability tools, using more complex self-organization and learning techniques (Babiceanu et al., 2004). Service composition in multi-agent manufacturing systems is the combination of single services and all the interaction patterns between them. The evolution and re-configurability of this class of production structure is facilitated using multi-agent systems supported by *web services technology* since it is possible to add, remove and modify resources and services without interrupting processes.

The SOA for enterprise manufacturing management and control integrates four areas: (1) offer request management; (2) management of client orders; (3) order- & supply-holon (OH, SH) management; (4) OH execution & tracking). The first area is responsible for generating offers in response to requests, based on: product knowledge (embedded in Product Holons – PH), resource capabilities (from RH data), supply constraints and activities planning (CAPP). Once received customer orders, they are interpreted, validated and mapped into aggregate production orders (APO) at ERP level. APO is the input to the GPS in Fig. 1,

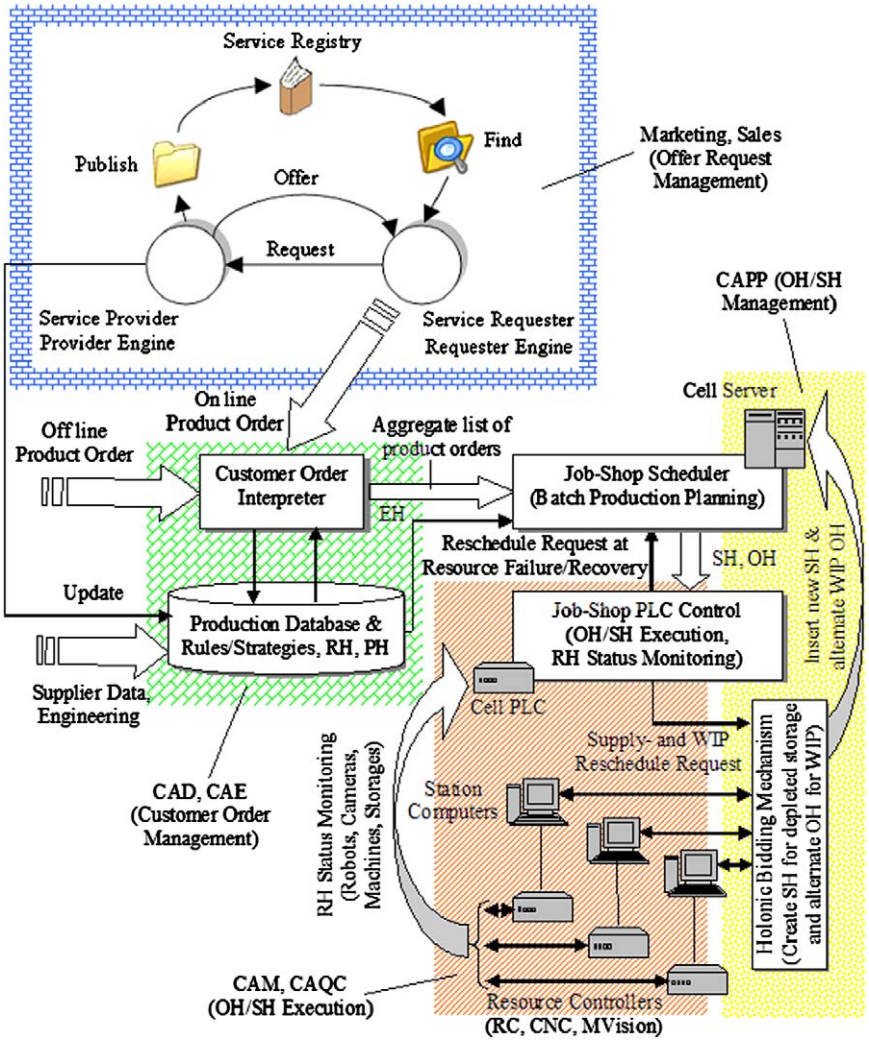

Fig. 2. SOA integrates job-shop, team-based manufacturing with holonic robot control.

which, generates the lists of supply- (SH) and order-holons (OH).

The information part of a SH describes the part supply parameters (types, number and placement of parts to be fed to local robot storages, strategies to retrieve components from central storages, transport data: part placement on supply pallets, path data for supply pallets). The physical part of SH is given by the pallets carrying assembly components and transporting them from two central ASRS to local storages in all workstations. SH are ordered, to minimize global supply time; executing an SH starts by routing an empty supply pallet towards one central ASRS from where a robot retrieves parts, then directs the full supply pallet to a workstation where a local robot empties it by placing the parts in the local storage.

Each OH passes through three phases: (1) *waiting*: a product order was created, scheduled within the batch and inserted in the multiple-batch planning; it waits to be introduced in the system; (2) *in execution*: a pallet is associated to the product to be realized; the pallet will be automatically routed to the resources that have been selected to carry out the operations on the product; (3) *finished*: once executed the pallet leaves the system, the product is separated from the pallet that will be reused for another product.

In case a rush production order is received, the OH management in area 3 evaluates its feasibility considering the current execution stage of already scheduled APO; if the insertion is possible, the existing OH recommendations will be updated by the GPS with the new rush order, without interrupting execution of production. In case a resource breakdown or a depletion of local storage is detected, there will be generated alternate OHs, respectively a new SH by direct negotiation among valid (operational and having necessary components) Resource Holons, until system recovery (Lusch et al., 2008).

Activities in area 2 are at ERP level, activities in area 3 link the ERP level for production planning and engineering (CAD, CAE) with production execution, tracking and quality control (CAM, CAQC) performed in area 4. Thus, area 3 integrates the business (ERP) and process levels of the HMES at enterprise level, based on the SOA approach. Although the HMES development is a global goal of the scientific project, this paper discusses only contributions to MES implementing in areas 3 and 4.

One important aspect considered in the design and implementing of APO through OH and SH definition, scheduling, execution and tracking was the development of generic solutions rending the manufacturing control structure agile at process level. In this respect, the KB job scheduler (GSP), operation & precedence definition for products, production rules, mapping of OH and SH in PLC format for *product_on_pallet* routing in the manufacturing structure, and the contract-based negotiation mechanism among resource controllers are generic. Also, the PH, RH and OH object classes are generic, such as the functionalities of the EH. Of course, the resources must be parameterized for specific manufacturing structures, and similarly the terms of the negotiation contracts; a decoupling between OH & SH *creation* (planning of pallet routing) and *execution* on a particular cell conveyor was done by parameterization of the transportation system (speed, connectivity and distances between workstations). Heterogeneous resources may be integrated in the HMES because manufacturing controllers (e.g. robot, machine) are agentified by *software wrappers* hiding the details of each component. The wrapper acts as an abstract machine to the agent supplying primitives that represent the functionalities of the physical component and its industrial controller. The agents (embedded in IBM PC generic station computers) access the wrapper using a local software interface (proxy) where all the services of interest are defined.

One central component of the system's software architecture is a *simulator*, designed as a multi-usage tool: (i) Production Configuration Manager (allows the user to download/configure batch production orders, production plans and resource capabilities); (ii) Step Scheduler (time-driven production planner and validation tool—outputs OH sequence on a cost base: throughput, resource load, total time); (iii) Failure and Recovery Manager (2-stage time-driven job re-planning: (s1) re-allocate resources for products in progress—introduced in the system but unfinished; (s2) redefine batch production and reschedule remaining products—redefine OH and validate their execution); (iv) Production

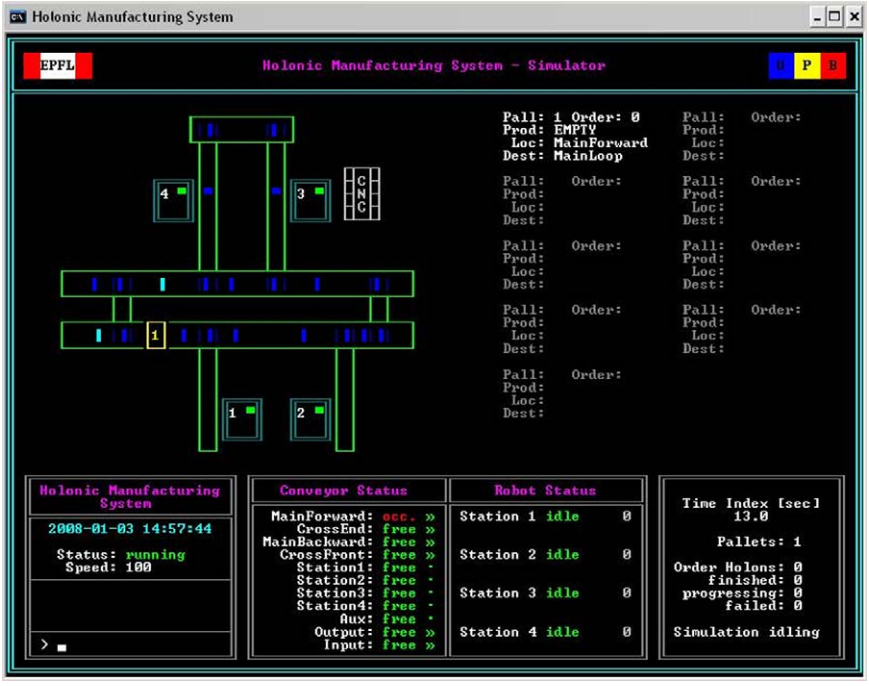

Fig. 3. The dynamic simulator is based on a process-oriented (*product_on_pallet* transport) Graphic User Interface.

