Optimization of Tasks Scheduling in Cooperative Robotics Manufacturing via Johnson's Algorithm

Case-Study: One Collaborative Robot in Cooperation with Two Workers

Ahmed R. Sadik
Fraunhofer Institute for Computer Graphics and Research
University of Rostock
Rostock, Germany
ahmed.sadik@igd-r.fraunhofer.de

Andrei Taramov
Fraunhofer Institute for Computer Graphics and Research
University of Rostock
Rostock, Germany
andrei.taramov@uni-rostock.de

Bodo Urban
Fraunhofer Institute for Computer Graphics and Research
University of Rostock
Rostock, Germany
bodo.urban@igd-r.fraunhofer.de

***Abstract*— The rapid development of information technology in the last half century led to the emergence of a new industrial revolution, often called Industry 4.0, the key element of which is the introduction of informatization in all spheres of human life and the widespread use of cyberphysical systems. The main attribute of such systems is the interaction of human and smart machines, this approach allows achieving the greatest flexibility and productivity simultaneously. The latest example of such systems is the collaborative manufacturing system, where the human worker cooperates in a close distance with a collaborative robot (cobot) in a production scenario. This cooperation is applicable when the final product requires a high degree of customization that a worker can provide, while cooperation with the cobot is greatly speeding up the productivity. In this context, one of the actual problems is to schedule the cooperative tasks in real time among the operational resources (i.e., the workers and cobots). This problem can be reduced to a special case of the flow-shop scheduling problem. The complexity of this problem increases with increasing the number of cooperative operational resources and the production steps. Undoubtedly, modern production often involves several processing steps serviced by several operational resources. Therefore, it is necessary to study a complex cooperative manufacturing scenario. The simplest and most understandable case scenario is the interaction of two workers and one cobot in two stages production workcell. Thus, in this paper we will consider the implementation of this case-study using the available scheduling algorithms.**

Keywords—Flow Shop Scheduling; Johnson's Rule; Collaborative Robotics; Cyber-physical System; Holonic Control Solution; Multi-agent System; Rule Management System.

I. INTRODUCTION

The combination of a human worker and the cobot within a single production workcell represents a new manufacturing approach that provides both flexibility and agility in modern production [1]. Involvement of a human in production allows to meet the requirements of manufacturing products with a high degree of customization, and also provides a quick and resource-efficient response to the continuous changes in production requirements, thus ensuring maximum flexibility of production and adaptability to external conditions. The cobot, on the other hand, can perform simple, often repetitive operations in a much faster and more efficient manner [2].

The implementation of this technology promises to bring a lot of advantages over classical production techniques, but also proposed new problems [3]. One of such problems is planning of the interaction between the worker and the cobot, which is not an easy task, because the nature of their functioning is very different from each other. Therefore, for successful implementation of this type of system, it is necessary to separate the tasks into production stages, distribute them between the worker and the cobot and create an algorithm for their interaction. The most common example of such interaction is the cooperative assembly. In this process, the cobot can be assigned to the pick and place function and/or primary item processing, after which the product part is delivered to the worker, where further assembly and customization is carried out. This approach corresponds to the scenario described above, clearly divides the production into stages attached to the worker or the cobot [4].

However, distributing the tasks among the cooperative resources in advance does not guarantee the optimal performance of the entire system, in the absence of an algorithm that optimize the scheduling of these cooperative tasks. The lack of an optimum scheduling algorithm may result the appearance of so-called bottlenecks and, as a consequence, a general decrease in productivity. The mathematical aspect of this problem has long been known in the scientific community and is called the flow-shop problem. Thus, in the present paper, a possible solution to the described problems will be presented on a concrete case-study.

The paper is structured as follows. Section II describes the problem at the hand, theoretical solution of which is presented in Section III. Section IV presents software and architectural basis which are needed for implementation. Section V contains an implementation of the proposed solution. Section VI summarizes the results of the current study.

