ON HOLONIC MANUFACTURING SYSTEMS DESIGN AND IMPLEMENTATION

BY

DORU PĂNESCU, GABRIELA VARVARA and MARIUS ȘUTU

Abstract. Some results of an on-going research on Holonic Manufacturing Systems are presented. This field refers a collection of methods and techniques seeking to enhance the performance of the manufacturing systems control. The considered approach is derived from a holonic reference model, namely PROSA architecture. Certain additional elements are proposed for the respective method, especially regarding the coordination and implementation issues. Some possibilities to obtain a better management of a distributed manufacturing system are discussed and the appropriate communication mechanism is sketched. A software implementation option that is offered by a specialized agent programming environment is illustrated.

Key words: Holonic Manufacturing System, PROSA Architecture, Holon Coordination, Service Oriented Operation, Agent Based Programming.

2000 Mathematics Subject Classification: 93C83, 68T27.

1. Introduction

Holonic Manufacturing Systems (HMS) represent a topical research field that aims at finding a solution to some challenges for the present manufacturing environment. Concisely, it is harder and harder to have an optimal control architecture for a manufacturing process whose inputs can have great variations (e.g. the production refers to small series and the custom commands require important changes for the manufacturing design and control), to comply with the time constraints and to operate with the smallest costs. A holonic approach can be the answer that results by applying in a synergetic way certain methods and tools from Artificial Intelligence (AI), Control
Engineering, Software Engineering and Mechatronics. Thus, the term holon was for the first time used by Koestler in order to better formalize the relationship between a part and the whole [1], [2]. Starting with Koestler’s idea that a new model is needed to describe the hybrid rapport between wholes, sub-wholes and parts in real-life systems, a new methodology has been developed regarding the manufacturing systems, under the name of holonic approach. Its attractiveness results from the way it provides means for combining both the hierarchical and heterarchical schemes, and the possibility to consider important features for the present manufacturing systems: autonomy, flexibility, adaptability. The problem still existing for the HMS is the lack of systematic methods for the designing and implementation phases, as well as the little practical experience. Thus, these aspects are the main points for this contribution.

As above mentioned, HMS suppose that a combination of ideas from conventional programming and control are used together with some concepts of AI. Thus, the main entity of HMS, namely the holon, is characterized by autonomy, co-operation and reasoning abilities [2], [3], which are the main features for the agent concept in AI, too [4]. In this way agents were thought as constructing blocks for HMS and the distributed nature of manufacturing together with the need for cooperation of the involved entities determined the use of multiagent systems. Even though, holons versus agents and HMS versus multiagent systems can be certainly individualized. For holons a new feature appears, namely the self-similarity of their structure, conducting to a recursive based approach [5]. To use agents for HMS implementation an appropriate composition method is needed, and this cannot be entirely based on the multiagent approach, due to the recursive holonic character and also due to the specific way holons are supposed to cooperate, in a service oriented manner [5]. The holonic requirements can be solved by a semi-heterarchic architecture, different from the classical multiagent systems’ schemes. The way the services offered by holons are to be used and combined has to be related with object oriented techniques and specific communication mechanisms [3], [5], [6]. Meanwhile, HMS are always combined with a robot based implementation, because robots can be the execution units providing the needed autonomy and flexibility. Thus the inter-disciplinary character of HMS is made clear.

The presented contribution starts from a holonic reference scheme, known as PROSA architecture. The respective method does not entirely clarify the coordination and communication issues that are difficult problems for a manufacturing system; it has to face the distributed character of its organization and to solve manufacturing tasks by dynamically establishing cooperation relations between the production units. These points are further analysed and some solutions are proposed. The contribution ends with an implementation variant based on a specific agent programming platform and a few conclusions. It is to notice that the proposed method is being tested on an experimental didactic manufacturing system, which benefits by the use of real industrial equipment: two 6 degrees of freedom industrial robots, a machine tool, a
conveyor, a computer vision system and several storage devices; this makes the respective system a sound benchmark for the manufacturing control systems [7].

