Chapter Number

Holonic Robot Control for Job Shop Assembly by Dynamic Simulation

Theodor Borangiu, Silviu Raileanu, Andrei Rosu and Mihai Parlea
University Politehnica of Bucharest
Romania

1. Introduction

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 to be constantly flexible, adapt to newly developed processes and technologies and to rapidly changing environmental protection regulations, support innovation and continuous development processes (Nylund et al, 2008). Although the optimization of the production process remains a key aspect in the domain of fabrication systems, adaptive production gains more and more field (Sauer, 2008). Flexible manufacturing systems should be able to quickly adapt to new situations like machine breakdown, machine recovery due to physical failure or stock depletion and also face rush orders (Borangiu et al, 2008).

In recent decades, scientific developments in the field of production control have led to new architectures including heterarchical/non-hierarchical architectures that play a prominent role in flexible manufacturing.

The **traditional** approach is mainly associated to the initial CIM concept (Computer Integrated Manufacturing) and usually leads to centralized or hierarchical control structures in which a supervisor initiates all the activities and the subordinate units respond directly in order to perform them. Due to the complexity of manufacturing problems, the usual practice has been to split the global problem into hierarchically dependent functions that operate within smaller time ranges, such as planning, scheduling, control and monitoring. This traditional approach is known to provide near optimal solutions, but only when hard assumptions are met, for example, no external (e.g., rush orders) or internal (e.g., machine breakdowns) perturbations, a priori known demands, and/or supplier reliability. Since reality is rarely so deterministic, this approach rapidly becomes inefficient when the system must deal with stochastic behaviour.

The above observations allowed researchers to design in a second approach new control architectures formed by a group of independent entities that bid for orders based on their status and future workload. There is no master-slave relationship; all the entities including the manager of a particular order are bidding for it. Due to the decentralized architecture, the entities have full local autonomy and can react promptly to any change imposed to the system. However, because the behaviour of a production 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. These control architectures, also called emergent or **self-organized**, can be categorized in four types (Bousiba et al, 2002): *bionic* & *bio-inspired*, as proposed by Okino (Okino, 1993) and Dorigo & Stuzle (Dorigo et al, 2004); *multi-agent*, as proposed by Maione & Naso (Maione et al, 2003); *holonic*, as proposed by Van Brussel (Van Brussel et al, 1998); and *heterarchical*, as proposed by Trentesaux (Trentesaux et al., 1998). An analysis of the state-of-the-art has been recently published by Trentesaux (Trentesaux, 2007). His main conclusion is that the expected advantages of such architectures are related to agility: on short term such architectures are reactive and on long term they are able to adapt themselves to their environment. However, these last control architectures suffer from the lack of long-term optimality, even when the environment remains deterministic, which can be considered as a "myopic" behaviour. This is the main reason why such control architectures are not really used by industrialists at the moment.

In order to benefit from the advantages of both types of architectures, traditional and emergent, a new control paradigm was proposed by (Sallez et al., 2009) in which traditional explicit control is combined with an innovative type of control called **implicit** control. This paradigm is called *open-control*, meaning that subordinate entities are characterised by autonomy and an open communication mechanism permits them to be influenced by higher level entities directly or indirectly.

In the heterarhical control approach there is also a new research direction nowadays focused on the concept "system controlled by the product" in which dynamical information and decisional capabilities are embedded into the product, making it an active entity in the fabrication process (McFarlane et al., 2002, Zbib et al., 2008).

Rather then combining the hierarchical and heterarchical control, an approach is proposed in the current work in which the two control architectures are alternated based on the current state of the system called distributed semi-heterarchic control (Borangiu et. al, 2008). Thus, the system starts working in a hierarchical manner, using an offline schedule, in order to optimize production, but as soon as a disturbance appears it switches to a heterarchical operation mode in which resources bid for the execution of orders.

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), 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 functionalities to other holons (Ramos, 1996; Deen, 2003).

Based on Koestler's concept, the next definitions, established by the Holonic Manufacturing Systems (HMS) consortium (Van Brussel et al., 1998) were accepted and used in this 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.
- *Co-operation*: A process whereby a set of entities develops and executes mutually acceptable plans.

- *Holarchy*: A system of holons that can co-operate to achieve a goal or objective. The holarchy defines the basic rules for co-operation of holons and thereby limits their autonomy (Wyns, 1997).
- *Holonic manufacturing execution system (HMES)*: a holarchy integrating in (custom designed) software architecture the entire range of manufacturing tasks from ordering to design, and production execution.
- *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 have developed holonic control frameworks to operate parts of a manufacturing system (e.g. part processing on multiple machine tools, part assembling on multiple robots, etc), but only a few considered material-handling tasks (Liu et al., 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; (c) verify the assembly in different execution stages (Borangiu, 2009).

The proposed holonic control framework faces the difficulties arising when moving from control theory to practice, because: (i) the real cell conveyor is modelled, parameterized and integrated in the generic job scheduling; (ii) the material components (parts, assemblies) are described by task-dependent features which are extracted from images processed in real time for material qualifying and product inspection; (iii) the mapping of job scheduling to job execution via conveyor devices (motors, stoppers, diverters) is granted to PLC networks. In order to face resource breakdowns, the job shop production 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, overall 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);
- real time vision-based robot guidance (GVR) during precision assembly and visual in line geometry control of products (AVI) are requirements imposed to increase the diversity and quality of services performed;
- efficient (optimal) use of available resources (robot, CNC machine tools) in normal operating mode;
- stability in face of disturbances (resource failure, storage depletion).

The paper describes: (i) the design and implementing of a PLC-based distributed control architecture for a production system with networked assembly robots and machine tools, automatically switching between 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 and execution of part qualifying operations by real-time, high-speed image processing and feature extraction (vi) the interconnection of job-shop control processes with business processes at enterprise level, by managing offer requests, customer orders and providing feedback on the current status of batch orders.
The proposed design and implementing framework addresses a *networked robotized job shop assembly structure* composed by a number or robot-vision stations, linked by a closed-loop transportation system (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).

2. Generic holonic control model for a FMS

2.1 Description of the FMS processes

According to (Brussel et al, 1998) a fabrication system is composed of the following generic entities and domains that are associated to the production:

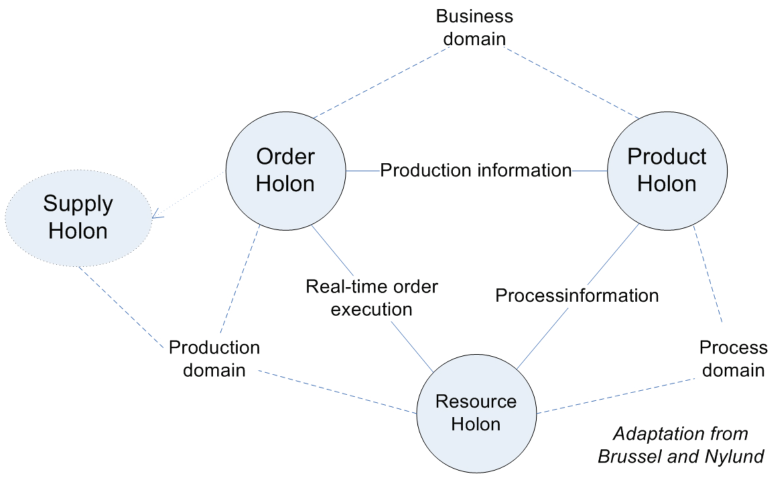

Fig. 1. HMS structure (Brussel et al, 1998, Nylund, 2008), with supply/domain extension

Entities and domains have different purposes in the system, and are described in the PROSA reference architecture which explains the structure of a fabrication system using three basic holons: Product- (PH), Resource- (RH) and Order-Holon (OH) (Brussel et al, 1998). These entities are interconnected two by two with the process-, production- and business domains (Nylund et al., 2008). The process- along with the production domains characterizes the system from the internal point of view. The first, *process domain*, is related with the system's capabilities to be able to execute certain operations that are needed to manufacture products offered to the clients. These capabilities are defined by the products and are realized by the resources. The second one, the *production domain*, manages the real-time information which relates the orders to the resources. This information is subject to offline and online scheduling in order to optimize the functioning of the fabrication system. The last domain, the *business domain*, relates orders to products and the fabrication system to the external world represented by clients.