978-1-5386-0386-4/17/$31.00 ©2017 IEEE

II. The Problem in Details and a Case-study

As mentioned above, the main problem in planning the interaction between cooperative resources can be incorrect assignments to certain cooperative task or job in the wrong order. It is easy to demonstrate on a concrete example, the simplest is two-stage production, where the cobot performs the initial processing of the product or just pick and place operation, and then the results of its work are sent to further post-processing by the worker. At the worker stage, there are two workers who work in parallel. The potential problem in this case lies in the fact that the processing speed of these modules may be uneven, which can lead to situations when the job is idle, waiting for execution or when the performer is idle, waiting for the job. The first situation occurs every time the first production stage is running much faster than the second one, releasing more jobs than the second can handle, which causes the job to be idle, waiting for further processing. The second situation occurs on the contrary, when the first stage does not work fast enough to supply work to the second stage, thereby forcing the latter to stand idle while waiting for job. Furthermore, we should put into consideration as well the fact that in the second stage the two parallel workers have different assembly speeds, which creates additional threats of non-optimal use due to incorrect distribution of work flows among these workers.

The research case-study considers all the conditions described above. The optimization of the current makespan, i.e. the total processing time of the present jobs queue, is the main problem the paper is answering. Implementation of this idea, however, is much more complicated than its description, since it requires a multistage approach to plan, as well as to monitor the speed of each worker to predict its workload.

III. Theoretical Basis of the Solution.

A. *Two-stage scheduling Problem. Introduction to the Johnson's method*

Makespan optimization of a sequential two-stage production is a typical flow-shop problem, which can be effectively solved by a simple and cost-effective approach called Johnson's method [5]. This method, when applied under the proper conditions, allows to minimize occurrence of both situations of delay in work described in the previous section, namely the worker is waiting for the job and the job is waiting for the worker [6]. In order to understand the principle of the Johnson's method, it is necessary to identify the causes and possible variants of the occurrence of delays. To demonstrate these variants, we introduce several terms - unscheduled job (J_i), production stage 1 (PS_1) and production stage 2 (PS_2). For each job, there is an execution time on PS_1 ($T_{Ji}@PS_1$) and PS_2 ($T_{Ji}@PS_2$).

Johnson's method assumes that there is a storage buffer between the stages to accumulate the results of the cobot before passing through PS_2, thus allowing the production to continue at PS_1 without waiting for the completion of the job at PS_2. This circumstance makes the order of job execution at PS_1 indifferent, since the makespan for this stage will always remain constant. However, the overall makespan is calculated by the completion time of the stage with the maximum duration. In this case, it will always be stage two, because it cannot be ahead of stage one. Thus, the optimal makespan at PS_1 does not guarantee the maximization of overall productivity due to the presence of a second stage that can be idle in case of insufficiently rapid provision of material from PS_1, thereby increasing the total production time. This makes PS_2 sensitive to the order of job scheduling. So, if jobs, which require more time in PS_1 than in PS_2 ($T_{Ji}@PS_1 > T_{Ji}@PS_2$) are scheduled at the beginning of the queue, this will cause a total performance penalty of

$$\sum_{i=0}^{n} (T_{Ji}@PS_1 - T_{Ji}@PS_2) \quad (1)$$

Where n is the number of scheduled jobs with $T_{Ji}@PS_1 > T_{Ji}@PS_2$. On the other hand, if we put jobs with $T_{Ji}@PS_1 < T_{Ji}@PS_2$ in the beginning of the queue, then this will gradually fill the buffer between the stages by an amount of

$$\sum_{i=0}^{n} (T_{Ji}@PS_2 - T_{Ji}@PS_1) \quad (2)$$

Where n is the number of scheduled jobs with $T_{Ji}@PS_1 < T_{Ji}@PS_2$. The amount of work accumulated in the buffer compensates for subsequent delays in the case of occurrence of a job with $T_{Ji}@PS_1 > T_{Ji}@PS_2$, since the storage buffer represents the finished jobs to be processed by PS_2, thus preventing its idle time. Thereby scheduling algorithms should always strive to maximize the buffer before the emergence of the job consumes it.