2. The Main Points of a Reference Model: PROSA

A widely accepted HMS architecture is provided by the PROSA reference model [2], [3], [8]. This ascertains that four types of holons are requested by any kind of manufacturing planning and control system, and establishes the type of knowledge each of them copes with. These are the resource, product, order and staff/expert holons. Their existence can be correlated with the specific manufacturing phases. Starting with the customer’s command there should be an order holon that is responsible for managing it. Thus, the order holons are able to react to the commands entering the manufacturing system and process them. This means they must follow the evolution of each item through the manufacturing process, by keeping a model of its progress until the initiating goal is satisfied. The product holons contain knowledge on each product manufacturing cycle, i.e. they had a corresponding model for each item that can be an output of the manufacturing system. It is to notice the difference between the product and order holons: the first contain a product model (as a pattern on how each product can be carried out), while the second keep a product state model, i.e. a model on how an explicit object evolves [3]. The resource holons are related to the manufacturing execution devices, incorporating knowledge on each equipment control. One can observe how these three categories are the necessary and sufficient entities to manage any manufacturing process. Due to the distributed character of both the resources and control units, a heterarchical control scheme would result with some drawbacks well established for this kind of approach [9]. If one thinks about the multiagent implementation then he discovers the way agents with local knowledge are not able to come out with a global optimal solution. That is why the fourth holon category, namely the staff holons may be considered. They can contain global methods regarding the optimization of planning and control processes and act as consultants for the other holons; this explains their name – staff holons. It is to emphasize that the order/product/resource holons make the decisions on the manufacturing process planning and control and only apply for the expert holons’ support when needed. Depending on the time constraints, the output of the staff holons may be used or not by the other holons. In this way the semi-heterarchic architecture for the HMS is justified.

Each holon possesses an information processing part and a physical processing part [10]. In manufacturing the holons should have the ability to manage the changes in their environment. This can be obtained by a proper holon’s structure and the way the holons coordinate and organize themselves in holarchies. As about the holon’s architecture, the generic scheme considered in the presented approach is shown in Fig. 1.
The holon’s decisional part is materialized under the form of a Jack agent (Jack being the used programming environment). This part is determining the holon’s behaviour as it chooses the performance of the holon, namely the actions to be done by the structural component. Because the main point in this research refers to the HMS inferential part the syntagma holon/agent will be used to denote both the holon and its corresponding agent; for the decisions to be taken during the manufacturing process the reference to the holon or the encapsulated agent makes little difference. The structural component (named also the embodiment part) encapsulates the holon effectorial section. It is the part that allows the recursive holon feature to be manifested, as it can be a standalone execution device in the case of certain resource holons (e.g. a robot, a machine tool, etc.), or it can contain a group of holons, i.e. a holarchy. Such a structure provides a composition of actions or services for the decisional part, resulting in an encapsulation of the manufacturing capabilities at different levels. This can be the case for a product or an order holon that takes decisions to be put into practice by several entities, which also have their own decisional (control) and operational parts, being holons by themselves. Thus, it is further explained the combination of hierarchical and heterarchical approach for the HMS. In this context the HMS structure is mainly dependent on the way the holarchies are formed. The principles to be considered that are further explained in the next paragraphs regard: a goal directed operation for the manufacturing planning and control processes, a service based activity, i.e. various holons offering and requiring certain services depending on their own goals that conducts to the formation of holarchies, the suitable reactivity to the environment changes, this including the cases of equipment failures. To put into practice these principles appropriate coordination and communication procedures should be set-up.

3. The Coordination Issues for the HMS

An important issue for the HMS, which must be considered since the analysis and design phases, refers to establishing the relationships between the holons in the system and the necessary coordination mechanism to manage them. The already discussed holon classification and the knowledge patterns the
holons handle are to be taken into account in setting up the holons’ relations. As already settled, for each holon the decisional part is represented by an agent so that the holon interactions must be related to the multiagent systems coordination mechanisms. The agent operation and its social abilities are dependent on the implementation method. The chosen scheme for the proposed approach is the Belief-Desire-Intention (BDI) architecture [4], [11]. This is inspired from the human reasoning process, that supposes an interleaving between a deliberation phase when the goals to be undertaken are decided and a means-ends reasoning phase when the ways the decided goals are to be achieved is determined. This method is appropriate for industrial applications, as it implies a periodical review of the agent’s goals in order to opportunistically take advantage of the changes in the environment and be able to consider the resource-bounded nature of the agent reasoning mechanism. Thus, an agent operates with beliefs (instead of tautologies), namely the agent’s current knowledge on its environment and goals. The agent’s options in satisfying its goals turn into desires, which follow a filtering process in order to become intentions. These assume a commitment process: an agent should commit to some of its objectives and try appropriate actions until obtaining the successful output. When BDI agents are involved, an event based solution is the natural choice: so, it is important to notice the types of events to be considered:

1. Goal based events that appear when certain products have to be manufactured. They can be either externally generated (e.g. when a custom command is received), or internally produced by the HMS in the procedure of goals’ processing and decomposition.

2. Sensorial based events that signify changes in the manufacturing environment. These are important in providing agents’ feedback in order to obtain their reactive behaviour. A certain dependence exists between the two types of events; a holon can opportunistically consider the new circumstances in the environment to handle a goal.