Generic structure of a holon

Fig. 2 shows the structure of a generic holon, containing the digital, real, virtual and communication parts.

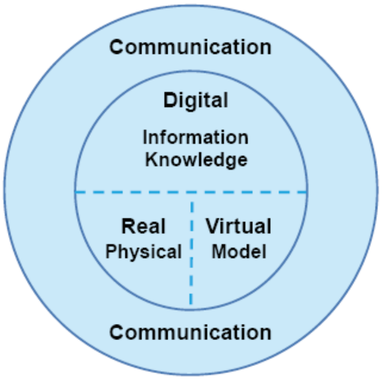

Fig. 2. The structure of a generic holon (Nylund et al., 2008)

In order for all of the different holons of the system to cooperate they must have similar structures, especially similar information structure. According to (Nylund et al., 2008) a general entity is composed of a *real part* which represents the physical resource capable of performing operations on products, a *virtual part* which is the model of the entity, a *digital part* in charge with the decision making and a communication port responsible for cooperation with the surrounding environment.

2.2 Component entities

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; Valkaenars 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 breakdowns) 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 parts 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* (RH), the *Product Holon* (PH), and the *Order Holon* (OH). Each of these holons may exist more than once to fully define the manufacturing cell. Order Holons are created by a Global Production Scheduler from the 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 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. 3 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,

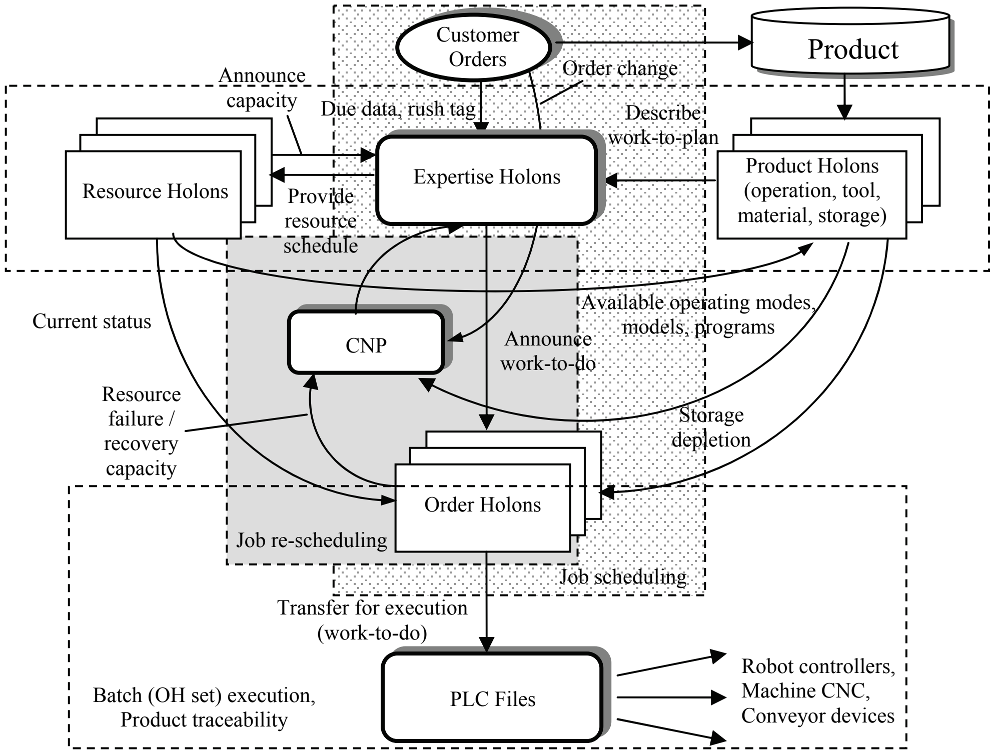
Customer Orders
Product
Order change
Announce capacity
Due data, rush tag
Describe work-to-plan
Resource Holons
Expertise Holons
Product Holons (operation, tool, material, storage)
Provide resource schedule
Current status
Available operating modes, models, programs
CNP
Announce work-to-do
Resource failure / recovery capacity
Storage depletion
Order Holons
Job re-scheduling
Job scheduling
Transfer for execution (work-to-do)
Batch (OH set) execution, Product traceability
PLC Files
Robot controllers, Machine CNC, Conveyor devices

Fig. 3. Basic holon cooperation and communication structure in the semi-heterarchical control architecture

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.

The **resource holon** (RH) holds information about manufacturing resources (robots manipulators, grippers, machine tools, video cameras, magnetic sensors, RFID devices, a.o.). In general any resource may have a number of sub-resources (e.g. a robot manipulator with gripper with two fingers with force sensors), which were seen as holons. This project considers an entire resource with all its sub-resources as a holon without making the distinction of sub-systems. The hardware part of this type of holon is the actual physical robot and gripper with its functions. A permanent data exchange between hardware and software ensures that the actual status is accessible through the software representation of the resource holon.

A **product holon** (PH) holds information about a product type. Any (assembly) type that may be produced within the manufacturing system and resource setup must be defined in a product holon. The fact that such a holon exists does not necessarily mean that the respective product is being really assembled. Only the array of order holons (described in the next paragraph) will specify that something is manufactured and in what quantity. The product information is more of a theoretical description of the physical counter piece but not directly associated with one individual physical item, unlike the resource holon. However, the availability of assembling components is ultimately checked by the PLC prior to authorize the final transfer of a pallet carrying the product to be assembled in a robot-vision workstation (see Fig. 6).

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 the information about the status of this very item at any time reaching from *assembly not started yet* throughout *order progressing to order completed*. Furthermore a complete manufacturing schedule must be computed holding all necessary information relevant for the production cell to successfully complete these orders, eventually satisfying a cost function such as the throughput or resource loading.

Before production starts for a specific aggregate order, customer commands exist in form of electronic information. If a certain product needs to be manufactured *n* times, *n* identical *raw order holons* are first created (Fig. 4).

During production execution orders can be seen as they progressively develop on a carrying support (pallet) in the system; after one order has been completed, the item gets cleared from the exiting pallet and has now a physical form. Before a schedule is defined for an aggregate order, raw order holons are created based on the information stored in the product holon.