Also, the problem of the first stage should be considered. Otherwise, the buffer will always be empty and the delay in PS_2 will increase, therefore this delay should be minimized by choosing the job with the minimum T, however, a condition such that $T_J@PS_1 < T_J@PS_2$ should be observed, otherwise an additional delay will occur.

Considering all of the above, the following formula for calculating the total production makespan in the case of optimal planning can be derived:

$$\sum_{i=0}^{n} T_{Ji}@PS_2 + T_{J0}@PS_1 + \max\left(\sum_{i=1}^{n} (T_{Ji}@PS_2 - T_{Ji}@PS_1); 0\right) \quad (3)$$

B. *The use and evaluation of the Johnson's method*

Input data of the method are presented in the form of an initial list of jobs (L), which is an unsorted table, where the execution time at each PS can be determined. At the output, the algorithm should produce a list of scheduled jobs (L′), with a strictly defined order, where the first records mean early execution, and the latter means later execution.

The flow chart of the Johnson's algorithm is shown on Figure 1. The first step is to find the smallest time value from all the input data in the initial list (L), regardless of the column. If the minimum time found is related to the PS_1, then the corresponding work is placed on the first free space in the scheduling list (L′), otherwise it is placed on the last place of the (L′). Then the record is deleted from the initial list (L) and the process is repeated until there are no jobs left in the initial list (L).

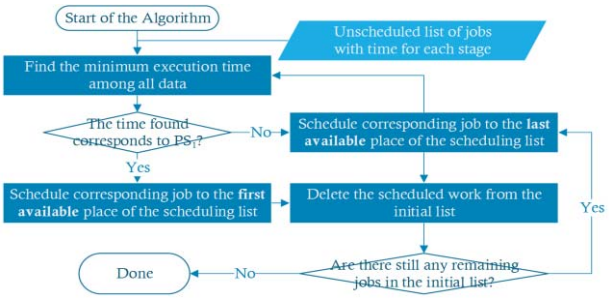

Fig. 1. Johnson's rule algorithm.

As a result of the algorithm, all the most rapidly executed jobs over PS_1 which the condition $T_{Ji}@PS_1 < T_{Ji}@PS_2$ is satisfied are placed on top of the list L′ and, therefore, they are executed first. While all the fastest jobs over PS_2 which the condition $T_{Ji}@PS_1 > T_{Ji}@PS_2$ is satisfied, moved to the end of the list L′ and are executed last. This will ensure, first, the minimum initial waiting time for the PS_2 due to the smallest possible execution time of the first task on PS_1, and secondly it produces the fastest filling of the buffer due to the difference in execution time ($T_{Ji}@PS_2$ - $T_{Ji}@PS_1$) in the initial jobs.

C. Distribution of Jobs Among the Workers

The previously described Johnson's method, as it was explained, completely solves the scheduling problem over two-stage production. However, the case-study under consideration, namely the production on the basis of one cobot and two workers is not completely reduced in the same manner. In our case-study, we attempt to abstract and divide the current task into two different subtasks, namely, the task of a two-stage flow-shop and the task of distributing workers within a single production stage. To solve the first problem, we presented two existing workers as a single unit within the black box, which made it possible to reduce the task to the required level of complexity for the application of the corresponding algorithms. The next step is to solve the second problem, considering what happens in the black box with two workers. Obviously, the assignment for the job in the queue depends primarily on the current status of workers. In the case of two workers, the following options are possible: both workers are free, then select a worker who has done less to the moment. Worker-1 is busy and Worker-2 is free or vice versa, then a free worker is selected. Both workers are busy, then wait till one is free.