It is interesting to relate this event classification with the one considered by the HMS and the Jack BDI agents.

a) External events refer to those that are produced within the environment where the holons activate. These can be the result of outside factors, like a change produced by a human operator, but also events that are the consequence of holons’ actions. For example, a robot holon’s action of placing a part on the conveyor is an external event for the conveyor holon that has to take a decision regarding the part moving and sending a message to another robot holon, which is supposed to take the part from the conveyor. From the Jack implementation point of view this kind of events are considered as being ephemeral events. The justification has to be related with the type of knowledge implied by these events, namely aposteriori knowledge [12]. Such knowledge is dependent on the sensorial acquisition that may be imperfect and can be contradicted in future. In consequence, when a BDI Jack agent does not succeed
to solve such an event in the first attempt will drop it from the event queue; depending on the next sensorial acquisition phase the respective event may be or not reconsidered. By comparison with the previous classification, ephemeral events correspond to sensorial based events.

b) Internal events are produced by agents in the decisional process, in accordance with the inference mechanism. They mainly refer to messages sent by an agent to the other agents in the multiagent system, representing a goal (sub-goal) to be solved; thus they correspond to goal based events. According to the BDI principles, these events have a permanent character, meaning that agents commit themselves in solving the internal events. They do not drop the event after the first attempt when this fails, but agents should try alternative plans, and keep the event in the queue until it gets a successful solution. It is clear that such an approach can determine the necessary robust operation for the HMS.

In conclusion, according to this classification the events can be ephemeral or persistent. This should have certain influence on the HMS operation. Thus, the holons themselves can be of the two types: ephemeral or persistent. Indeed when a new manufacturing goal appears, a new holon must be created that should be responsible with the respective goal fulfilment, having a limited existence. It is to observe that the proposed implementation solution admits such an approach. The agent that will determine an order holon arrangement can be created at a certain time (as explained above, when a manufacturing goal appears), by being instantiated from an agent class (pattern). The instance agent inherits some general characteristics (in our case the ones applicable for the order holons), while being completed with specific elements determined by the generating event. The permanent holons are created off-line, in contrast with the previous case that implied an on-line generation. This is the case for the resource holons. Their agents are determined at the manufacturing system layout establishment, being endowed with the corresponding features of the controlled resources. Even for this category the HMS must be flexible with respect to the cases when new resources are added, but this updating process should be an off-line one.

Taking into account all the above mentioned issues and having as a guideline the service oriented operation, the proposed architecture conducted to a pattern for the holon interactions that is sketched in Fig. 2. The arrows are numbered according to the order of events. The starting event (denoted with 1) corresponds to a manufacturing goal appearance. This event is sensed and the consequence will be the creation of an instance for the order holon/agent that should contain all the manufacturing goal specifications (the arrow marked with 2). The respective holon/agent will be responsible with the fulfilment of the goal that determined the holon formation.

The order holon/agent will be able to achieve the goal by making use of the product and resource holons, during the solutioning process the adequate holarchies being organized. The principles that were considered to get the
coordination mechanism were:

1. a service oriented solution, with the necessary communication procedure based on the CORBA (standing for Common Object Request Broker Architecture [13]) standard;
2. the holarchies are created and broken up depending on the current goals and the context of the manufacturing environment, by the means of appropriate multiagent systems organization;
3. the management within a holarchy is solved by applying an adapted form of the contract net protocol, the usual multiagent system coordination scheme.

The proposed method combines the top-down and bottom-up approaches, which is justified for an HMS that merges the hierarchical and heterarchical control. Regarding the order of events in Fig. 2, the order holon/agent launches a service request (event marked with 4) according to the specifications of the goal which it is in charge with. Until this moment the approach is a top-down one, conducted by the goals to be solved. Afterwards the bottom-up operation is implied, by taking into account the available resources and the distributed character of HMS. The communication is managed according to the CORBA standard, having as a main part an ORB (Object Request Broker) hub, that serves the client-server relations for all the existing objects, these being the agents in our case [14], [15]. The order holon/agent request is transferred to the appropriate product holon/agents. The CORBA techniques facilitates this process, as the entity launching the request must only know about the existence of a product holons’ community, that should possess knowledge on the products’ manufacturing, but the respective entity must not know which the precise product holons are or how many holons are in the respective category. The communication between the producer and the consumer is entirely anonymous. In this way, by combining the agent based approach and the CORBA service oriented architecture an important flexibility is obtained. The order holon/agent has only to transmit the service request, and this will be solved according to the current manufacturing context, which is handled by the product and resource holons.