Tracking (event-driven pallet tracking in conveyor segments and workstations).

The dynamic user-configurable, interactive simulator was integrated in the semi-heterarchical production control software (aras 3 and 4—product scheduling, execution and tracking), being adapted to the real transportation system (see Fig. 3).

The entire system is dotted by stopping mechanisms allowing keeping a pallet at a certain fixed position while other pallets may move. The stoppers block a pallet as it reaches the stopper's position while the conveyor is still running, ensuring that other pallets are still moved. At each intersection an elevator serves as stop–switch combination to stop first an approaching pallet and then transport it to an intersecting conveyor (either a lateral branch or to the second main conveyor). These main elements are all present in the simulation and work exactly as in the real system. The software uses a *transportation time matrix* (TTM), which was created by measuring the actual time used by the real system to transport a pallet from one point of interest to another (in general from one stopper or elevator to the next). The simulation's smallest time index corresponds to one advancement step of a pallet and is 0.5 s.

The transport simulation is also used to in-line validate global production schedules. However, there is a fundamental difference in using the core routines developed to realize the correct pallet transportation:

- In the case of *visual simulation*, the routines run in a timed mode. This means that after each iteration of the main program loop a timer stops the program and waits until the smallest time index of 0.5 s has passed by and only after that allows another iteration of the main program loop. The result of this pause is a smooth running in 0.5 s steps of the simulation, which, in combination with the measured transportation time matrix or driven by events (signals received by the PLC from sensory devices), reproduces the behaviour of the real system.
- When the routines are used to solve the *scheduling problem*, they "transport" the pallets in the system with an infinitely high speed (limited only by the computer's calculation speed). As soon as one iteration of the transport functions has finished, the next one starts. Since none of the belt segments or the TTM values is changed in this simulation, the resulting time indexes still correspond exactly to 0.5 s and may be used to define the production schedule (the OH list).

The simulator's working principle and role in scheduling are further detailed: at each iteration, the simulator's functions check all pallets that are currently in the system. A pallet is transported one step if the conveyor segment is running and if there is no active stopper or elevator at the pallet's current position. If a pallet happens to be in the way of another, the latter bumps into the first one, as it would be the case in reality. With this basic operation mode all pallets may be transported freely in the system.

Fig. 4 shows the 6-robot assembly cell with self-supply used for to develop the holonic control.

Constraints imposed by the cell architecture ask for another control layer which ensures that blocking situations do not occur while the system is operational. Since there are four robot-machine tools stations, there is no need to have more than five pallets simultaneously circulating in the cell (one exits the cell).

The robotic stations are interconnected by a closed-loop twin-track, pallet-based power-and-free conveyor with four linear bi-directional derivations that move subassemblies fixed on pallets

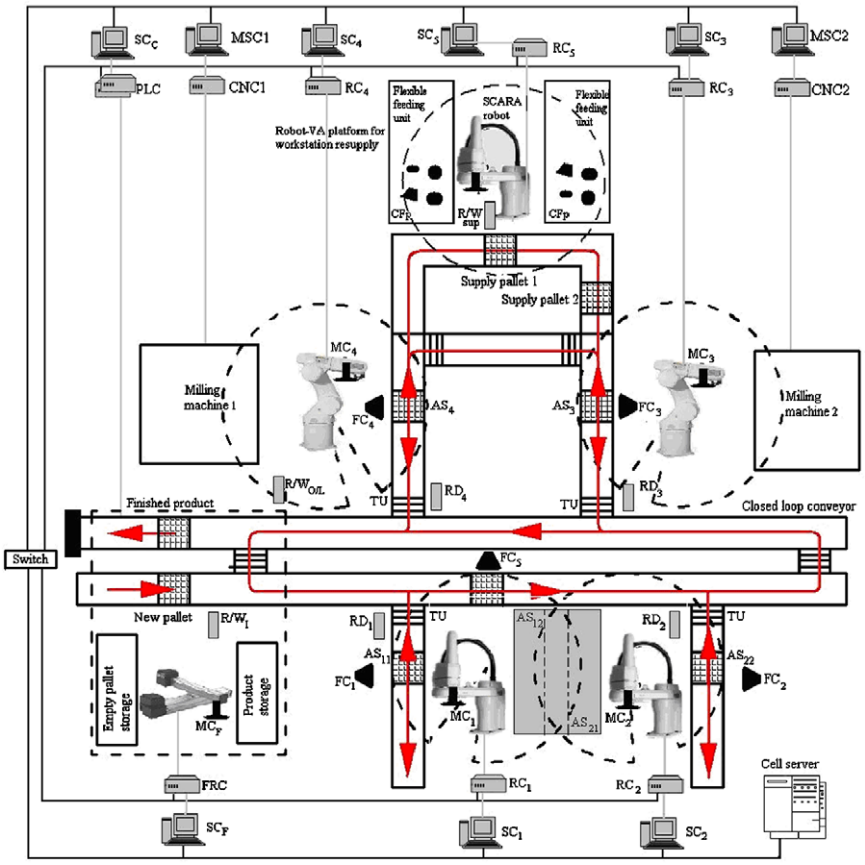

Fig. 4. Holonic manufacturing system with self-supply of assembly parts: 1 Cartesian robot for pallet feeding, 2 SCARA and 2 vertical articulated for assembly, each one with 2 cameras: fixed (FC_i) and mobile (MC_i)), tending 2 CNC milling machines. Products are tracked by magnetic code read/write devices (MC R/W_i).

in single or double access production stations: $AS_i, 3 \leqslant i \leqslant 4$, $AS_{ij}, 1 \leqslant i \leqslant 2$, $1 \leqslant j \leqslant 2$. The robot controllers RC_i have the ability to communicate via digital I/O lines and Ethernet with a PLC. The PLC's main task is to operate the conveyor system and to manage the execution of production, and supply, i.e. OH and SH. Each controller is connected to a Station Computer SC_i used to: edit and debug application programs, create vision models, monitor the robot's activity, keep a log, and replicate the resource's functionalities.

The PLC also assigns tasks to robots upon resource negotiation, and receives task-termination signals and status information from robot controllers. The PLC serves thus as *general interface* for scheduling, execution, and failure/recovery management of production and supply at shop-floor level.

3. Semi-heterarchical manufacturing architecture and holarchy

The distributed control solution proposed in this project provides a set of functionalities rending the material-conditioning cell flexible, rapidly reacting to changes in client's orders (batch size, type of products, alternate technologies, rush orders, updated programs), and fault tolerant to resources getting down temporarily. In fact, the holonic control architecture proposed follows the key features of the PROSA reference architecture (Van Brussel et al., 1998; Valckenaers et al., 1994), extended with:

- Automatic switching between *hierarchical* (for efficient use of resources and global production optimization) and *heterarchical* (for agility to order changes, e.g. rush orders, and fault tolerance to resource break downs) production control modes.
- Automatic planning and execution of assembly component supply; automatic generation of self-supply tasks upon detecting local storage depletion,
- In-line vision-based part qualifying and quality control of products in various execution stages.
- Robotized material processing (e.g. assembling, fastening) under visual control/guidance.

As suggested by the PROSA abstract, the manufacturing system was broken down into three basic holons, the *Resource Holon*, the *Product Holon*, and the *Order Holon*. Each of these holons may exist more than once to fully define the manufacturing cell. Order Holons are created by the Global Production Scheduler from the aggregate list of product orders (APO) generated at ERP level (stage 2 in the enterprise SOA, Fig. 2).

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 as well as functionality. Beside the information part, holons usually possess a physical part, like the *product_on_pallet* for OH (Duffie and Prabhu, 1994).

The way in which different types of holons communicate with each other and the type of information they exchange depends on the nature of the manufacturing cell. Fig. 5 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 (with help of Expertise Holons—EH) and management of OH exactly as Staff Holons in the PROSA architecture do; in addition, the HolonManager interfaces the application with the exterior (maps OH into standard PLC file and tracks OH execution for user feedback). Since a single holon may be seen as a *class object* in the object-oriented programming environment (C# and .net 3.0 framework tools have been used), each of the three basic holon types was realized as a separate class. The instances necessary to define the manufacturing cell are then hosted by the class *Holons.HolonManager*. Each type is present as an array which may be scaled dynamically if necessary. Thus, the array of *Holons.Product* class instances assumes a size of existing product types; each element represents one product type with all necessary info to generate OH of this product type.