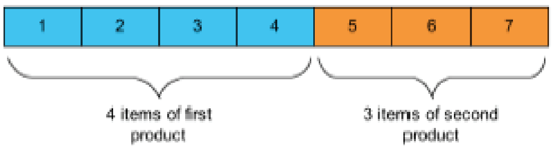

Fig. 4. Queue of two products (raw order holons) with a total of 7 items

A layer of Order Holons ($\mathrm{OH}_p, 1 \le p \le P$) of variable depth, corresponding to *assembly plans* computed off line for the P final products is the output of the production scheduler fed with *raw customer orders*. A basic (quasi optimal) process plan is generated as a sequence of Order Holons (assembled products). Production planning uses the Step Scheduler developed both for production start up and resource failure and recovery situations. To formalize the OH scheduling process, the notations and definitions below are introduced:

O = Set of all operations (assembly, conditioning)

P = Set of all assemblies (final products)

OA_p = Set of operations for assembly $A_p, p \in P$

L = Set of all resource types

Q_l = Set of resources of type $l, l \in L$

t = Current scheduling time

r_{lq} = Resource q of type l, $q \in Q_l, l \in L$

For the networked assembly problem, the following types of resources were defined:

- r_{1q} = assembly robot manipulator, $q = 1,2$: SCARA, $q = 3,4$: vertical articulated;
- r_{2q} = gripper, $q = 1,2$: 2(3) –finger number, $q = 3,4$: flat / concave-contact profile;
- r_{3q} = end-effector tool, $q = 1,2,3$: none / bolt / screwdriver;
- r_{4q} = physical-virtual camera duality ($\mathrm{P}_i\mathrm{V}_j$), $q = 1,2,\ldots,\sum nv_i$, $1 \le j \le nv_i$, where nv_i = no. of virtual cameras defined and installed for each physical camera $i, 1 \le i \le 9$;
- r_{5q} = magnetic code R/W device, $q = 1,2,3,4$.

Resource r_{lq} is: *operational* if it can be used after a finite time delay $\Delta_{lq}, q \in Q_l, l \in L, \Delta_{lq} \ge 0$, *available* if $\Delta_{lq} = 0$, and *down* otherwise. 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:

$$\mathrm{AP}_p^{(\delta)} = [\ldots,(o_i, t_i^{(\delta)}, R_i^{(\delta)}),\ldots],\ 1 \le i \le f,$$

where $R_i^{(\delta)} = \{r_{1q|i}^{(\delta)},\dots,r_{5q|i}^{(\delta)}\}$, $q \in Q_l, l \in L, 1 \le i \le f$.

The Step Scheduler for assembly computes off-line the $\mathrm{OH}_p^{(\delta)}, 1 \le p \le P$ at batch level rending assembly plans $\mathrm{AP}_p^{(\delta)}$ *available* for products $A_p, p \in P$. One operation $o_i \in O$ in the p^{th} OH is *executable* if all resources needed to carry it out are defined as *operational* by at least one $\mathrm{AP}_p^{(\delta)}$. Operation $o_i \in O$ is *schedulable* at time t , if

1. No other operation (mounting, inspecting) upon the same product is being processed at time t .
2. All operations preceding o_i have been completed before time t .
3. All resources needed by the basic assembly plans $\mathrm{AP}_p^{(\delta)}$ to process operation o_i are available.

Since a single holon may be seen as a class object in the object oriented programming environment (in this project the 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 *Holons.HolonManager* class.
The array of *Holons.Product* class instances assumes a size of currently present product types; each element represents one product type with all necessary information to generate orders of this product type. The last array composed of *Holons.Order* class instances has a number of elements equal to the total count of items that needs to be manufactured. Each element defines an order of a certain product type with its specific assembly schedule.
According to the definition (Koestler, 1967) a holon is an autonomous and collaborative entity which contains a hardware and software part. In the case of the **supply holon** the hardware part is represented by the pallet carrying pieces in the system and the software is the application on the PLC which controls the path pallet and manages the exchange of messages between the workplaces and the supplied station. The relationship between the two sides is 1 to 1 and synchronization is done via the code written on the pallet (251-254). Depending on when the supplies are sent there are two types of holons: one supplying the workplaces at production start up, and the one re-supplying the workstations during production execution. The life of a Supply Holon spreads during all the manufacturing process. Unlike a normal order or a Supply Holon needed for initial feeding, where the operations to be performed are established in advance, for a Supply Holon used at re-feeding the operations are chosen dynamically depending on the usage of workstations. During production a single Supply Holon stays in the system pending for re-feeding operations. Another re-feeding constraint is the following: a pallet can carry only one type of parts for re-supply because when a workstation signals that it has an empty stock it means that it lacks a single type of piece. The type of piece is signaled to the PLC by a code similar to code that is sent when an order requires the execution of an operation. This signal is then sent to the workstation and is stored into a FIFO-type stack. From this stack re-feeding requests will be taken.
A particular case of re-feeding is the initial supply, when all workstation stocks are empty. In this case the number of Supply Holons will be extended to 4, which is the number of

pallets which ensure that the system will not block. After re-feeding only the pallet that supplies workstation during production remains in the system. The number of pallets is computed based on the configuration of the transportation system (Fig. 5).

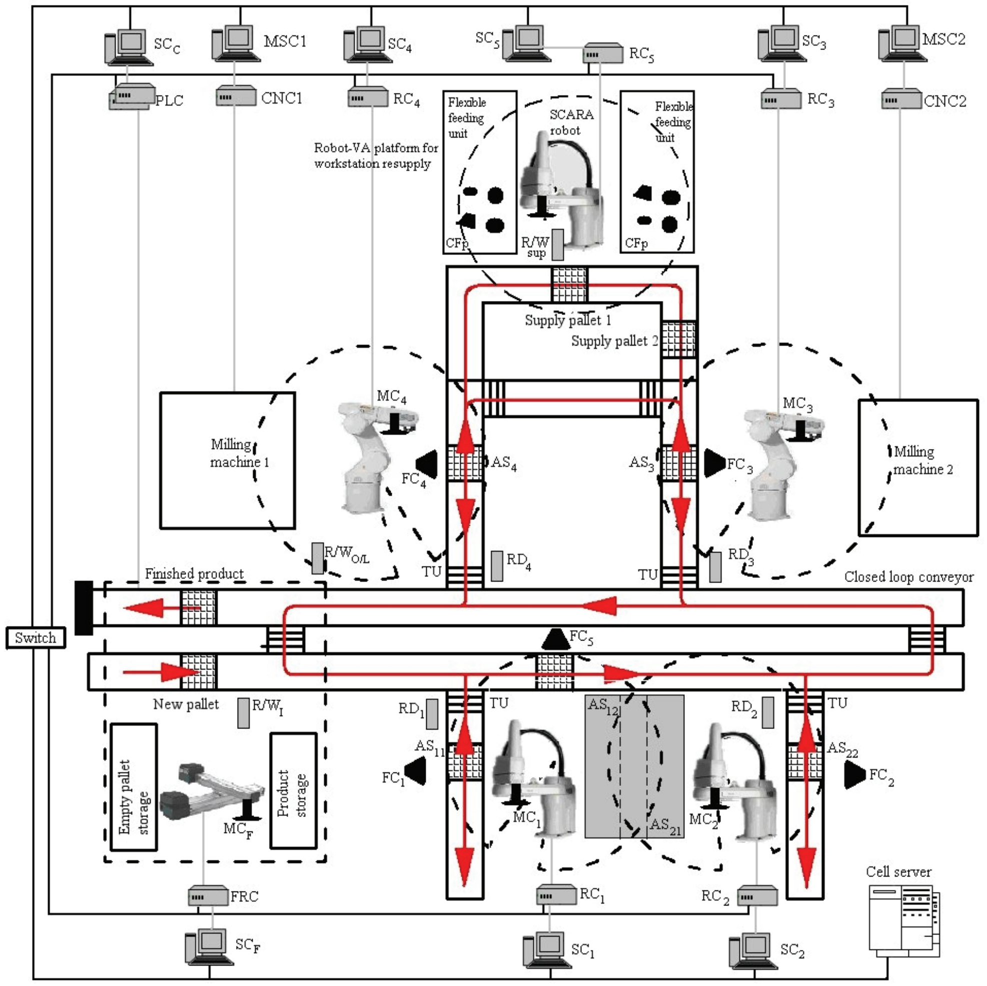

Fig. 5. Holonic manufacturing system with self-supply of assembly parts

Before production starts for a particular Aggregate List of Product Orders (APO created at ERP level), the OHs exist only in electronic 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. OHs 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 is distinct 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 entity describing a certain product type, 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.