PS_2 presents the workers stage, which is a dependent block with a buffer. This means that the work on this stage is performed asynchronously with the PS_1 and the jobs can be assigned regardless of whether PS_2 is occupied or free. The stage conditions are also applicable to the workers, i.e assignment is possible even when both of them are busy, which requires integrated approach to assess the current workload of a worker based on the expected amount of work and the worker's performance. Also, since the performance of the workers is measured, it is possible to choose the most productive worker instead of choosing worker who has done less.

Measuring the productivity of the worker, makes it possible to roughly estimate the total amount of time that will be taken to process the entire jobs list [7], in order to figure out which of them will end earlier. However, given the likely emergence of "borderline" results, which are characterized by a small-time difference between the first worker and the second, the projected time for performing the current work should also be added to the total time. A comparison of the final processing time results will give the final solution. The calculation of the processing time is shown below on the Gantt chart in Figure 2. In the case presented in the chart, the job will be given to the worker-2, since his overall predicted time is shorter than that of the worker-1.

Fig. 2. Workers Gantt chart.

IV. Software and Architectural Basis for Implementation.

Holonic Control Architecture (HCA) is a distributed control and communication topology, which is used to solve a wide range of problems, including the flexible and agile system problem [8]. HCA implies the existence of special autonomous cooperative blocks as its constituent elements, they are called holons. Holon can transform, transport, store and validate information and physical objects via two defined components which are an interface and communication components.

HCA allows the use of diverse types of holons, distributing production processes and responsibilities. Product-Resource-Order-Staff-Architecture (PROSA) is the most common HCA model. The PROSA basic holons are: Product Holon (PH), which is responsible for processing and storing the different production plans required to insure the correct manufacturing of a certain product; Order Holon (OH) is responsible for composing, managing the production orders, it also can assign the tasks to the existing operating resources and monitor the execution status of the assigned tasks; Operational Resource Holon (ORH), which is a physical entity within the manufacturing system, it can represent a robot, machine, worker, etc.

The most common technology to apply the HCA concepts is the autonomous artificial agent. Conceptually, an agent is a computing machine which is given a specific problem to solve [9]. It chooses certain set of actions and formulates the proper plans to accomplish the assigned task. The set of actions which are available to be performed by the agent are called a behavior. The agent behaviors are mainly created by the agent programmer. An agent can execute one or more behavior to reach its target. The selection of an execution behavior among others would be based on a certain criterion which has been defined by the agent programmer.

A Multi-Agent System (MAS) is a collective system composed of a group of artificial agents, teaming together in a flexible distributed topology, to solve a problem beyond the capabilities of a single agent. JAVA Agent Development

Environment (JADE) is a distributed MAS middleware framework. Each JADE instance is an independent thread which contains a set of containers. A container is a group of agents run under the same JADE runtime instance. Every platform must contain a main container. A main container contains two necessary agents which are: an Agent Management System (AMS) and a Directory Facilitator (DF). AMS provides a unique ID for every agent under its platform to be used as an agent communication address. While the DF announces the services which agents can offer under its platform, to facilitate the agent services exchange, so that every agent can obtain its specific goal [10]. JADE applies the reactive agent architecture which complies with the Foundation for Intelligent Physical Agent (FIPA) specifications, and provides a graphical interface to deploy and debug a MAS. FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies. JADE agents use FIPA-Agent Communication Language (FIPA-ACL) to exchange messages either inside its own platform or with another platform in a distributed MAS.

V. Implementation of the Proposed Solution

A. *Johnson's scheduling and worker distribution implementation on Detailed Case-Study*

The Case-Study is engaging the production of continuously incoming jobs or orders. Each order at a minimum contains information about the configuration of the required products and the number of these products. A customizable pump was chosen as an example of a customized order. Each job order must be processed first by the cobot, and then by one of the workers. Since the cobot works with relatively constant speed, it can be assumed that the processing time of each pump by the cobot is constant. Thus, the total processing time of the job can be calculated by multiplying the robot time constant (RTC) parameter by the number of pumps (n_i) in the order as it is been explained in the figure below.