A Resource Holon holds information about cell resources. Any resource may have a number of sub-resources, which are also seen as

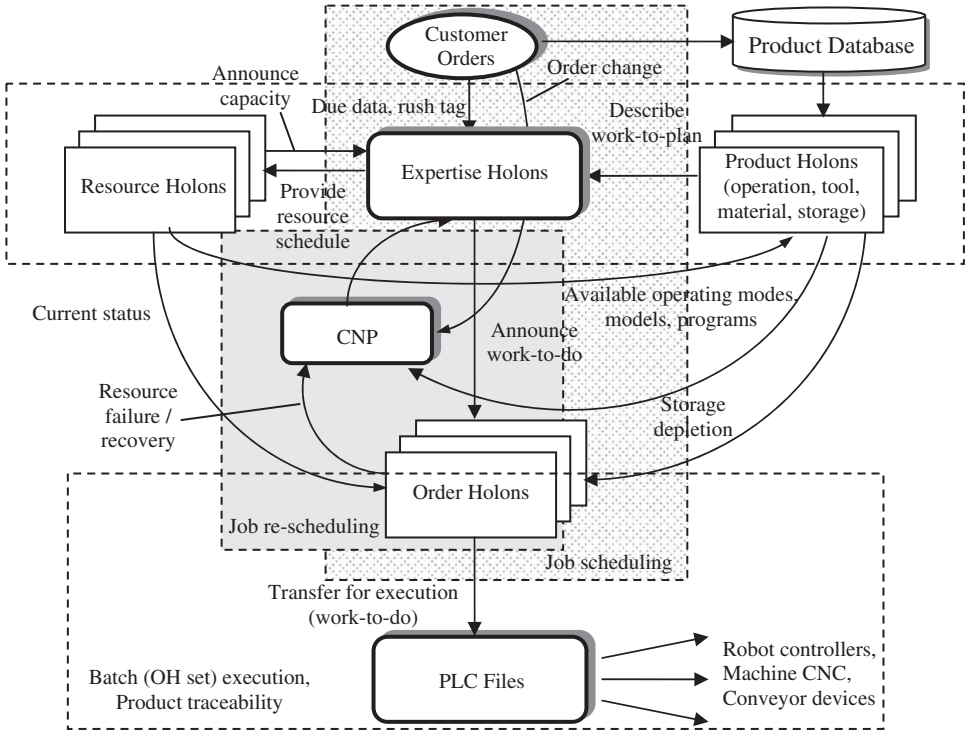

Fig. 5. Basic holon cooperation and communication structure in the semi-heterarchical control architecture.

holons. This project considers a global resource (e.g. *robot system*) with all its sub-resources as a holon without the distinction of sub-systems. The hardware part of this type of holon is the physical manipulator and controller with its functions. A permanent data exchange between hardware and software ensures that the actual RH status is accessible through its software representation:

A Product Holon holds information about a product type. Any type that may be produced in the manufacturing system and resource setup should be defined in a Product Holon. The fact that such a holon exists does not necessarily mean that the concerned product is being executed (e.g. assembled). Only the array of Order Holons will specify that something is manufactured and in what quantity. The product information is more than a theoretical description of the physical counter part but not directly associated with one individual physical item, unlike the Resource Holon (Leitao and Restivo, 2006).

Public Member Functions

Void **DefineRobot** (SubResource[] sub_resources)
Entirely defines a robot's sub-resources (operations).

bool **OrderHandOver** (short order_index, short sub_resource)
Completes the order hand-over to a resource and immediately starts the robot.

void **OperationCompleted** ()
Frees the resource and notifies the order that an operation has been completed.

void **AllocateTime** (ulong time of arrival, unit task time)
Allocates operation time on this resource.

void **SubResourceFailed** (string op_name)
Marks specific or all sub-resources as failed.

void **SubResourceRecovered** (string op_name)
Marks specific or all sub-resources as operational.

void **ResetHolon** (bool status_reset)
Resets the reservation-end-time and status (prepare for new run).

Public Attributes

Sub_Resource	**SubResources**
	Indexec class containing all available operations of this resource.

Properties

short	**HolonIndex** [get, set]
	Unique identifier corresponding to array position.
string	**Name** [get, set]
	Name of this resource spelled out.
short	**OperationCount** [get]
	Total number of operations installed on this resource.
RobotStatus	**Status** [get, set]
	Current operational status of this resource.
ulong	**MachineHours** [get]
	Total amount of time this resource was in use.
ulong	**ReservationEndTime** [get]
	Time index to which this resource will become available, if busy.
short	**OrderHolonIndex** [get]
	Identifyer of order holon currently processed by this resource.
short	**InProgressPointer** [get]
	Points to the operation which is currently in progress.
unit	**RemainingTime** [get, set]
	Time remaining until current operation is completed.

An Order Holon represents 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*. The information, data and processing methods embedded in an Order Holon are given below:

```
//Information about the assembly plan of a product:
  //String of characters for the OH name, identical to product name
string name
  //Number representing the index of the current order holon
  int holon_index
  //Integer representing the no. of operations to be done on current
    OH
  int operation_number
  //Vector with names of operations to be executed
  String[ ] operations = new String[maximum_operations]
  //Vector of precedences (e.g. let ibe the index of some operation
    //from the operations vector; then precendece[i] contains the
     index
    //of the operation that must be executed before the current one
    int[ ] precedence = new int[maximum_operations]

//Information about system resources:
  //Number of resources in the system
  int resource_number
  //Array storing information about operations executable on
    resources
  int[,] resources = new int[maximum_operations,
    maximum_resources]
  //Array storing the execution times of operations, on each
    resource
  //one operation may have different execution times on different
    resources
  float[,] operation_time = new float[maximum_operations,
    maximum_resources]

//Unique information resulting from the planning process:
  //Vector containing times at which empty pallets (OH starting
    execution)
  //must be introduced in the system
  float[] planning_time = new float[maxim_planning]
  //Vector containing the operations to be executed on the product
  //in the scheduled order
  int[] operation_planning = new int[maxim_planning]
  //Vector containing the resources which were associated to
    scheduled
  //operations
  int[] resource_planning = new int[maxim_planning]
  //Integer representing the number of scheduled operations
  int var_planning

//Information used during scheduling:
  //Index of the resource on which is currently the OH
  int actual_position
  //Status of the holon: scheduled or not
  bool holon_status
  //Status of each operation: not planned, selected for planning,
    planned
  int[] operation_status = new int[maximum_operations]
  //Resource on which is the current holon
  int current_resource
  //Operation executed at present on the current holon
  int current_operation
  //Time at which ends the execution of the current operation
    (different from
  //operation execution time)
  float remaining_time
  //Vector indicating whether current operation was chosen for
    planning or not
  bool[] operation_checked = new bool[maximum_operations]

//Processing methods for the Order holon class:
  //Constructor
  public order_holon()
  //Copying Constructor
  public order_holon(order_holon oh)
  //Operator = = indicates whether two holons are identical
  public static bool operator = = (order_holon oh1, order_holon
    oh2)
  //Operator ! = indicates whether two holons are different
  public static bool operator ! = (order_holon oh1, order_holon oh2)
  //Function checking whether a holon is valid for scheduling
  public bool valid_holon(int index_op)
  //Functiion checking whether transport operations must be
    inserted
  public bool test_transport(int index_op,int index_res)
  //Function which inserts an operation to be scheduled
  public void insert_operation(string operation_name,int
    index_resource,
  float prel_time, int prec)
  //Function which inserts a scheduled operation
  public void add_planned_op(int index_op, float time)
  //Function which returns the duration of the operation of specified
    name
  public float duration(int op)
```

Before production starts for a specific Aggregate List of Product Orders (APO created at ERP level), the OH exist only in electronic