The mappings between the (holonic) system requirements and the functional architecture are included in Figs. 1 and 4. Fig. 6 describes the mappings between the functional architecture and the physical one (for a 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.

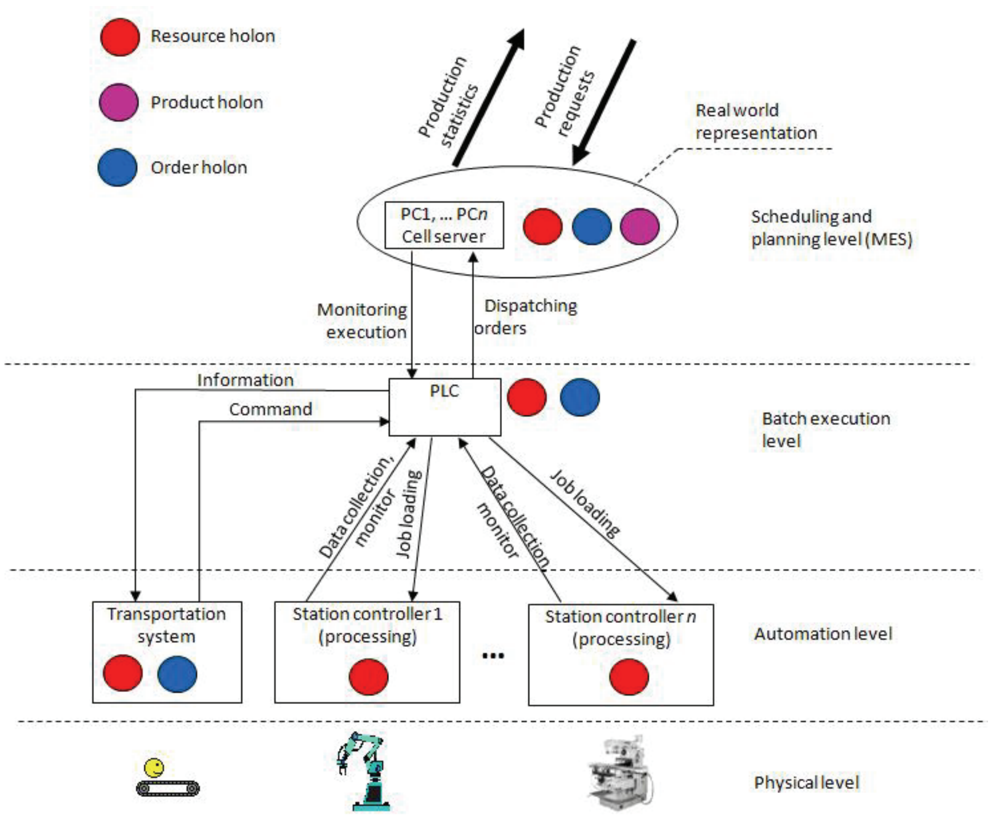

Fig. 6. Mapping between the (holonic) system and the physical architecture

3. Implementation methodology using holonic principles

The general information flow that characterizes the production system at enterprise level is described in Fig. 7:

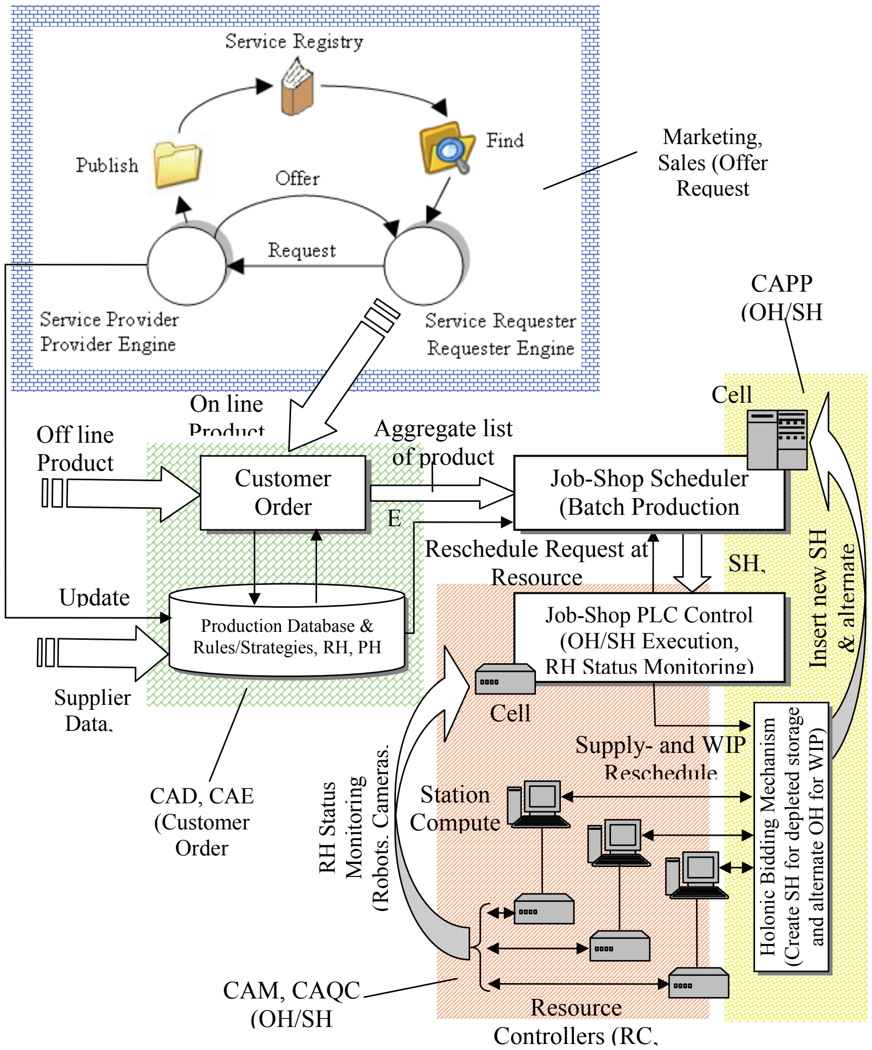
Service Registry
Publish
Find
Offer
Request
Service Provider Engine
Service Requester Engine
Marketing, Sales (Offer Request
CAPP (OH/SH
On line Product
Off line Product
Aggregate list of product
Customer Order
Job-Shop Scheduler (Batch Production
E
Reschedule Request at Resource
SH,
Cell
Update
Production Database & Rules/Strategies, RH, PH
Job-Shop PLC Control (OH/SH Execution, RH Status Monitoring)
Insert new SH & alternate
Supplier Data,
Cell
Supply- and WIP Reschedule
CAD, CAE (Customer Order
RH Status Monitoring (Robots, Cameras,
Station Compute
Holonic Bidding Mechanism (Create SH for depleted storage and alternate OH for WIP)
CAM, CAQC (OH/SH
Resource Controllers (RC,