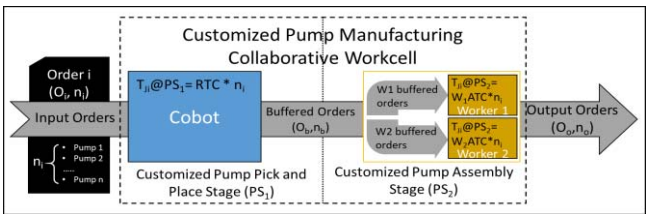

Fig.3. Case-study manufacturing workcell.

After the cobot has processed the product, the product enters the buffer on the secondary processing queue at PS1. PS1, in its turn, consists of two workers whose productivity are not constant, but it can be predicted by analyzing the previous production history. One of the simplest and effective method is averaging. Thus, the time spent on processing an order by one worker is measured by multiplying the worker's average time consumption (W_iATC) by the number of items in the order (n_i). Each worker has his own schedule and his own buffer, orders are distributed between them, and thus the total PS2 time will equal the maximum of the two processing times as shown in formula 4.

$$T_{\text{total}}@\text{PS2} = MAX\left(\sum_{i=0}^{n} T_{Ji}@W_1, \sum_{i=0}^{n} T_{Ji}@W_2\right) \qquad (4)$$

This makes the previously chosen distribution algorithm relevant, because it is aims to balance the load between workers. The figure below describes the implementation algorithm of Johnson's rule and worker distribution.

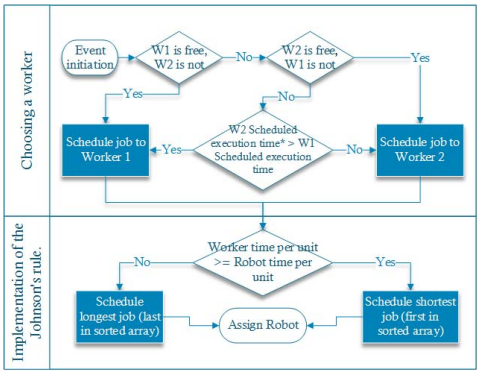

Fig. 4. Worker and robot job scheduling.

The workers's processing time is calculated according to the algorithm described in the third section - W_iATC is multiplied by the sum of all n_i of all the scheduled orders. After selecting the least loaded worker, a suitable job is selected using the Johnson's rule, which implementation is simplified, because the $T_{Ji}@PS_1$ and $T_{Ji}@PS_2$ values are correlated through n_i, so the smallest values will always be in the stage with the smallest T_{Ji}. Thus, if $T_{Ji}@PS_2 < T_{Ji}@PS_1$, then all work by the Johnson's rule should be sorted in n_i decreasing order, i.e. in the array of orders sorted by n_i, the order with the largest n_i will be scheduled first. In the case if $T_{Ji}@PS_1 < T_{Ji}@PS_2$, everything will be exactly the opposite and the first order with lowest n_i should be scheduled.

B. *First Come First Serve Scheduling*

As noted earlier, W_iATC is needed to evaluate worker productivity. This productivity can of course be measured empirically, but this requires time and several production cycles performed without using this data (and therefore without planning). In other words, we are faced with the problem of a cold start. This is a fairly well-known problem, especially in the field of recommending systems. It can be solved in a number of ways, for example, substitution of default data, use of zero values, etc. However, all of the above measures are inaccurate and do not allow to correctly sort the jobs for scheduling. Therefore, we made the decision to use a simple planning algorithm that does not take into account the worker's performance, until the accumulation of enough data to use in the Johnson method. For this case, the usual First Come First Serve (FCFS) scheduling method is used, which adds job sequences in the order in which they were batched.