Between the entity that is making a request and the ones solving it a trading object service is interposed according to the CORBA standard, as it will be further explained. Such a mechanism is very general, being independent of the number and the specific features of the relations between holons. The first step in this procedure is the communication of a service availability, that is made in our case by the resource and product holons (the arrows named as Event 3 in Fig. 2; normally this announcement is prior to the service requests, which explains the order of events 3 and 4). The details of the communication between the service importers and exporters are not given in the above scheme (some points are to be discussed later), for the HMS being significant the establishing of the entities’ roles in the coordination process. A specific issue for the HMS management appears when several holons are potential providers
According to the CORBA service oriented principles, a service offer is accompanied by specifications regarding the service functionality and the quality that is warranted for the respective offered service. This information can be used by the manager of the contract net protocol, which is the agent of the order holon in the discussed case. As represented in Fig. 2 by the arrow of Event 5, the order holon is informed about the product and resource holons that can solve the issued goal and these holons/agents play the contractors’ role in the contract net protocol. From the specifications of the offers their cost can be calculated. Two cases are possible: the cost is already sent as an offer parameter, or it should be determined by the manager using certain offer characteristics. Both cases can appear in an HMS. Some services have their costs directly quantifiable by the contractor agent, usually when elementary goals are involved, i.e. the ones to be solved by a single operation. For example, the agent of a robot holon (as a resource holon) can determine the cost of an elementary action (e.g. the movement of a part) by making use of the information provided by its structural component. Generally, in the HMS the parameters to be considered for the offers’ costs calculus refer to: the earliest time possible for the operation start, the estimated finish time, implied consumption [16]. For complex goals, like the case of assembling a product, the manager agent is supposed to make the cost calculation based on the information provided by the contractors referring to various assembling

Fig. 2 – The pattern for the holon interactions.
Independent of the way the costs of the offers are determined, the manager holon/agent assigns the contracts to the chosen contractors; this last stage is represented by the arrow marked as Event 6 in Fig. 2. It must be pointed out that phases denoted by the events 4 to 6 suppose decomposition and a message exchange operations that include the product and resource holons. The product holons can be managers for the resource holons, but even between these a hierarchy may be established. For example, in the considered manufacturing system the robot holon was manager requesting a service from another resource holon, namely a vision system. The efficiency of the architecture is an important feature: the same coordination and communication mechanism, once designed and implemented, can be used for all the levels of the manufacturing system. The recursivity, as a distinct feature of the holonic approach is got, too. Every holon may contain some other holons (a holarchy) as its structural component, and all the holons use the same coordination principle, as explained above. The obtained holarchies encapsulate the decisional and operational capabilities requested by the manufacturing system. Especially for the planning process, that is intensively decisional, the proposed approach can be the solution to limit the complexity and to provide the necessary flexibility to the manufacturing environment changes. It is to mention that the solution is based on some known tools (multiagent systems, contract net protocol, CORBA), but these are adapted to certain manufacturing process features.

In HMS, besides the recursive character of the architecture, the respective property is applied in settling the coordination. It is about the way a request is recursively solved on several hierarchical levels. Consequently, the contract net protocol must be modified to face the recursive structure of the HMS [16]. This is clear the case for the considered experimental manufacturing system, as a request from an order holon is first decomposed by the product holons/agents and then further divided at the resource holons/agents level. The result to the initial request must be recursively composed from the partial results of the holonic scheme. This can be put into practice by combining the extended version [17] with the recursive (or nested) version of contract net coordination protocol [16]. A situation from the
considered manufacturing system that supposes such an approach is the one presented in Fig. 3. This reveals the case of an agent – the one corresponding to a product holon – that is both contractor and manager. It is contractor for the agent of an order holon and manager for some agents of resource holons. The order holon/agent is requesting solution (the corresponding service) for the goal denoted G1. It gets the offer from the agent of a product holon named Pr1 that has the knowledge (plans in the BDI architecture) on the respective product manufacturing. In fact, the offer is constructed after the product holon/agent makes the goal decomposition (the plans it possesses provide the necessary means) and asks the resource holons to solve the goals G2 and G3. The offers for these sub-goals are received from the resource holons representing the robots Rob1 and Rob2. Even these holons/agents are not able to entirely solve the requested goals, but ask and receive offers for two sub-goals (G4 and G5) from other resource holons/agents, namely one corresponding to a storage device Str1 and the other to the computer vision system CV. The robot Rob1 needs a part that can be delivered by the storage device, while the robot Rob2 needs a part to be identified by the vision system. This example illustrates again the double crossing of the decomposition tree: one (top-down) for the goals’ announcement and dividing, and the other (bottom-up) for the solution formation. As a consequence, the agents have to keep specific information in order to be able to create the solution: the name of the agent that initiated a goal and respectively the name of the contractor agent that is chosen to contribute; these pieces of information were represented in Fig. 3.

From the communication point of view the goals passing and the holons/agents structuring suppose some additional aspects that are further discussed.

### 4. Some Points Regarding the HMS Communication Mechanism

In order to design a communication mechanism for an HMS certain architectural constraints have to be considered. The holonic system has to support the recursive organization and a service based interaction protocol, as presented above. Considering the holon as a composite entity, spatially and functionally distributed, a solution for the implementation is to use a communication middleware to transparently ensure the support for addressing applications running on different computers, publishing services and managing events in the distributed system. CORBA contains a set of standards for implementing object request brokers (ORB), the core of distributed middlewares, and adjacent services. By making use of the CORBA specific services, the holon entity can be built around an ORB which handles the communication between the software agent and the structural component; in this way the previous scheme of Fig. 1 is materialized according to Fig. 4.