Fig. 7. Information and data flow of HMES with knowledge-based Service Oriented Architecture for customer request management

After the definition of the process and production domains the fabrication cell is ready for utilization and the business process can begin. In order for this process to be made flexible it is proposed that the information on offer requests, offers to clients, order collection and production feedback is retrieved through a web interface which is interconnected with the process as described in Fig. 8 below:

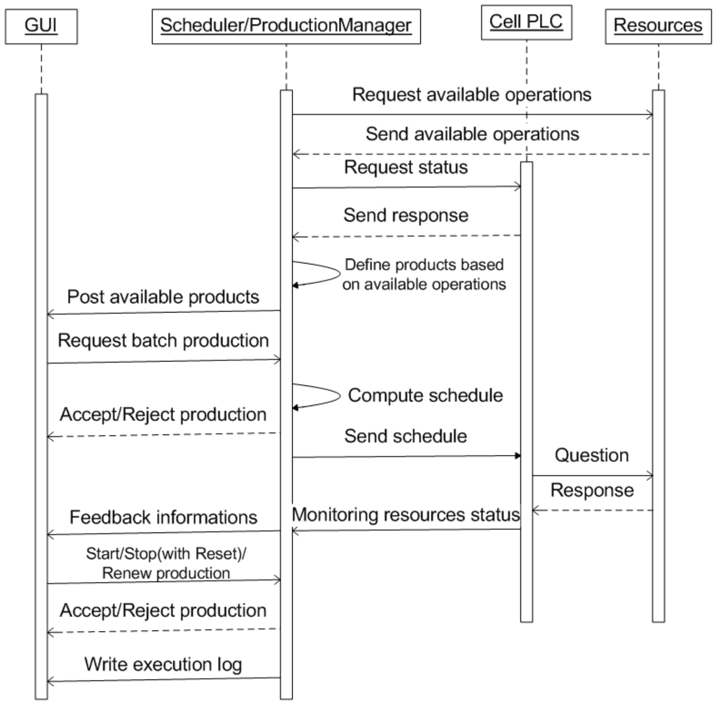

Fig. 8. Time diagram representing messages exchanged between entities from the business domain

Synchronization between the client interface and the planning and management application for the work cell will be done via the exchange of text files. There will be three types of files:

a. Input files for the scheduler

1. input_*nr_orders*.txt

For a command the client will provide the manufacturer with the following details: product types, quantities and priorities. There will be four levels of priority (0, 1, 2 and 3) where 3 represent the orders with the highest priority. Once this information is provided a connection between them and production domain must be created in order to report its state to the client. Therefore, besides the three fields that define a client order another field that will make contact with customer orders is added. This is the customer index.
In this way, the file has the following structure:

nr_products$priority$product_name $index_client

Here, nr_orders (from the file input_*nr_orders*.txt) is an integer which increases at each command. At each cycle of planning and production the application will retrieve the information from the file with the lowest index and then will delete the file.

2. command.txt

It is better that once the orders are introduced in production it exists a way to intervene in their execution. The reason is that for an undetermined cause the cancellation of orders' execution should be possible. Therefore, through this file that contains a single row (the command), messages will be sent to the scheduler and management application. The commands in the file are interpreted as follows:

start_production – represents the start command to the scheduler
stop_production – stops the production immediately and cancels all orders in work.
express_order – represents the command that stops current production, reads a new entry (input_nr_orders file) and then plans and executes the entire batch.

b. Output files from the scheduler
1. feedback.txt
This is a file that contains the state of the orders released for execution. This file has the following line structure:

product_typetip_produs, start_time_execution,stop_time_execution,client_index,state

having the following description: product_type – the type of product indicated by the representation; start_time_execution – the time at which the product execution has begun; stop_time_execution – the time at which the product execution will end; client_ index – the index that uniquely attaches a product to a client (e.g.: if there are two clients with the same product it must be identified to whom it belongs to); state – represents a product state as follows: failed_execution (the product cannot be executed due to lack of raw materials or resources), failed (the product can not be executed because of a malfunction that occurred during its fabrication), processing (execution in progress) and done (product executed).
This file will be analyzed with a frequency that permits sending the information to the client in real time.
At the end of the execution of a batch of products the file will contain two types of products: executed and non-executed. At every event in the system (resource failure or recovery) the scheduling is recomputed taking into account the products previously marked as compromised due to lack of resources. The non-executed products have reached this state due to two possible reasons: either they failed on the production line or there were no raw materials for their execution. For this reason the planning and management application checks one last time the "non-executed" status of products. If products still cannot be done they are finally rejected (e.g. a part is wrongly mounted because the palette doesn't arrive perfectly aligned with one of the robot's base axis; this is identified using machine vision, and the respective product remains marked as "failed").
2. lock
This is a temporary file that reflects the utilization state of the scheduler and management application: if it exists, the application is occupied with a previous order; if not, it means that orders can be sent to it. This file contains the date and time the system was blocked.
3. A response from the scheduling and management application side should exist to confirm that it is in a running/stopped state.

c. Storage files for memorizing execution logs
1. output_date_time.txt
Following the execution of a batch of products the resulting information is stored for subsequent processing. Thus there will be stored all information related to the traceability of products: operations, execution times, entry and exit times, visited resources and the final state of product (done/failed). All this information will be stored in text files whose names

start with the *output* keyword, followed by time and date when the file was written. Inside the file the following information will be found:

Order 1 with index 1 of type H
----------FAILED----------
Priority = …
Insertion time: …
Exit time: …
Processing time: …
Transporting time: …
Operations:
axe(on resource …, at time …), …

The FAILED field appears only if it is the case. If production is stopped to remove / add products from execution the log file will be written at the end of the production.

2. error.txt

This file contains all the failures and recoveries the system went through in the form of records with the following structure:

###
Date: the date the failure/recovery happened
Time: the time the failure/recovery happened
Error type: string that uniquely identifies the problem and the resource that was affected.

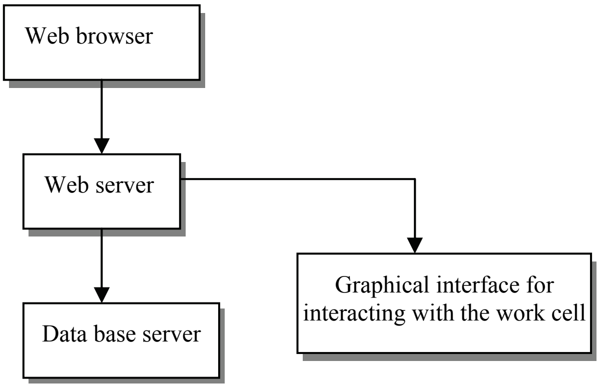

Fig. 9. Diagram of the user interaction interface

In the representation of Fig. 9, the Web browser is a common web browser like: internet explorer, firefox, opera, etc; Web server is the location that hosts the user interaction page; the Data base server is the data base server that contains the information representing client orders, products, etc.