FCFS is applied at the level of planning the interaction of different stages of production, but the distribution of jobs must also be performed between the workers of one stage, which complicates the task. To solve it, the multi-level logic of revealing an attached worker is used, based on several parameters: Worker-1 and Worker-2 state – is worker busy or

free; Worker-1 and Worker-2 done counter – the number of orders successfully processed by this worker; Change Flag - A special switch that allows to alternate workers in a situation when all other parameters are equal. Below is a truth table showing the action taken, depending on these parameters (Table 1):

Table 1. First Come First Serve Worker Scheduling.

Case	State					Action
	Worker 1 busy	*Worker 2 busy*	*W1 done counter > W2 done counter*	*W2 done counter > W1 done counter*	*Worker Change Flag*	
Init. state	False	False	False	False	False	-
1	False	True	Not matter	Not matter	Not matter	Worker 1 scheduled
2	True	True	False	True	Not matter	
3	False	False	False	True	Not matter	
4	True	True	False	False	False	Flag change
5	False	False	False	False	False	
6	True	False	Not matter	Not matter	Not matter	Worker 2 scheduled
7	True	True	True	False	Not matter	
8	False	False	True	False	Not matter	
9	True	True	False	False	True	Flag change
10	False	False	False	False	True	

C. *Measuring the worker's time and the conditions for choosing an algorithm*

Table 2. Dools Assigments.

Event	State Rule					Action	State Explanation
	Done by Workers Counter	Robot Status	Worker Status	Received Orders List	Scheduled Orders List		
New Order Event	At least one is equal 0	Free	Don't Care	Don't Care	Don't Care	• FCFS scheduling • Robot order assignment	• New Order event is received at the very beginning. • New Order event is received after the robot is done with all the previously assigned order, while the worker has not finished the first assigned order.
New Order Event	At least one is equal 0	Busy	Don't Care	Don't Care	Don't Care	• FCFS scheduling	• New Order event is received while the robot is executing one previously assigned order, while the worker has not finished the first assigned order.
New Order Event	Both greater than 0	Free	Don't Care	Don't Care	Don't Care	• Johnson scheduling • Robot order assignment	• New Order event is received after the robot is done with all the previously assigned order, while the worker has finished at least the first assigned order.
New Order Event	Both greater than 0	Busy	Don't Care	Don't Care	Don't Care	• Johnson scheduling	• New Order event is received while the robot is executing one previously assigned order, while the worker has finished at least the first assigned order.
Robot Done Event	Equal 0	Don't Care	At least one is free	Not Empty	Don't Care	• FCFS scheduling • Worker order assignment	• The robot finished one assigned order while the worker is free and has not assigned any previous order.
Robot Done Event	Equal 0	Don't Care	Both busy	Not Empty	Don't Care	• FCFS scheduling	• The robot finished one assigned order while the workers are executing the first assigned order.
Robot Done Event	Both greater than 0	Don't Care	At least one is free	Not Empty	Don't Care	• Johnson scheduling • Worker order assignment	• The robot finished one assigned order while the worker is free and has finished at least one assigned order.
Robot Done Event	Both greater than 0	Don't Care	Both busy	Not Empty	Don't Care	• Johnson scheduling	• The robot finished one assigned order while the workers are busy and have both finished at least one assigned order.
Worker Done	Don't Care	Don't Care	Don't Care	Don't Care	Not Empty	• Calculate the order execution average time • Worker Task Assignment	• The worker finished one assigned order while still there is at least one order in the scheduled orders list.
Worker Done	Don't Care	Don't Care	Don't Care	Don't Care	Empty	• Calculate the order execution average time	• The worker finished one assigned order while the scheduled orders list is empty.