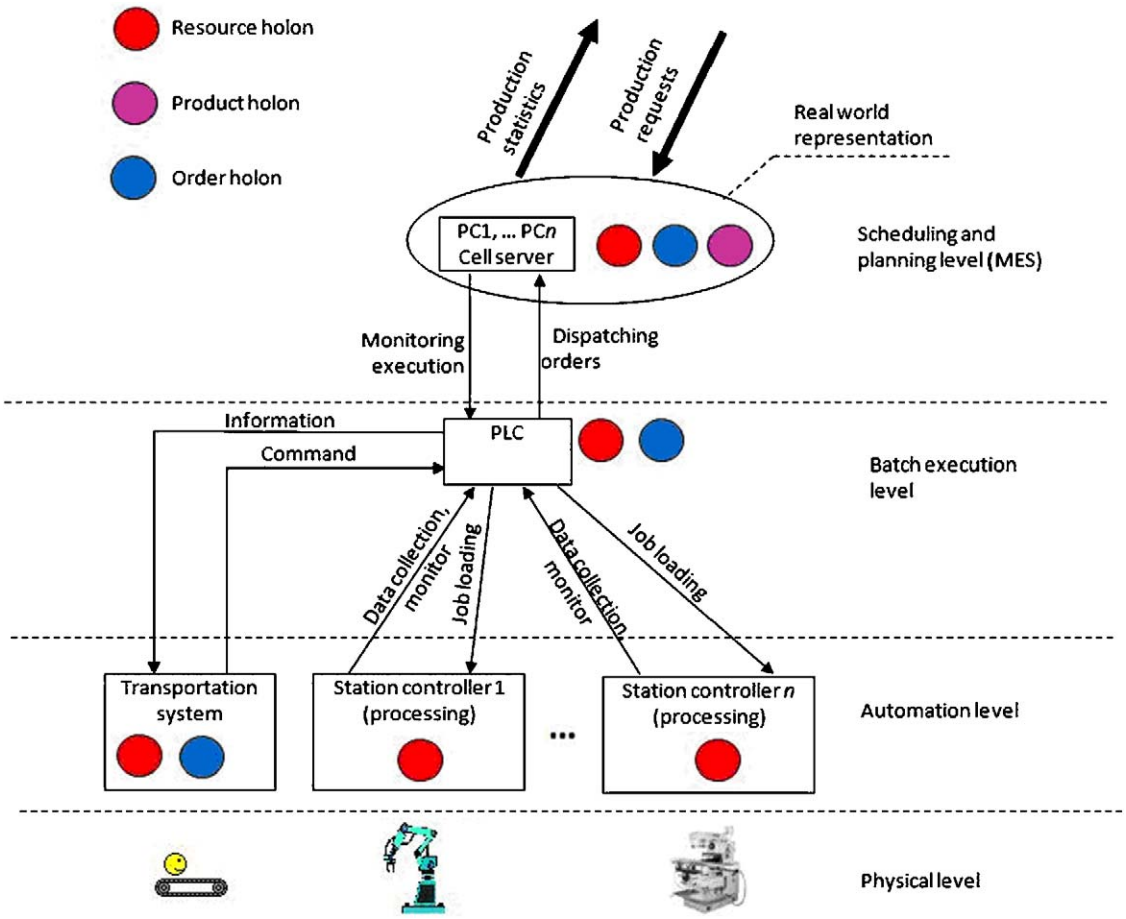

Fig. 6. Mapping between the (holonic) system and the physical architecture.

format; during production execution each OH develops on a pallet in the system; after completion, the item gets cleared from the exiting pallet and has now a physical form. OH are created, from raw orders (items in the APO list), which are based on the information stored in the Product Holon. If a certain product needs to be manufactured n times, then n identical raw orders are created first; When OH for these raw orders are created, the information differs for each OH in terms of robot stations, which need to be visited and the time at which they are visited (Onori et al., 2006).

Unlike the Product Holon, seen as a general, static description of a certain product type, which is rarely changed, the Order Holon is the actual realization of one item of a product type and undergoes many changes (of information as well as of physical nature) during manufacturing. An OH is represented by a *pallet carrier* with a unique identifier on it (magnetic tag), the *manufactured product* (on the pallet), and a *management program* running on the PLC communicating with resource controllers.

Because operation and resource indexes are used for implementation, indexes must conform to a standard structure and be at the same time globally consistent and valid; thus, operations will be indexed in increasing order, and resources added in time.

The mappings between the (holonic) system requirements and the functional architecture are included in Figs. 1 and 5. Fig. 6 describes the mappings between the functional architecture and the physical one (for the particular implementation). The real-world representation refers to the model (the software counterpart of the RH, PH and OH set) of the real-production system which exists at the planning level.

4. Job scheduling and management of changes

The problem of generating a schedule for an aggregate batch production and maintaining it despite changes in orders and resource failures is very complex in general, and was subject of concentrated development efforts during the past years (Pétin et al., 2007; Higuera and Montalvo, 2007).

A *basic process plan* (quasi optimal) is generated initially, upon receiving from the ERP level (customer order management in Fig. 2) an aggregate production order (raw orders), based on a method that can be selected from the following two: (i) knowledge-based scheduling, inspired by Kusiak (1990) reference research; (ii) Step Scheduler, acting as a Resolved Scheduling Rate Planner (Borangiu et al., 2007). 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 storages of robot stations and a list of Order Holons (OH) responsible for product execution.

An *assembly plan* $\mathrm{AP}_p^{(\delta)}$ of a product A_p is embedded in a resulting Order Holon (OH) as a vector of triplets, each specifying operation number o_i, processing time $t_i^{(\delta)}$ of operation o_i using assembly plan δ, and set of resources $R_i^{(\delta)}$ to process the operation o_i:$\mathrm{AP}_p^{(\delta)} = [\ldots,(o_i,t_i^{(\delta)},R_i^{(\delta)}),\ldots]$, $1 \leqslant i \leqslant f$, where $R_i^{(\delta)} = \{r_{1qi}^{(\delta)},\ldots, r_{5qi}^{(\delta)}\}$, $q \in Q_l, l \in L$, $1 \leqslant i \leqslant f$, r_{lq} = resource q of type l, $q \in Q_l$, $l \in L$. The basic assembly plan with o_f as final operation is denoted by $\delta = b$ (initial OH list), whereas $\delta = a$ are alternative plans ($b = 0$, $a = 1,2,\ldots$)—upon rescheduling requests triggered by a

rush orders, resource failure/recovery or missing assembly components. Experiments showed that $t_i^{(b)} \leqslant t_i^{(a)}$, $o_i \in O$.

4.1. Knowledge-based scheduling algorithm

The global production scheduling problem is of NP complex job shop type, for which the computing of an analytical solution increases rapidly with the problem's dimension (number of resources, number of operations executable by resources, number of product operations, etc.). Therefore, heuristic methods were chosen. The GPS entity has a global view of the production system, which allows optimizing the total production time by using a generic set of priority rules and equalizing the resource loads.

However, at resource failure or occurrence of a rush order, the GPS exits the heuristic algorithm and enters the inference engine of the KBS, where the procedure of selecting alternative plans is carried out.

The procedural knowledge of the KBS is in the form of *production rules*, triggered by Expertise Holons, from which three sets were used to generate Order Holons. Some rules in these sets stop the search of the inference engine and switch the control process back to the heuristic algorithm: SEL_ALG (selection of the appropriate scheduling algorithm), ALT_PLAN (starts and controls the mechanism of selecting alternative OHs for a job), and EVAL_RES (evaluates the computed assembly plans and decides upon rescheduling to improve the global quality of solution at batch level).

A criterion considered was to maximize the load of available resources, i.e. each of the four assembly robots should have a minimum of idling time. To achieve a maximum load of each robot, the conveyor system should never be jammed by any pallet carrying an item waiting to be processed by a robot.

Each individual item is being scheduled one step at a time. The process is initialized by generating a queue of all the items of all different products that must be manufactured. The queue is composed in such a way that all items of one product follow each other, and then the number of next products is added.

All possible operations of all items present in the system are collected. An operation is considered as *possible* if the preceding operations have been completed and if a resource is available to carry out the task. An operation may appear more than once in the list if it may be carried out on several robot stations. Once the list created, seven priority rules are applied in a defined order until only one operation remains. This operation is then scheduled on the related robot. The list of possible operations is updated and the priority rules are applied again. This process is being repeated until the list of possible operations is empty. Now the time index is increased to allow transportation and processing operations. As time passes, robots become free, thus the list of possible operations contains again elements. In that case, time is stopped and the priority rules are applied again to schedule the next operation. The planning algorithm is:

Step 0: set time index to zero, collect all possible operations based on robot status and predecessor constraints of all items, store them in list S (the list of possible operations)
Step 1: apply the following priority rules to select one operation
P0: if any of the operations belong to an item already introduced in the system, only consider these operations, otherwise consider all found operations (the items in progress have priority)
P1: choose the item(s) with the largest number of successive operations
P2: choose the item(s) with the smallest number of operations in S
P3: choose the item(s) with the largest number of immediate successive operations
P4: choose the item(s) with the largest number of unfinished operations
P5: choose the item(s) with the shortest processing time
P6: if several items remain in S, one is chosen randomly
Step 2: schedule the above chosen operation and update the resource status
Step 3: update the list of possible operations S, if the list is empty go to Step 4, else go to Step 1
Step 4: update the transport simulation and increase the time index by one step
Step 5: if a robot finished to this time index, mark operation as completed and store operation in schedule
Step 6: update robot status and list of possible operations S, if S is empty go to Step 4, else go to Step 1

4.2. Step scheduler algorithm

The simulation program routines play an essential role in this type of scheduling process, used at rush order or resource failure. At each time moment it is imperative to know the exact location of each pallet carrying the product so that no other pallet assumes the same location, or that no pallet passes by another pallet, which is physically impossible on the real system. For this reason, the simulator is being used (but without any graphical output and running on its maximum speed dictated by the processor execution frequency) to ensure a realistic behaviour of the transport action. The simulation variables also contain information about the current status of the entire system, such as how many pallets are in the system at the current time index. Moreover, the time index is incremented in steps of 0.5 s, thus at any point of the scheduling the current time index may be read and stored in the schedule.