The Graphical interface for interacting with the work cell is the module through which communication is done between the scheduling and management application and the client.

Fig. 10 presents the application modules. These are:

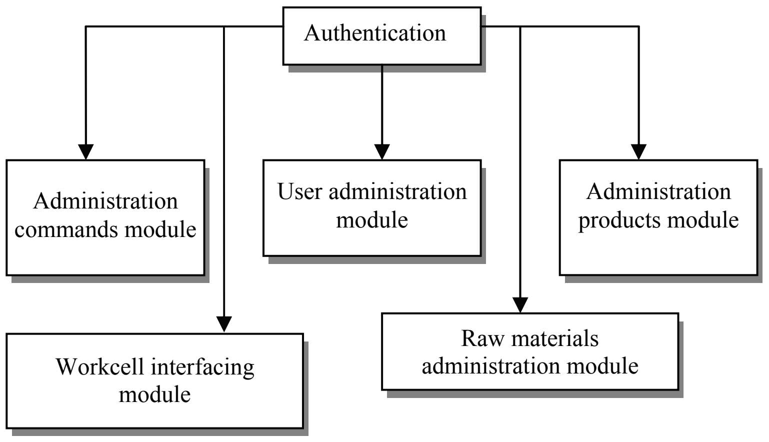
Authentication
Administration commands module
User administration module
Administration products module
Workcell interfacing module
Raw materials administration module

Fig. 10. Application modules

Authentication: login/logout and user administration module (for allowed zones and permissions).
Administration commands module: module that monitors the client commands currently in execution.
User administration module: module in charge of user personal data.
Administration products module: module in charge with creating a product
Administration products module: module in charge with the materials needed to create new products.
Workcell interfacing module: module needed for the communication with the application from the work cell.
Fig. 11 shows the data base structure, with the following components:

Users: a table that contains the authentication data for registered users.
User data: a table that contains user personal data address, telephone, etc.
Client orders: a table that memorizes the clients' orders already sent to execution.
Status: a table that contains the status of each client order.
Product-orders link: the link table that does the connection between the possible products to order and the orders sent to execution
Products: a table containing the list of possible products that can be executed by the system.
Products properties: a table that describes the products composition (operations to perform and precedence between them).
Products-materials link: the link table creating the connection between the list of materials and the possible products that can be executed by the system.
Materials: a table containing the existing materials that can be used to assemble products.
Materials prices: a table containing the cost of materials.
Materials properties: a table containing the description of assembly materials.
Measurement units: a table containing the measurement units for the materials used in the fabrication process.

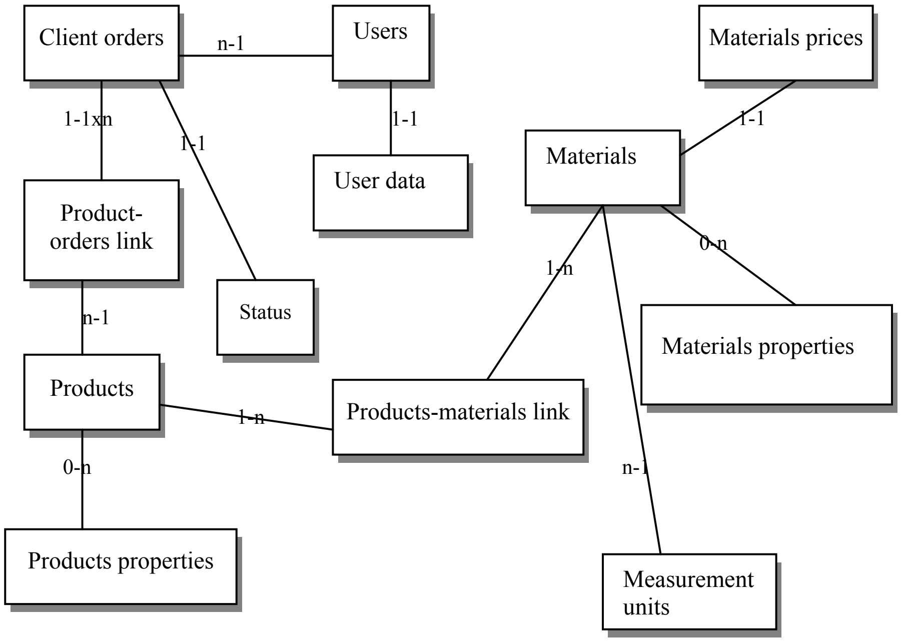

Fig. 11. Data base structure

4. Essential production processes

4.1 Step scheduler based on dynamic simulation

4.1.1 Scheduling production at batch level

Expertize Holons (EH) were defined and implemented as a set of rules creating an optimal schedule (maximizing the load of all available resources), which means that each of the four robots in the system 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. If the transport system is not blocked for most of the time each robot station will be always reachable, thus ready to receive an item and carry out a task.

Based on this idea a **Step Scheduler** has been developed. Each individual item (product) is being scheduled one step at a time. The process is initialized by generating a queue of all raw customer orders (products to be assembled – Fig. 4). Once the queue has been generated at production start up, failure detection or recovery from failure, the following algorithm is executed (an iteration of the algorithm checks and completes the following action steps):

Step 1. Check the number of pallets in the system, if there are less than 2 pallets, go to **Step 2**, else go to **Step 3**

Step 2. Choose any item randomly from the queue

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 capable of executing the task and calculate the waiting time for each robot before the task could be executed once the item arrives at the station

Step 2.3: Choose the operation with the 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 in the *conveyor simulator* and the robot operation execution; if a robot finishes an operation go to **Step 4**; if a pallet arrives at a workstation 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 completed 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 capable of executing the task and calculate the waiting time for each robot before the task could be executed once the item arrives 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. **Step 6**: For the arriving item store the current time index as operation start time, allocate this item on 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 criterion that the waiting time must be minimal) of alternating product types may be introduced into the system. Tests showed that different sequences (different runs of the algorithm on the exact same product definitions) yield slightly different total production times. For this reason, an alternative running mode has been integrated into the software. The so called Step Scheduler Best Sequence mode runs the algorithm 100 times and outputs the best result of these runs.

4.1.2 The Simulator scheduling tool

A Simulator has been developed and integrated in the global software system to assist and stepwise validate the creation of order holons list (i.e. the sequenced raw order holons). The simulation program routines play an essential role in the scheduling process, both at:

- Production start up, detection of a resource failure, and recovery after failure (off line, preparing production)
- Tracking of production execution, graphic monitoring (real time during production execution)

The main quality of simulating the transportation of products on pallets is the capability to vary the time base. The software furthermore uses a *transportation time matrix* which has been 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 (transport time slice) corresponds to one advancement step of a pallet and was defined as 0.5 seconds.

The transport simulation is used off-line to generate global production schedules, and in real time to track production execution. There is a fundamental difference in the use of core routines developed to realize the correct transportation of pallets. In the case of the 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 seconds has passed by and only after that allows another iteration of the main program loop. The result of this pause is the fluent running in 0.5 second steps of the simulation which, in combination with the measured transportation time matrix, corresponds exactly to the behavior of the real system

When these routines are used to solve the scheduling problem, they transport the pallets in the system with an infinitely high speed (limited by the computer calculation speed). As soon as an iteration of the transport functions terminates, the next one starts. Since none of the dimensions or the transportation time matrix has been changed for this kind of simulation, the resulting time indexes still correspond exactly to 0.5 seconds and may directly be used to define the production schedule.