Based on the foregoing, the final implementation of the system should be based on two planning principles - FCFS for the accumulation of initial data and the Johnson method at the remaining production time. So, it is necessary to create a switching mechanism that determines the condition for changing ordering method. For these purposes, a rule management system (RMS) Drools is used [11]. It is a form of an expert system which usually uses an ontology based representation to codify the input data into a knowledge base which can be used for reasoning [12]. An RMS depends on two kinds of memory. The working memory holds the facts which present the domain knowledge, while the production memory holds a set of rules which are usually presented in form of conditional statements [13]. The reasoning engine is a problem solver which solves a given problem by matching the present facts with the existing rules [14]. Table 2 describes the rules for starting an algorithm as well as other start/attachment events.

The calculations of the worker's time can also be seen from the table. These actions are necessary to measure W_iATC. After assigning a job for the worker, a software timer is turned on and it stops after receiving an ACL message from the worker which informs the completion of job. Time difference is divided by the n_i, after this value is summed with the sum of similar values obtained before, and then divided in half, the result is W_iATC. Below is the calculating formula:

$$W_{i+1}ATC = \frac{W_iATC+(t_{end}-t_{start})/n_i}{2} \quad (5)$$

D. *Final HCA Implementation*

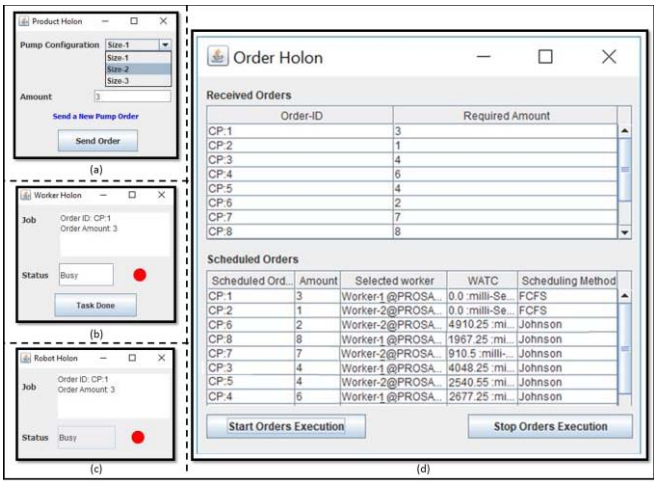

Fig. 5. HCA GUI.

Figure 5 shows the Graphical User Interface (GUI) of the implemented case-study. While Figure 6 shows the interaction and the behaviors model of those holons. The first holon can be seen in the implementation is the PH with a GUI shown in Figure 5-a. The PH is generating the pump orders by choosing the pump size and amount. By pressing send order button, the product agent which is shown in Figure 6-a triggers a one shot behavior which to send the pump order via an ACL-message to the OH. An example of the agent communication via the ACL-messaging can be seen in Figure 6-b. This ACL-message contains the pump order details and it has an AGREE communication act. The OH GUI is shown in Figure 5-d. The order agent implements a cyclic behavior which continuously receives the pump orders from the pump agent.

A one shot behavior is triggered by Drools to send a REQUST-message to the robot agent, which has a cyclic behavior to continuously receive the order assignments. When the robot agent receives the REQUST-message, it assigns the pump order to the cobot via the Robot Holon (RH) GUI as it

can be seen in Figure 5-c. Also, the robot agent starts the RTC timer, in our case study we dedicated a 2.0 seconds value for the RTC timer. When the RTC timer value is elapsed, the robot agent sends an INFORM-message to the order agent to inform it has finished the assigned task. When the order agent receives a robot done INFORM-message, it fires a new Drools reasoning session to schedule the existing standing pump orders due to the decision table rules shown before and the status of the two workers. Therefore, Drools assigns a task for the worker by triggers a one shot behavior to send a REQUST-WHEN-message to the Worker Holon (WH) GUI as it can be seen in Figure 5-b. When the worker finishes assembling the pump orders, he presses task done button via his GUI. By pressing worker task done, the worker agent triggers a one shot behavior. When the order agent receives a worker done event, it fires a new Drools session. Drools will calculate the W_iATC every time the order holon receives a worker done event. The rest of the scheduling will be based on the current W_iATC.