These time indexes correspond to the conveyor divert points when production is executed on the real system. Time indexes of interest stored while the schedule is generated are the time of insertion of an item into the system, the time to which a certain operation is started, and the time to which a certain operation is completed; consequently, the algorithm acts as a Resolved Scheduling Rate Planner (RSRP), because the global solution is *resolved* in evolutions progressing at time slices depending on effective product processing and transportation times. By integrating the simulation variables, all necessary information is present to run and validate the scheduling algorithm.

An iteration of the algorithm checks and completes the following steps:

Step 1: check how many pallets are in the system, if there are less than two pallets go to Step 2, else go to Step 3
Step 2: choose any item from the queue, based on termination priority
Step 2.1: for the chosen item generate a list of all possible operations based on predecessor constraints
Step 2.2: for each possible operation find all robots able of executing the task and calculate the wait time for each robot before the task could be executed once the item arrived at the station
Step 2.3: choose the operation with smallest waiting time and introduce the item on a new pallet with the destination acquired before, store the current time index as insertion time

Step 3: execute a step of one time index increment with the conveyor simulation and the robot operation execution. If a robot terminates an operation go to Step 4. If a pallet arrives at a robot station go to Step 6, else go to Step 5
Step 4: for the item that just finished an operation, store the current time index as operation completion time, mark the robot as *free*, then do the following:
Step 4.1: determine whether this item has been completed (all operations have been carried out); if so, mark the item as completed and send the pallet to the output, then continue with Step 5
Step 4.2: for the chosen item generate a list of all possible operations based on predecessor constraints
Step 4.3: for each possible operation find all robots able to execute the task and calculate the waiting time for each robot before the task could be executed once the item arrived at the station
Step 4.4: choose the operation with the smallest operation-start-time and send the pallet to that robot station
Step 5: if there are still items in the queue, or pallets in the system, go back to Step 1, else exit the algorithm
Step 6: for the arriving item store the current time index as operation start time, assign this item to the robot and mark the robot as *busy*, continue with Step 1.

Once an item has been introduced, it will remain in the system until it is completed. No item will leave the system and re-enter it to a later point in time. In other words any sequence (respecting the insertion rule of minimal waiting time) of alternating product types may be introduced into the system.

4.3. Change/failure and recovery management with embedded CNP negotiation

Alternative process plans, triggered by resource failure/recovery, local storage depletion or occurrence of rush orders, are automatically pipelined: (a) at the horizon of p_E products in course of execution in the system, based on heterarchical contract negotiation schemes (e.g. CNP) between valid resources, and (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 manufacturing resource (e.g. robot, machine tool); (ii) failure of one inspection operation (e.g. visual measurement of a component/assembly); (iii) depletion of one workstation storage (e.g. assembly parts are missing in one local robot storage).
2. Change occurring in production orders, i.e. the system receives a *rush order* as a new batch request (a 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 when a resource recovers from breakdown or missing parts are fed to the empty storage.

The states describing the processing capabilities of a resource and the actions taken while transiting from one state to another are presented in Fig. 7.

Upon monitoring the processing resources (robots), their status may be at run time: *available*—the resource can process products; *failed*—the resource does not 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 (robot, CNC) for estimation of their status during production execution:

- *Background interrogation*: periodic polling of RQST_STATUS and ACK_STATUS digital I/O lines between the PLC–OH coordinator and the Resource Controllers (robot, CNC).
- *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 (see Fig. 8). This communication practically validates the execution of the current OH operation on the particular resource (robot).

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 signaled 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

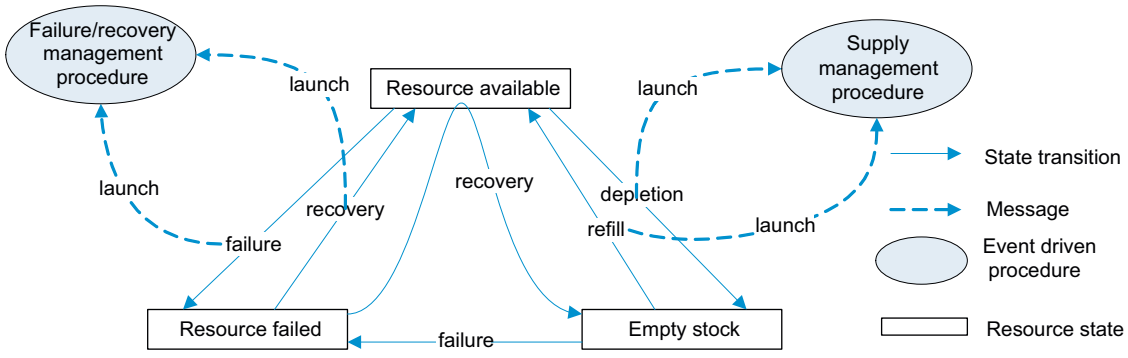

Fig. 7. Actions taken when a resource is changing from a state to another.

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).

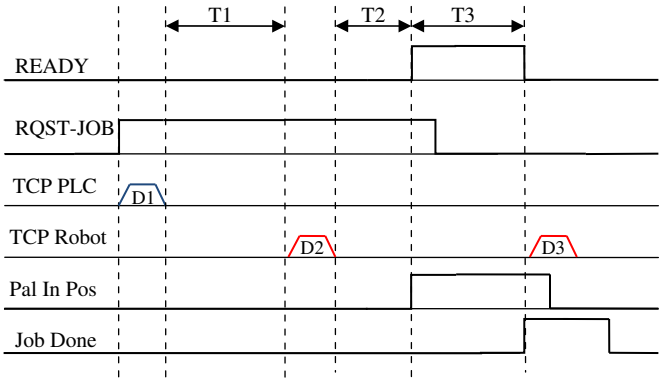

Fig. 8. Communication protocol between the PLC and a Robot Controller for authorizing an OH operation execution.

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. 9):

1. Stop immediately the transitions of executing OH, i.e. the circulation of *products_on_pallets* in the cell; production continues, however, at the remaining valid resources (robots, machine tools).
2. Update the Resource Holons with the new states of all robots.
3. Read Order Holons currently in execution (which are currently in the production 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*;

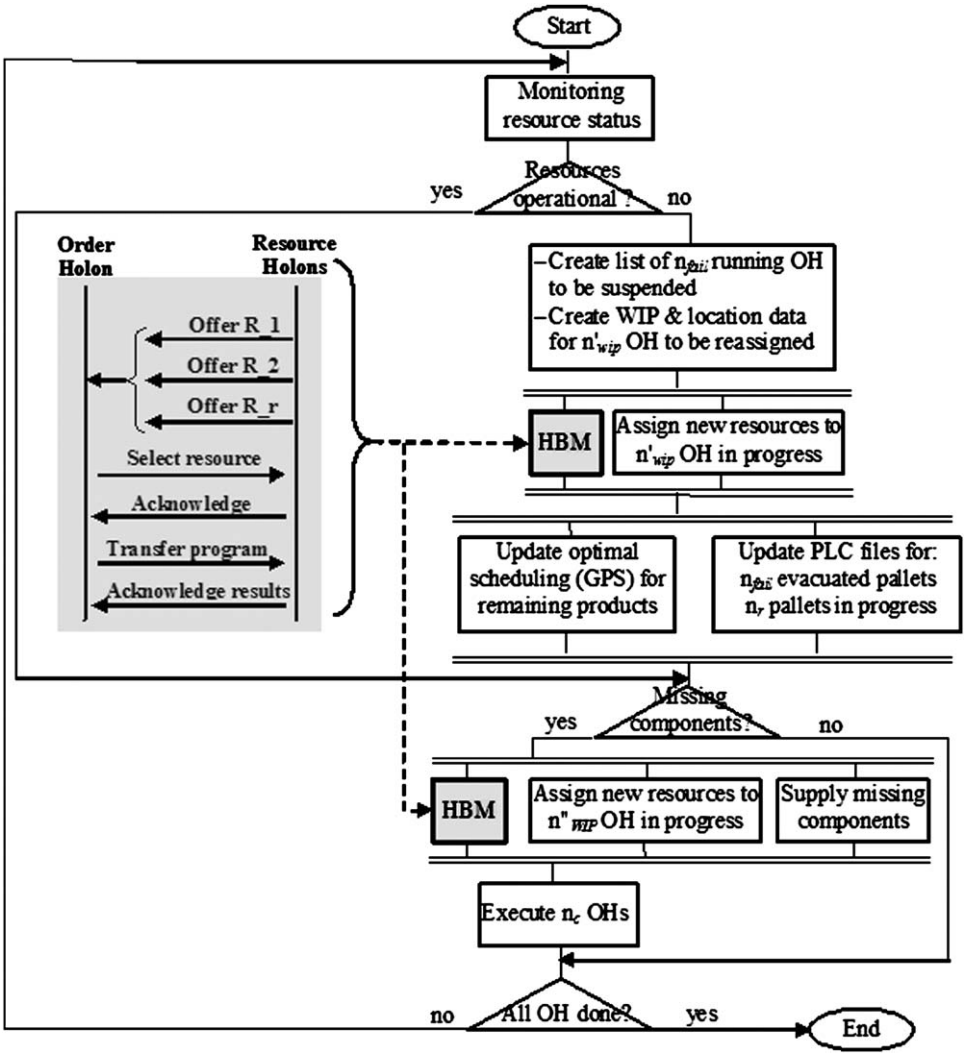

Fig. 9. Dynamic OH rescheduling at resource failure/storage depletion with embedded CNP job negotiation (monex).