The only difference is that time has been compressed at maximum by doing the calculations in absence of a timer which ensures the realistic execution of the simulation. The simulation functions check at each iteration all the pallets which are currently in the system. A pallet gets transported one step if the conveyor segment is running and if there is no active stopper or elevator at the pallet's present position.

Certain constraints given by the cell architecture ask for another control layer which ensures that odd situations do not occur while the system is operational. Since there are four robot stations in the cell, the number of pallets with products circulating in the system was limited to four to five (including the one just leaving the production system).

4.2 Failure management and system integration

The fail-safe mechanism for controlling production is triggered whenever a resource (robot controller, sensor, video camera) is down or the result of an operation is negative (visual inspection). With the help of the basic holons RH, PH, and OH and the scheduling algorithm based on EH, a *FailureManager* was created. A virtually identical counterpart, the *RecoveryManager*, exists, which takes care of the complementary event when a resource recovers from the malfunctioning and comes back online.

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 $P - p_{\mathrm{T}} - p_{\mathrm{E}}$ remaining products, $p_{\mathrm{T}} =$ 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. 12.

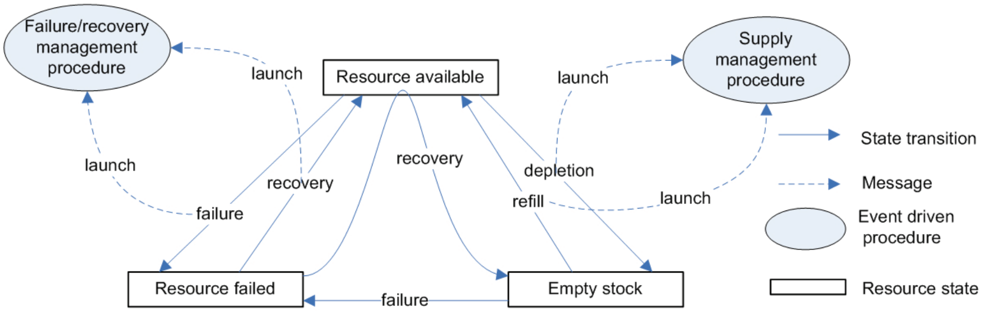

Fig. 12. Actions taken when a resource is changing from a state to another

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 differently (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. 13). 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 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.

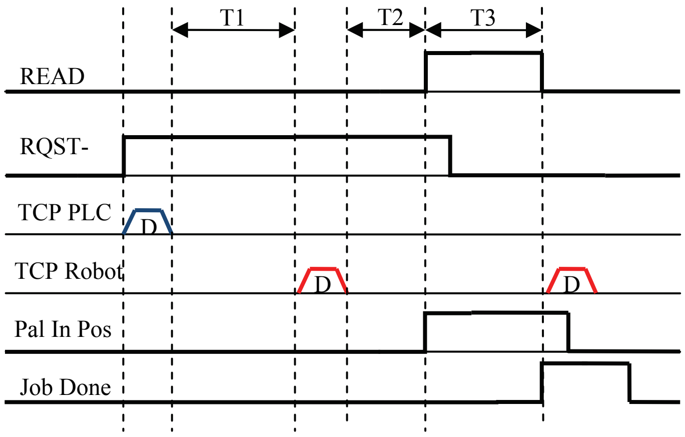

Fig. 13. Communication protocol between the PLC and a Robot Controller for authorizing an OH operation execution

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).

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. 14):

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*;
- 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, $n_{wip} \leq 5$), 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.

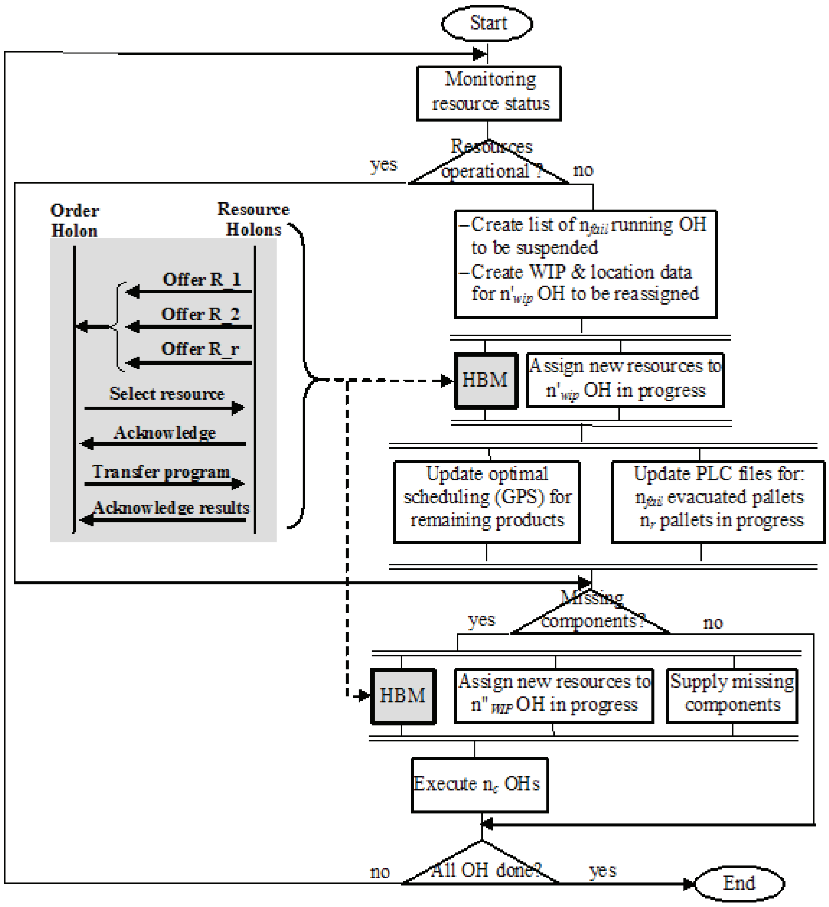
Start
Monitoring resource status
Resources operational ?
yes
no
Order Holon
Resource Holons
Offer R_1
Offer R_2
Offer R_r
Select resource
Acknowledge
Transfer program
Acknowledge results
–Create list of n_{fail} running OH to be suspended
–Create WIP & location data for n'_{wip} OH to be reassigned
HBM
Assign new resources to n'_{wip} OH in progress
Update optimal scheduling (GPS) for remaining products
Update PLC files for: n_{fail} evacuated pallets n_r pallets in progress
Missing components?
yes
no
HBM
Assign new resources to n''_{WIP} OH in progress
Supply missing components
Execute n_c OHs
All OH done?
no
yes
End

Fig. 14. Dynamic OH rescheduling at resource failure/storage depletion with embedded CNP job negotiation

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.

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 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).

4.2 Automatic re-supply of workplaces

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 re scheduled 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 re scheduled by the holonic bidding mechanism (HBM) to robots disposing of necessary assembly parts.

A lock is put on the system, and no further OH (a new pallet) 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 re-scheduling being not necessary.