The structural component can be either a physical execution device commanded through a driver application for certain resource holons, or a holarchy formed from other holons for the upper level holons in the semi-
heterarchic organization. The communication protocol for the information flow

![Fig. 4 – A holon building scheme.](image)

between the holon components has to be transparent related to the communication layers, so the agent will only have to know the parameters required by the structural component’s communication interface. The agent, as being the holon decisional part, is responsible for finding plans in order to satisfy the requests received from other holons, using the ORB component to discover available services which can implement the actions corresponding to specific plans. The ORB component facilitates the service oriented approach in the distributed environment. A service is a mechanism used by a server entity, usually a software application, to export a certain functionality which can be requested by one or more client entities, also software applications, by making use of platform independent interfaces granting communication protocol transparency. A service is implemented as a method exported by an object instantiated by a server application, and in order to be referred by client applications the object has to be addressable. The ORB application maintains a list of all available objects and makes them public through a set of global services, as follows [14]:

1) The naming service is a component of the ORB application which transparently associates a reference to each object published by any of the connected applications, disregarding of software application location or network protocol used to communicate with the ORB [18]. A client application can retrieve the object reference and directly call the service implemented with the specific object.

2) The event service is another component of the ORB, managing the events generated by the inter-connected software applications in the distributed system [19]. One entity can register as producer for an event channel, publishing events for eventual consumer applications. The consumers register with specific channels and they are transparently notified in real time upon event generation by the event service. The transparency is given by the way a producer application doesn’t have to maintain itself a list of consumers and implement peer-to-peer notifications.

3) The trading object service is complementary to the naming service and implements the support for dynamically establishing producer-consumer relations in distributed systems [14], [20]. The producers (agents or driver
applications in the proposed architecture) publish service offers using the trading service, and the consumer applications can query the trading service in order to find available services to fulfil required actions; this facility is to be combined with the multiagent specific coordination mechanism, as already explained.

In a further detailed view, to implement the communication for a composite resource holon with four distributed components (Fig. 5), an initialization phase is required, with the following steps:

a) The connection to the physical device is the first step, when the device becomes operational in the manufacturing process. The corresponding driver application initiates the permanent connection with the device, using specific protocols matching the device’s inter-connecting capabilities. If any booting sequence or calibration operations are required upon device start-up, then they are executed immediately after the connection becomes active.

b) The connection to the HMS takes place next, when the driver application registers with the ORB all the services specifying the physical device’s functionality according to a convenient granularity level, using the trading service.

c) The driver application registers as producer in the event channel corresponding to the resource holons and generates an event to notify the specific software agent.

d) Finally, the agent is being notified receiving a goal in the manufacturing process.

Fig. 5 – Communication channels in the holonic system.
After the initialization phase, when a goal is received the agent interrogates the trading service to identify the service solving the goal and then in accordance with the coordination procedure, as previously mentioned, calls the chosen service. On a service request, the driver application initiates the execution sequence on the physical device by transmitting a routine code or the necessary parameters to the program running on the device, or by changing the appropriate digital/analogue outputs within the corresponding controller, PLC, etc. Then, the agent fetches and evaluates the called service’s result and enters a stand-by state waiting for a new goal at a successful result, or starts a diagnosis sequence at a failure outcome.

As about the inter-holon communication (see Fig. 5), the event channels are initiated at ORB application’s launch in execution and the agents register afterwards when the multiagent system platform starts up. The only a priori knowledge the agents need related to the holon to whom they belong is the driver application descriptor for the corresponding device and the event channel descriptor. Given the fact that software agents run within the same Jack agent platform, internal communication channels can be used with enhanced reliability for the inter-holon communication. Indeed, no network delays or communication protocol specific issues can happen between parts communicating within the same platform, and local channels can be used for synchronization, when more holons/agents need to cooperate in solving a task.

The organization of the inter-holon communication mechanism is important for the efficiency of the HMS. In usual scenarios (see the example in Fig. 3) an order holon encapsulating a global goal in the manufacturing system can have as structural component a holarchy formed from one or several product holons, which contain the operational context to manufacture the product types referred in the goal, and each of them uses a set of resource holons. This holarchy will be the result of the coordination mechanism. A typical approach is to define the relations between the agents corresponding to different holons and also the specific communication channels either a priori, at design time, or in real time if the agents know each other’s capabilities and dynamically establish group relations by a negotiation process.

Both these cases imply a knowledge coupling between one agent’s abilities/roles in the holonic system with the other agents’ abilities, so the overall system flexibility exclusively depends on the negotiation protocol’s performance. The decoupling could be obtained by transferring the bias towards a service oriented architecture, when the communication between holons on different hierarchical levels is implemented with services (Fig. 6). The holons/agents should have knowledge only of their own role in the manufacturing process, while discovering the other holons/agents’ functionality is mediated by the ORB application using the trader service. As a consequence, each holon/agent exports its own functionality by registering services with the trader, while the other holons/agents can query the ORB trader service when appropriate and choose a service from the list of services available at the
specific time.