- 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. Authorize the n'_{wip} OH to launch Contract Net Protocol-based negotiations (HBM) with the remaining available Resource Holons for rescheduling of their associated operations. n_{wip} are the OH currently introduced in the system (in the present implementation, $n_{wip} \leqslant 5$), 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 during their execution.
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).

In case of local storage depletion, the OH waiting to enter the robot station with exhausted storage will be either delayed if the resource is critical or rescheduled to another resource disposing of the missing component and able to perform the current operation. In such a situation, two actions take place:

1. One Supply Holon (SH) is created by the GSP, by specifying the type and number of parts to be retrieved, the supply source (a central cell storage tended by a SCARA robot under visual guidance), and the restoring destination (the exhausted local robot storage). The SH is immediately started.
2. From the n_{wip} OH currently in execution, n_d will be delayed until the empty storage, which is critical for certain of their mounting operations, is restored and $n_{wip}'' = n_{wip-} n_d$ OH will be rescheduled by the holonic bidding mechanism (HBM) to robots disposing of necessary assembly parts.
3. A lock is put on the system, and no further OH (new pallets) is introduced in the system until the last one of the n_d delayed OH is completed and exits the system. When both the SH and all n_d OH are terminated, the lock is suspended and the remaining OH are introduced in the system in packets of n_{wip}, their rescheduling being not necessary.

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 resource unavailability are now untagged and included in the APO list for scheduling as the may be manufactured again due to resource recovery (Lastra and Delamerm, 2006; Leitao et al., 2007).

The system is agile to changes occurring in production orders too, i.e. manages rush orders received as *new batch requests* from the ERP level while executing an already scheduled batch production (a sequence of Order Holons).

Because of the similarities 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), it was decided to use the Earliest Deadline First (EDF) procedure to schedule new batches (rush orders) for the robotized assembly cell.

Earliest Deadline First (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, Lipari, 2005). 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 will be 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 showed that under the following assumptions: (A1) all tasks are periodic, independent and fully preemptive; (A2) all tasks are released at the begining of the period and have deadlines equal to their period; (A3) all tasks have a fixed computation time or a fixed upper bound which is less or equal to their period; (A4) no task can voluntarily stop itself; (A5) all overheads are assumed to be 0; (A6) there is only one processor, a set of n periodic tasks can be scheduled if $\sum_{i=1}^{n} C_i/T_i \leqslant 1$, $n =$ number of tasks, $C_i =$ execution time, $T_i =$ cycle time, or, in other words, if the utilization of the processor (resource) is less than 100%.

A *batch* or Aggregate Product Order (APO) list 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 $\sum_{j=1}^{i} \text{makespan}_j \leqslant \text{delivery\_date}_i$, $i = \overline{1,n}$, then the batches can be assigned to the production cell using EDF without passing over the delivery dates, Tanaya et al., 1995".

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. 10):

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).
6. Analyze the possibility of allocating the batches to the manufacturing cell using the Earliest Deadline First procedure and second equation for feasibility test.
7. Allocate the batches on the real-production system according to the EDF procedure.
8. Resume execution process with new scheduled Order Holons.

In this mechanism for the management of changes in production orders, an i*nserted batch* is a batch that arrives while another

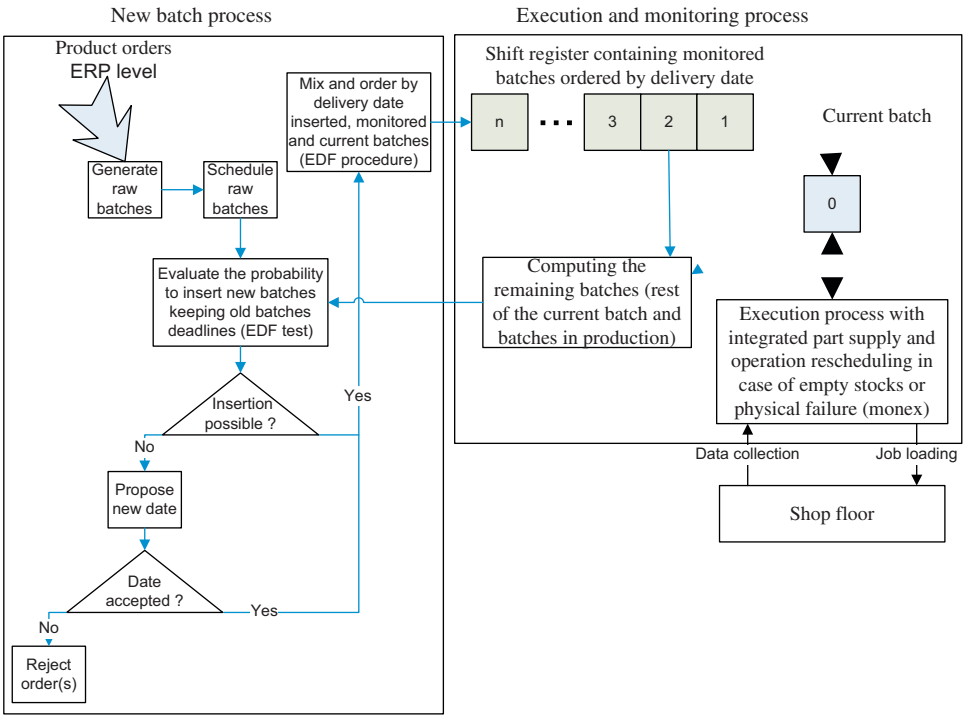

Fig. 10. Add *rush order* diagram and integration with dynamic job rescheduling based on CNP negotiation (monex).

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	The time interval needed for the current batch to be executed if started now and not interrupted (it is a result of scheduling)

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 *current batch* is in execution.

The capability of adding rush orders to production needs a new entity, the batch. In this way 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).

Because the process of batch execution is interruptible (preemptive system), new batches (rush orders) can be introduced exactly at the moment of their arrival. The insertion process is triggered by the arrival of a "new order" event; a real-time acceptance response can be provided (via the ERP level) to the customer if the rush order can be executed by the requersted delivery date.

5. Fault-tolerant integration of the distributed control system

The distributed control system uses an entity coordinating Order Holon execution, agentified as a Resource Holon—a Programmable Logic Controller (PLC), which controls the state transitions of OH by controlling the routes of *products_on_pallets* inserted in the manufacturing cell in packets of $n_{wip} \leqslant 5$. The PLC is the hardware counterpart of the cell conveyor control; to correctly control the devices of the transportation system (motors, stoppers, elevators, diverters) for product routing, an efficient way to identify and track *products_on_pallets* was conceived: each pallet has its own, unique code written on a magnetic device placed on the pallet carrier; this code is read in decision-taking locations on the conveyor (in pallet carrier diverting points) by reading devices connected to the PLC.

The PLC is thus able to locate in permanence the products within the production cell and, by checking these locations against the scheduled destinations (resource workplaces assigned for each operation), issues the appropriate commands to route products. The PLC maintains the background and ultimate interrogations about the status of resources (robots), and consequently validates the OH state transitions.

A cell server holds permanently the list of Order Holons, update by pipelined GPS (hierarchical) and CNP negotiation (heterarchical); this list is mapped in a standard file set containing 256 arrays (which correspond to max. 256 pallets in one batch) each one consisting of 16 structures (a *product_on_pallet* supports maximum 16 operations):

```
Pallets_array: ARRAY [0..255,1..16] OF PalletData;
```

PalletData is the base structure of information about an operation that will be done on a product at a robot station and has the following components:

```
TYPE PalletData:
STRUCT
   station: BYTE; (*number of station where the
   operation will take place*)
   operation: BYTE; (*operation type*)
   timemin: WORD; (*min. duration of the operation*)
   timemax: WORD;(* max. duration of the operation
   *)
```

```
    report: BYTE; (*a coded byte indicating how was
    executed the operation
       ok, failed, etc*) END_STRUCT
END_TYPE
```

These values can be modified anytime at failure/recovery or change-triggered rescheduling. The cell server also sends the PLC a time variables array, representing the suggested time (computed by the GPS) of inserting each pallet in the conveyor: `time_insertion_array[0..255]: word`

An important element in the PLC software project is a 256-item array *index_array*; each item is associated with the *product_on_pallet* (OH) having as binary code the index of this item in the array. The value of this item is the index of the next operation to be performed on the associated *product_on_pallet*:

```
Index_array: ARRAY [0..255] OF BYTE;
```

A complex PLC project was developed in Indralogic, consisting of visual interfaces, Sequential Function Chart and Structured Text programs for execution and tracking of the OH sequence. As several independent processes have to be controlled by a single PLC, the best solution is to *separately* control each execution device (actuators, valves, etc.) by means of a single logical control diagram. The synchronization between logical diagrams is made by global variables.