4.2 Treatment of rapid 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 Order Holons).

Because of the similarities between a task run on a processor and a batch of orders executed in a cell (both are preemptive, independent of other tasks/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 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 analysis 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 beginning 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, and $\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1, n = $ number of tasks, $C_i =$ execution time, $T_i =$ cycle time (a set of n periodic tasks can be scheduled if, 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 a periodic. This means that instead 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 makespan, if $\sum_{j=1}^{i} \text{makespan}_j \leq \text{delivery\_date}_i, i = \overline{1, n}$, 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. 15):

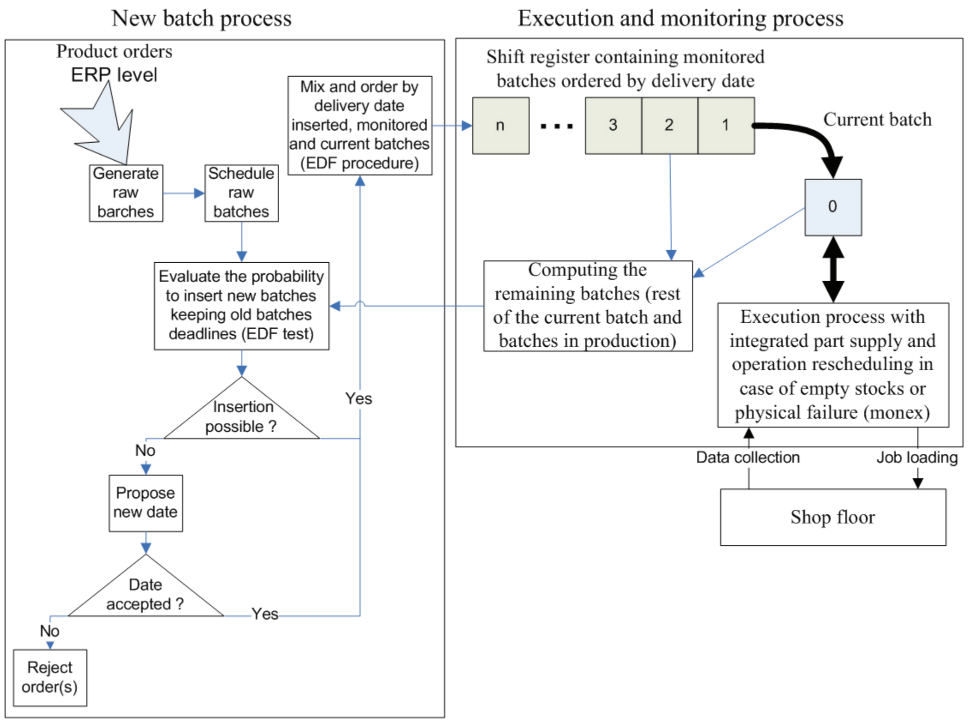
New batch process
Execution and monitoring process
Product orders
ERP level
Generate raw barches
Schedule raw batches
Mix and order by delivery date inserted, monitored and current batches (EDF procedure)
Evaluate the probability to insert new batches keeping old batches deadlines (EDF test)
Insertion possible ?
Yes
No
Propose new date
Date accepted ?
Yes
No
Reject order(s)
Shift register containing monitored batches ordered by delivery date
n
...
3
2
1
Current batch
0
Computing the remaining batches (rest of the current batch and batches in production)
Execution process with integrated part supply and operation rescheduling in case of empty stocks or physical failure (monex)
Data collection
Job loading
Shop floor

Fig. 15. Add *rush order* diagram and integration with dynamic job re-scheduling based on CNP negotiation

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. Analyse 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 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 one 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).

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

Table 1. The minimal structure of a *batch holon*

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 requested delivery date.

5. Experimental results

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.

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 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.
An example of production definition at batch level for four products (H-, U-, L-, and C-type products) resulting from the succession 8 operations consisting of assemblies, milling and visual inspections.
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):

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*iF*)	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)

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

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

6. Conclusion

The scope of this chapter was the definition of a PLC-centred framework for developing an integrated solution aiming at controlling the resources from a flexible manufacturing system and at the management of the clients' orders. The key characteristics of the proposed framework are autonomy of the control systems' resources and the cooperation between them.
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. Features 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 (RSRP) 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.

7. References

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

Bongaerts, P., Monostori, L, McFarlane, D., Kadar, B., 1998. Hierarchy in distributed shop floor control. Proceedings of the 1st Int. Workshop on Intelligent Manufacturing Systems IMS-EUROPE, Ed. EPFL, Lausanne, 97-113

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

Borangiu, T., Ivanescu, N., Raileanu, S., Rosu, A., 2008. Vision-Guided Part Feeding in a holonic Manufacturing System with Networked Robots, Proceedings of Int. Workshop RAAD 2008, Ancona, Italy

Borangiu Th., Gilbert P., Ivanescu N., Rosu A., 2008. Holonic Robot Control for Job Shop Assembly by Dynamic Simulation, Int. Conference MED'08, Ajaccio

Borangiu Th., Gilbert P., Ivanescu N.A., Rosu A., 2009. An Implementing Framework for Holonic Manufacturing Control with Multiple Robot-Vision Stations, Engineering Applications of Artificial Intelligence 22 (2009), 505-521, Elsevier

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, Ed. Pro Literatur Verlag Robert Mayer-Scholz Germany, Vienna

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

Dorigo, M., and Stuzle, T., 2004. Ant Colony optimization. The MIT Press

Koestler, A. The Ghost in the Machine. Hutchinson publishing Group, London, 1967

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

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

Maione, G., and Naso, D., 2003. A soft computing approach for task contracting in multi-agent manufacturing control. Computers in Industry, 52, 199–219

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

McFarlane D, Sarma S, Chirn Jin Lung and Wong C Y, and Ashton K,. 2002. ''The intelligent product in manufacturing control and management''. Proceedings of the 15th Triennial World Congress, Barcelona

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

Nylund H., Salminen, K., Andersson, P.H., 2008. A multidimensional approach to digital manufacturing systems, Proceedings of the 5th International Conferebnce on Digital Enterprise Technology, Nantes

Okino, N., 1993. Bionic Manufacturing System in Flexible Manufacturing System: past – present – future. J. Peklenik (ed), CIRP, Paris, 73-95

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

Ramos, C., 1996. A holonic approach for task scheduling in manufacturing systems, Proceedings of the IEEE Int. Conf. on Robotics and Automation, Minneapolis, USA, 2511-2516

Sallez Y., Berger T., Trentesaux D., 2009. Open-control: a new paradigm for integrated product-driven manufacturing Control, Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM '09), Moscow

Sauer O., 2008. Automated engineering of manufacturing execution systems – a contribution to "adaptivity" in manufacturing companies, 5th International Conference on Digital Enterprise Technology Nantes

Sha, L. et al., 2004. Real Time Scheduling Theory: A Historical Perspective, Real-Time Systems, Vol. 28, No. 2-3, 101-155

Trentesaux, D., Dindeleux, R. and Tahon, C., 1009. A MultiCriteria Decision Support System for Dynamic task Allocation in a Distributed Production Activity Control Structure. Computer Integrated Manufacturing, 11 (1), 3-17

Usher, M.J., Wang, Y-C., 2000. Negotiation between intelligent agents for manufacturing control, Proc. of the EDA 2000 Conference, Orlando, Florida

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L. and Peeters, L,. 1998. Reference architecture for holonic manufacturing systems: PROSA. Computers in Industry, 37 (3), 255–274

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

Zbib, N., Raileanu, S., Sallez, Y., Berger, T. and Trentesaux, D., 2008. From Passive Products to Intelligent Products: the Augmentation Module Concept. Proceedings of the 5th International Conference on Digital Enterprise Technology, Nantes