When a holon/agent calls two or more services offered by different holons/agents in order to solve its current goal, the holons/agents making available those services will dynamically form a group relation and aggregate to become the structural component for the caller holon. Fig. 6 illustrates a case when an order holon applies for two product holons and each of these constitutes its own holarchy by making use of appropriate resource holons. Recursively applying this mechanism up to the highest hierarchical level (shop floor holon, company holon), the HMS becomes highly auto-organizable and gains adaptability to faults at equipment level. Each holon/agent will be able to choose alternative services if they are available when a specific device determines the invalidation of its own services and the other services depending on them. To put into practice these principles appropriate agent based
programming tools are needed, as further detailed.

5. On HMS Jack Agent Based Implementation

For the modern control applications the object-oriented programming methods and the classical client-server distributed development are still difficult to implement, modify and maintain. Moreover, these technologies do not offer the flexibility and adaptability to the environment changes. The respective issues can be surpassed by an appropriate agent-oriented software architecture tailored to the holonic approach. The agent – as a decision entity - is completely responsible for achieving its own goals, reacting to the environment events and communicating with other agents from the system. As already mentioned, the BDI agent architecture provides further means useful for a holonic system. All these justify the election of the JACK™ Intelligent Agents platform, developed by Agent-Oriented Software Group [21], [22]. It comprises an agent-oriented programming language built on the top of Java language, supporting the BDI reasoning method, an agent execution environment and the design tools for visual modelling (design and debugging views, facilities for building teams hierarchies).

Based on Java platform, JACK™ agent-oriented language has attractive features in the direction of source code portability and direct access to an impressive number of existing libraries [11] ,[23]. As a superset of an object-oriented language, JACK™ brings conservative but versatile agent-oriented extensions at the following levels:

a) Syntactic level referring to new data types for class definition associated to the BDI notions as agent, plan, event, belief-set, capability, view and team; new declarations (statements beginning with #) for the relationships identified between agent entities.

b) Semantic level introducing new statements to operate with the Jack structures (statements beginning with @) that model the BDI reasoning mechanism.

The JACK™ compiler will convert the agent source code in Java statements. The agent shell is formed by a set of classes that enables the management of concurrent tasks and offers an infrastructure for agent message based communication; it allows the existing of multiple agents inside a process with grouping facilities. The agent represents the fundamental entity of the Jack language. It has a rational behaviour, based on the BDI paradigm and released by pro-active and reactive stimuli. An agent contains its own data set representing its beliefs about the environment, the events that it can handle, the goals that it may assume and the plans to achieve each goal. As a new agent is built, it will wait inside the multiagent system until it receives a goal or perceives an event that it must respond to. Then, it identifies the set of applicable plans and selects an appropriate one after a deliberation process. If the execution of the chosen plan fails, the agent will try the next applicable plan
and the process is repeated until no alternate plan can be found. Despite the fact the agent seems to behave like an expert system, this is contradicted by its decision making characteristics [21] [23]: the goal-directed focus, real-time context sensitivity, real-time validation of approach, multi-tasking concurrency adoption. All these properties make it appropriate for the use in HMS. An agent declaration pattern can look like the next Jack code:

```jack
agent AgentType extends Agent [implements Interface1, ...] {
    // the bold characters are Jack keywords
    // declaration of the beliefsets used by the agent
    #private data BeliefType belief_name(args);

    // declaration of the events handled and/or sent
    #handles event EventType;
    #posts event EventType ev_ref;
    #sends event EventType ev_ref;

    // declaration of the plans used by the agent - the order is important!
    #uses plan PlanType;

    // declaration of the agent capabilities
    #has capability CapabilityType cap_ref;

    // other declarations for data and method definitions
}
```

The Event represents the main motivational factor for the agent actions. As already mentioned, the Jack environment can manage both perception (also named external) events and goal (also named internal) events. A possible event definition referring to a raw part localization for an agent included in a robot holon is depicted below:

```jack
event RawPartLocation extends Event {
    // event - Jack keyword, Event - package class for events manipulation
    float x;
    float y;
    float z;
    #posted as location(float xx, float yy, float zz){x=xx; y=yy; z=zz;}
}
```

The Plan represents the agent’s answer procedure to a set of sensed events. During the execution of a multiagent system, Jack associates every event with a certain number of relevant plans. These are filtered against applicable conditions and then passed through a deliberation process in order to obtain the next agent’s intentions. The following source code represents the pattern of a possible plan definition:

```jack
plan CurrentPlan extends Plan{
```
// declaration of the handled event
#handles event SignalON_event ref_ON;