The holonic manufacturing control software is distributed on several processors, grouped as follows:

1. Management of changes in production orders:
   - Cell server (IBM xSeries 3500 in the project): mix and order by delivery date inserted, monitored and current batches, perform Global Production Scheduling *at global batch horizon* using Expertise Holons in KBS and the Dynamic Cell Simulator in Step Scheduler, create Order Holons and Supply Holons using information provided by Product Holons and Resource Holons;
   - Station Computers (IBM PC workstations in the project) SC_i, $1 \leqslant i \leqslant r$: participate in the contract-based negotiation for the jobs offered by (at least) the n_{wip} products in execution, in the process of rescheduling *at packet horizon* when resource failure or storage exhaustion occurs.
2. Execution and monitoring of Order Holons:
   - Cell PLC (Bosch in the project): controls the cell conveyor for product routing and interrogates processing resources (robots, machine tools, storages) for their availability;
   - Station Controllers (Adept in the project) RC_i, $1 \leqslant i \leqslant r$: run programs for product operations.

The holonic control can be configured from totally hierarchical ($\mathrm{INT}(P/n_{wip})+1$ packets of products are optimally scheduled by the GSP and 1 packet is negotiated by valid resources in case of failure) to totally heterarchical (all $\mathrm{INT}(P/n_{wip})+1$ packets of $n_{wip}=5$ products are negotiated).

The control and communication structure is fault tolerant (Fig. 11).

The Cell Server and PLC, which are SPOF, will be hardware backup-ed. The Robot Controllers have multiple communication facilities; their integration in the manufacturing control system allows fault-tolerant communication, because:

- Station Computers are connected to Station (Robot) Controllers via a SC server—RC client *Switched Ethernet Network*.

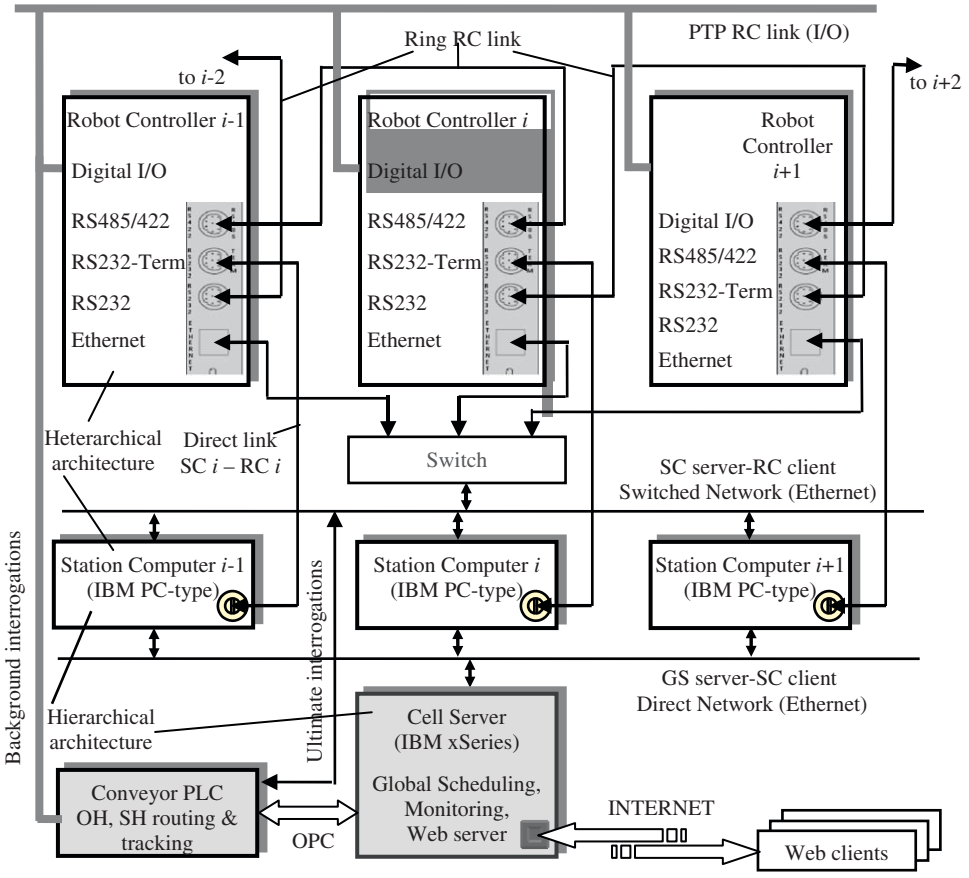

Fig. 11. Fault-tolerant distributed control and communication architecture.

- The failure of a SC is detected by monitoring via the *direct serial links* SC_i–RC_i and determines in consequence the rescheduling of jobs for the $(r-1)$ remaining valid RC_j, $1 \leqslant j \leqslant r$, $j \neq i$.
- If one of the Station Computers is down, its role is taken over by one of the remaining $(r-1)$ workstations, as each SC database is replicated and updated on-line in all the other $(r-1)$ ones.
- If the switch is down, the communication between the SC (or PLC) and the RC clusters still operates via the *direct links* SC_i–RC_i and the Ring RC *link*.
- High-speed, real-time operation synchronizing between process and transport resources is done by a *cross-connection I/O* network.

6. Experimental results and conclusions

The distributed control solution was implemented, tested and validated on a real manufacturing structure with industrial assembly robots and 4-axis CNC milling machines, using the holonic approach. This development platform was recently put in place in the Centre of Research in Robotics and CIM within the University Politehnica of Bucharest (Fig. 12).

The described holonic implementation framework allows networking equipment from different producers. The cost of the development platform is directly reflected in its high-precision performances, integrated inspection services, relaxation of material presenting constraints, fixture simplification and management of changes.

The control is structure is fully operational, both in the normal hierarchical mode and upon switching automatically to the heterarchical one in response to rush order requests, part supply and resource failures.

Production scheduling at batch level was implemented and tested using the EDF method; Fig. 13 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 $T_i j = (m, dd)$ 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).

An example of production definition at batch level for four products (H-, U-, L- and C-type products) resulting from assembling axes and T-, I-, L- and r-type components is given in Fig. 14.

As can be seen, there are several types of operations to be carried out in order to manufacture the four types of products: machining (milling), mounting support axes, assembling parts and visual inspecting.

For the experiments reported, the number of products simultaneously in execution was limited to 5. Table 2 below gives the production times resulting from the Step Scheduler RSRP computation in two scenarios: (i) only H-type products; (ii) equal number of H-, U-, L- and C-type of products within one of the four batch sizes (batch sizes were 4, 20, 40 and 60 products):

The system's behaviour was tested with good results at storage depletion (less than 68 s to generate a SH and restore the furthest local robot storage) and resource failure (SC, switch and RC). Future research work will be directed towards integrating the process control- and ERP areas through an enhanced information management system based on the RFID technology.

The general features of the proposed holonic implementing framework facilitate, beyond the product assembling with machined components, the development of any other discrete, repetitive manufacturing applications. Feature like: decomposition of the production system into entities relative to the basic areas specific to an enterprise (production, process and business), description of the types of manufacturing entities and of the communication protocols that take place between them, and the decision scenarios during resource failure/recovery and stock restoring are reusable.

From the algorithmic point of view, the proposed resolved scheduling rate planner based on variable-timing simulation, facing the NP complexity aspect of the batch scheduling problem can be reused for any topology of the material transportation system, due to its graph-type, object-oriented description.

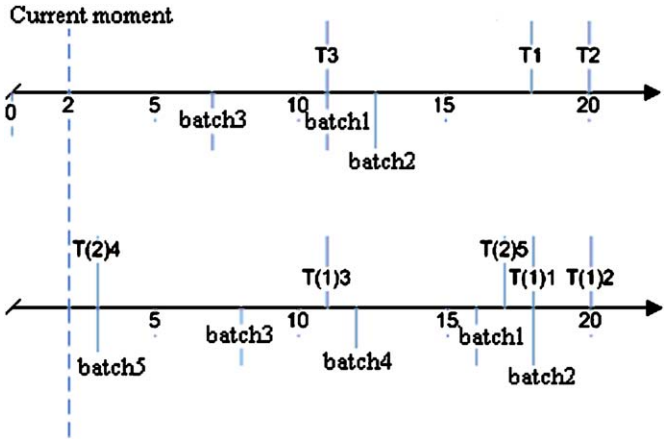

Fig. 13. Inserting new batches among executing ones with the EDF algorithm.