// plan methods definitions – statements starting with @ that define the relationship
// with other Jack entities, data, reasoning statements
static boolean relevant(EventType ev_ref){
    // filtering code to establish the plan relevance to the given event
    return true;
}

context(){
    // logical condition to validate the plan instances that match the current working conditions
    true;}

body(){
    // the plan execution statements
    @wait_for(robot1.planWait());}

The Beliefset represents the declarative tuple-based relational form of an agent’s beliefs. The type of the value field can be a Java primitive type or a user-defined one. A beliefset can be either open world or closed world and the kernel ensures its logical consistency [11]. Furthermore this class offers a lot of useful non-retrieval functions like automatic event generation that allows a plan to change its activities. A generic definition of a beliefset is depicted below:

beliefset CurrentBelief extends CloseWorld{
    /* closed world approach assumes the memorization of only the true relationships; any non found relationship is considered to be false */
    #key field Key_Type key_name;
    #value field Value_Type val_name;
    #indexed query get(Key_Type, key_name, logical Value_Type, val_name);

    CurrentBelief(String data_source_name){
        // the beliefset updates are made inside the constructor method
        add(key_name,val_name);
    }
}

The View acts as a data abstraction mechanism that allows agents to work with heterogeneous data sources using a single interface – the same as a beliefset [11]. In Jack the team appears like an extension of the BDI paradigm to allow social, coordinated activities across the multiagent system. The members of a team have their own beliefs, possible different from the team entity. Jack also offers a communication mechanism to facilitate coordination within the team hierarchy. In order to organize the above mentioned functional components of an agent, Jack language allows the definition of an entity named Capability, as a hierarchical reusable structure. This satisfies a pressing
software engineering requirement, namely the development of re-usable agent-oriented libraries. Referring to the time management, Jack is a neutral language providing three types of time measuring: real-time, dilated and simulation. Any agent has a default timer switched on real-time that can be shared with agents from another computer (real-time) or from a process (dilated, simulation), the user being responsible for its management. Beside the agent language, Jack offers to designers and developers powerful visual tools as JDE (Jack Development Environment) and Jack discrete events simulation (allowing views of activities, processes, perceptions and BDI events and assisting multiagent system management) or languages for inter-process operation (JACOB) or Web development (WebBot).

As already presented, agents have been used to develop the holonic structure according to the PROSA architecture. At present, the resource holons were implemented in Jack, taking into account the specific features of the available experimental manufacturing system. An interesting case was the one of the robot that can exchange information with the computer vision system. In the HMS, there is a holon dedicated to the robot, and another one for the vision part as a sub-resource. From the service operation point of view the robot holon may request services that can be obtained from the computer vision holon (e.g. part detection and location, part identification, quality control). From a Jack agent design and implementation experience, one may notice two main points: one about the hierarchy of capabilities with their associated plans and internal messages and another on the agent’s communication with other external entities.

For illustration, a robot agent was developed with three capabilities, as shown in Fig. 7: processing, interaction and sub-resources planning. The interaction capability allows the agent to treat the received messages and to generate events as response. As the entire communication mechanism is oriented towards services, the requests are generically named “robot-service-request” and the answers “robot-service-discharged” for a goal achievement, respectively “robot-service-not-discharged” for operation failure. The “robot-state” is the response that contains the actual state variables and does not require a plan to establish them. The availability is measured by the response “robot-availability” and it requests a plan to test the driver application in order to find whether the robot is free or not. The processing plans as part of the Processing capability are strongly connected with the sub-resources planning activities.

Thus, at an assembling or part transfer request, the robot agent (as the decisional part of the robot holon) will demand the support of the vision system through the internally generated event “sub-resource-request”. The plan “sub-resource-planning” becomes active, uses its beliefset with the sub-resource task priorities and generates new events of request-service type. No matter what plan was activated, if it was completely executed a success message is posted; otherwise, a failing plan becomes active and posts specific messages. The
failure event produces new service-requests until a plan will be successful or a specific "service-not-discharged" message will be sent to the manager holon/agent.

Fig. 7 – Capability view of the robot agent.

As about the agents’ communication mechanism, Jack inter-agent communication diagrams are illustrative. For the robot agent of the proposed HMS the corresponding diagram is depicted in Fig. 8. According to the adopted architecture, the order holon/agent can make requests on processing tasks or
state information that the robot holon/agent should be able to respond. It can be
noticed that the plan “working-task-coordination” is responsible for “robot-
service-request” event generation. When the robot agent receives such messages
it activates appropriate service discharge plans.

Fig. 8 – The robot agent communication diagram.