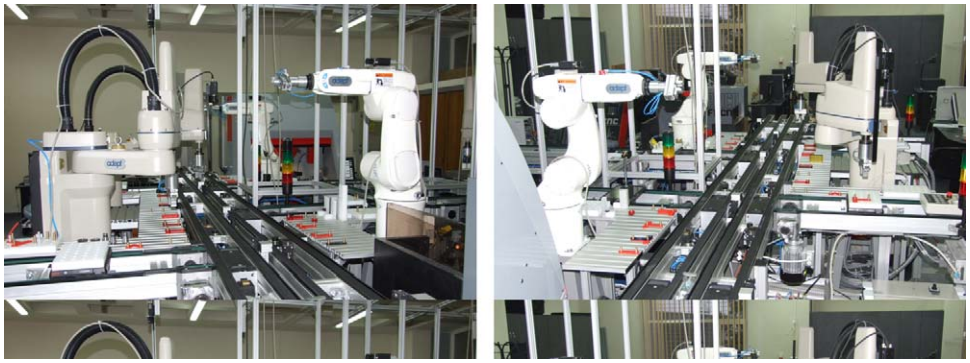

Fig. 12. Layout of the physical manufacturing cell subject to the holonic control development.

Assembly sequence H-product:	Assembly sequence U-product:	Assembly sequence L-product:	Assembly sequence C-product:
1. Mount axis	1. Mount axis	1. Mount axis	1. Mount axis
2. Mount axis	2. Mount axis	2. Mount axis	2. Mount axis
3. Mount axis	3. Mount axis	3. Mount axis	3. Mount axis
4. Mount T	4. Mount L	4. Mount l	4. Mount r
5. Vision inspection	5. Vision inspection	5. Vision inspection	5. Vision inspection
6. Mount l	6. Mount l	6. Mount l	6. Mount r
7. Mill l	7. Mill l	7. Mill l	7. Vision inspection
8. Vision inspection	8. Vision inspection	8. Vision inspection	

Fig. 14. Example of production definition at batch level (assembly, machining and visual inspection tasks included).

Table 2
Production time for H-, U-, L- and C-type batches and resuming times at resource failure.

Batch size	Production time [time units]		Worst recovery time [time units]	
	H-type [RSRP/CNP]	Equal number of H-, U-, L-, and C-type [RSRP/CNP negotiation]	Alternate OH at [packet = 5] level (resource *i* failure: R*i*F)	New SH for restoring Local Storage *i* (LS*i*) at depletion
4	684/734	663/687	6.4 (R1F)	97 (LS1)
20	2841/3112	2550/2712	6.5 (R2F)	112 (LS2)
40	5485/5962	4934/5288	6.8 (R3F)	136 (LS3)
60	8129/9089	7362/7902	6.5 (R4F)	83 (LS4)

References

Babiceanu, R.F., Chen, F.F., Sturges, R.H., 2004. Framework for control of automated material-handling systems using holonic manufacturing approach. International Journal of Production Research 42 (17), 3551–3564.

Barata, J., 2005. Coalition Based Approach for Shop Floor Agility, Orion Ed. Amadora, Lisbon.

Bongaerts, L., Wyns, J., Detand, J., Van Brussel, H., Valckenaers, P., 1996. Identification of manufacturing holons. In: Albayrak, S., Bussmann, S. (Eds.), Proceedings of the European Workshop for Agent-Oriented Systems in Manufacturing, Berlin, pp. 57–73.

Bongaerts, P., Monostori, L., McFarlane, D., Kadar, B., 1998. Hierarchy in distributed shop floor control. In: Proceedings of the First International Workshop on Intelligent Manufacturing Systems IMS-EUROPE, Ed. EPFL, Lausanne, pp. 97–113.

Borangiu, Th., 2004. Intelligent Image Processing in Robotics and Manufacturing. Romanian Academy Publishing House, Bucharest.

Borangiu, Th., Balibrea, L.M.T., Anton, F.D., Ivanescu, N.A., Tunaru, S., Dogar, A., Raileanu, S., 2007. Holonic fault-tolerant control of networked robot-vision assembly stations. Preprints of the IFAC Workshop Intelligent Assembly and Disassembly-IAD, Alicante.

Cheng, F.-T., Chang, C.-F., Wu, S.-L., 2006. Development of Holonic Manufacturing Execution Systems, Industrial Robotics: Theory, Modelling and Control, Advanced Robotics Systems. Pro Literatur Verlag Robert Mayer-Scholz Germany, Vienna.

Deen, S.M., 2003. Agent-Based Manufacturing—Advances in Holonic Approach. Springer, Berlin.

Duffie, N.A., Prabhu, V.V., 1994. Real-time distributed scheduling of heterarchical manufacturing systems. Journal of Manufacturing Systems 13 (2), 94–107.

Higuera, A.G., Montalvo, A.C., 2007. RFID-enhanced multi-agent based control for a machining system. International Journal on Flexible Manufacturing Systems 19, 41–61.

Iwamura, K., Taimizu, Y., Sugimura, N., 2005. A study on real-time scheduling for holonic manufacturing systems—simulation for estimation of future status by individual holons. Knowledge and Skill Chains in Engineering and Manufacturing Information Infrastructure in the Era of Global Communications, Lecture notes in Computer Science, vol. 4659/2007, Springer Berlin/Heidelberg, pp. 301–312.

Kusiak, A., 1990. Intelligent Manufacturing Systems. Prentice-Hall, Englewood Cliffs, New York.

Lastra, J., Delamerm, I., 2006. Semantic web services in factory automation: fundamental insights and research roadmap. IEEE Transactions on Industrial Informatics 2 (1), 1–11.

Leitao, P., Restivo, F., 2006. ADACOR: a holonic architecture for agile and adaptive manufacturing control. Computers in Industry 57 (2), 121–130.

Leitao, P., Mendes, J., Colombo, A.W., Restivo, F., 2007. Reconfigurable production control systems: beyond ADACOR. Preprints of IFAC Workshop Intell. Assembly and Disassembly – IAD07, pp. 89–94.

Lipari, G., 2005. Sistemi in Tempo Reale (EDF). Course Scuola Superiore, Sant'Anna, Pisa.

Liu, C.L., Layland, J.W., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. Journal of ACM 20 (1), 46–61.

Lusch, R.F., Vargo, S.L., Wessels, G., 2008. Towards a conceptual foundation for service science: contributions from service-dominant logic. IBM Systems Journal 47, 1.

Markus, A., Vancza, T., Monostori, L., 1996. A market approach to holonic manufacturing. Annals of the CIRP 45 (1), 433–436.

Morel, G., Panetto, H., Zaremba, M., Mayer, F., 2003. Manufacturing enterprise control and management system engineering: rationales and open issues. IFAC Annual Reviews in Control.

Onori, M., Barata, J., Frey, R., 2006. Evolvable assembly systems basic principles. In: Shen, W. (Ed.), IT for Balanced Manufacturing Systems 220. IFIP, Springer, Boston, pp. 317–328.

Pétin, J.-F., Gouyon, D., Morel, G., 2007. Supervisory synthesis for product-driven automation and its application to a flexible assembly cell. Control Engineering Practice 15, 595–614.

Rahimifard, S., 2004. Semi-heterarchical production planning structures in the support of team-based manufacturing. International Journal of Production Research 42 (17), 3369–3382.

Ramos, C., 1996. A holonic approach for task scheduling in manufacturing systems, In: Proceedings of the IEEE International Conference on Robotics and Automation, Minneapolis, USA, pp. 2511–2516.

Sha, L., et al., 2004. Real time scheduling theory: a historical perspective. Real-Time Systems 28 (2–3), 101–155.

Tanaya, P.I., Detand, J., Kruth, J.P., 1995. Holonic Machine Controller: a study and implementation of holonic behaviour to current NC controller. Cooperation in Manufacturing: CIM at Work, pp. 375–396.

Usher, M.J., Wang, Y.-C., 2000. Negotiation between intelligent agents for manufacturing control. In: Proceedings of the EDA 2000 Conference, Orlando, FL.

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

Van Brussel, H.V., Wyns, J., Valckenaers, P., Bongaerts, L., Peeters, P., 1998. Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry, Special Issue on Intelligent Manufacturing Systems 37 (3), 255–276.

Wyns, J., Van Ginderachter, T., Valckenaers, P., Van Brussel, H., 1997. Integration of resource allocation and process control in holonic manufacturing systems. In: Proceedings of the 29th CIRP International Seminar on Manufacturing Systems, pp. 57–62.

〈http://hms.ifw.uni-hannover.de〉 HMS Consortium web site; www.rvholon.cimr.pub.ro, RVHLON site.