An advantage of the Jack implementation refers to its automatic
generation according to the visual design. Only the specific actions’ code needs
to be directly introduced by the programmer. The source is identical with a
classical Java program: a main class and a number of supporting classes. From
the main class the agent instances are created, a minimal number of events for
simulation purposes are generated and the first GUI appearance is set-up. The
other classes should contain the definitions of Jack multiagent system entities
and Java supporting classes. For example, the Agent entity for a robot resource
holon of the proposed HMS is defined, according to the design of Fig. 7, by the
next statements:

```java
agent Robot extends Agent{
#handles event Robot_service_request;
#handles event Robot_state_request;
#handles event Resource_availability_request;
#posts event Resource_availability_request;
```
#posts event Working_task_achieved;
#posts event Working_task_request;
#sends event Subresource_service_request;
#sends event Robot_service_request_discharged;
#sends event Robot_service_request_failed;
#sends event Robot_state;
#sends event Robot_availability;
#uses plan Transfer_plan;
#uses plan Assembling_plan;
#uses plan Sub_resource_plan;
#uses plan Robot_service_discharging_plan;
#uses plan Robot_achieved_task_plan;
#uses plan Robot_failed_task_plan;
#uses plan Robot_state_plan;
#uses plan Resource_availability_plan;
#has capability Robot_processing;
#has capability Robot_interaction;
#has capability Robot_planning;

//other data declarations and method definitions used by the robot class
}

The unspecified declarations of data and methods from the robot class will facilitate Java/Jack object instantiations, the end of agent execution, message posting/sending as response to plan execution and/or external demands, timer specification, agent name identification inside the multiagent network and so on. Furthermore, to synchronize the different physical entities from the HMS, for example when the multiagent system must handle the concurrent agents’ access to certain resources, the use of a Jack semaphore mechanism is possible. An \texttt{ON} value assigned to a semaphore will keep the agent blocked until a certain event is generated. This is implemented by an \texttt{autorun()}} method inside the appropriate capability class definition:

    capability Processing extends Capability{
....
    protected void autorun(){ Robot.process_OK_signal();}
}

The code presented in this section has to be completed according to the specific operation conditions and the considered communication interfaces, as previously explained. The overall multiagent system will be finally executed by the invocation of the Java Virtual Machine, as any classical java source.

6. Conclusions

1. An important point of the presented research refers the obtained
characteristics of the manufacturing control system. The considered semi-heterarchic architecture, with the holons belonging to several holarchies that can be temporarily formed, conducts to a systematic method for the analysis, design and implementation of the manufacturing control systems. The respective structure gains in adaptability and reliability. These features that already exist in agent based architectures are enhanced in the HMS. In the proposed holonic solution a few types of holons can be dynamically organized using a service oriented architecture. While agent based programming is presently already considered [4], [23], one can think about a holon based design. Indeed, the homogeneity obtained for all the phases, from analysis to implementation, is determined by the holons being the only building blocks, and all having the same structure: a decisional part materialized as an agent and a structural part. This last component is supposed to offer certain services, and it can be recursively obtained as a holon. The links between the holons that are created according to a contracting mechanism, taking into account the available services, represent contextual relations without any rigid, a priori settled parameter.

2. The multiagent systems are generally considered as tools for the HMS. The present research proved the power and flexibility of Jack environment for the development of agent-oriented applications to be integrated in manufacturing systems. It augments the existing and popular Java language with new constructs for agent definition/communication, goal-based/reactive programming and knowledge representation. The programmers that are familiar with Java will be able to make a rapid integration of the new agent tools. The Jack orientation toward the BDI paradigm does not become dominant since Jack component-based architecture still supports different agent-programming styles. The JDE represents a strong tool that simplifies the knowledge acquisition phase, allowing the experts’ direct involvement in the software engineering process for all the development phases.

3. The holons suppose a closed-loop operation that admits failures for the plans chosen for execution and the necessity of recovering based on monitoring, diagnosis and re-planning operations. About this, the planning method and the implementation solution have to support the respective procedure. From this standpoint the Jack solution introduces specific constraints on the planning phase, which is solved by the means of certain plan patterns that are endowing the agents. This leaves some open research issues about the manner in which the Jack BDI deliberative process should be adapted to entirely sustain the holons’ functionality.

Acknowledgements. This work was supported by the research project SOFHICOR, Contract no. 11-042/2007.

Received: November 14, 2008

"Gheorghe Asachi” Technical University of Iași, Department of Automatic Control and Applied e-mail: dorup@ac.tuiasi.ro
REFERENCES


UNELE ASPECTE ALE PROIECTĂRII ȘI IMPLEMENTĂRII SISTEMELOR HOLONICE DE FABRICAȚIE

(Rezumat)

Sunt prezentate rezultate ale unei cercetări aflate în desfășurare privind sistemele holonice de fabricație. Domeniul respectiv cuprinde o serie de metode și tehnici având ca scop îmbunătățirea performanțelor sistemelor de control folosite în fabricație. S-a plecat de la un model holonic de referință și anume arhitectura PROSA. Contribuțiile aduse privesc rezolvarea coordonării și unele aspecte ale implementării sistemelor holonice de fabricație. Este vorba de noi posibilități de control pentru sistemele cu o structură distribuită, folosind mecanisme adecvate de comunicație. S-a ales pentru construirea sistemului software un mediu de programare orientat agent, fiind ilustrate avantajele și unele limitări pe care acesta le aduce.