The OH-GUI which can be seen in Figure 5-d is showing the scheduling results. The first two pump orders (CP:1, CP:2) were scheduled as FCFS and assigned to worker-1 and worker-2 respectively. This is because neither worker-1 nor worker-2 were done by assembling any previous orders until that instance. When worker-2 has finished CP:2, W_2ATC could be calculated as 4.91 seconds which is greater than the value of RTC (i.e., 2.0 seconds). Thus, Johnson algorithm could be applied via using the previously mentioned formulas, therefore CP:6 has been assigned to worker-2. When worker-1 has finished CP:1, W_1ATC could be calculated as 1.96 seconds which is less than the value of RTC (i.e., 2.0 seconds), therefore CP:8 has been assigned to worker-1 via Johnson algorithm. Furthermore, the rest of the orders at Figure 5-d are scheduled in a similar manner.

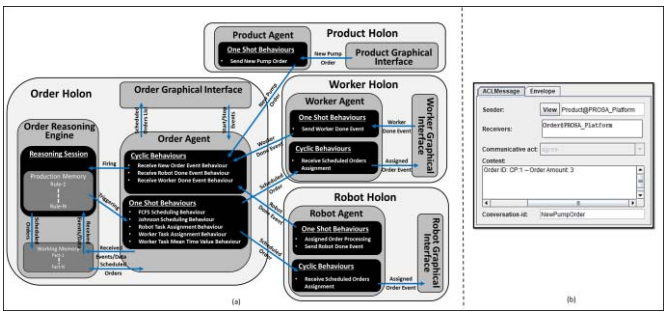

Fig. 6. Interaction of Holons.

VI. Conclusion

As a result of this work, a method was found for planning the interaction between the cobot and two workers and scheduling the jobs among them within the same production cell. The task under consideration was divided into two steps - the distribution of work between the production stages and the distribution of work within the production stage. For the first case, Johnston's method was used, which proved to be very useful in solving such problems. While the distribution of work among workers was based on a more complex pattern of predictions using worker average time (W_iATC). The solution was successfully implemented on the basis of multi-agent HCA architecture, the algorithmic basis of the solution was developed as well as the graphic shell, which allows to use the existing solution at the program level in real conditions.

During the case-study implementation it has been noted that when the value of W_iATC is greater than RTC, Johnson algorithm tends to schedule the jobs based on ascending the number of required units. On the contrary, Johnson algorithm tends to schedule the jobs based on descending the number of required units if the value of W_iATC is greater than RTC.

Nevertheless, the topic under consideration has not been exhausted, and further promising topics are the increase in the number of agents in the production cell, as well as the improvement of algorithms for predicting employee productivity and the introduction of a system for calculating its productivity factor.

References

[1] Hoda A. Elmaraghy, "Flexible and reconfigurable manufacturing systems paradigms", International Journal of Flexible Manufacturing Systems, vol. 17, Issue 4, pp 261-276, October 2005.

[2] A.R. Sadik, B. Urban, "A Holonic Control System Design for a Human & Industrial Robot Cooperative Workcell," IEEE-ICARSC, pp. 118-123, 2006.

[3] G. Michalos, S. Makris, J. Spiliotopoulos, I. Misios, P. Tsarouchi, G. Chryssolouris, "ROBO-PARTNER: Seamless Human-Robot Cooperation for Intelligent, Flexible and Safe Operations in the Assembly Factories of the Future", *Procedia CIRP*, vol. 23, pp. 71-76, 2014.

[4] A.R. Sadik, B. Urban, "Combining Adaptive Holonic Control and ISA-95 Architectures to Self-Organize the Interaction in a Worker-Industrial Robot Cooperative Workcell," Future Internet, vol. 9, Issue 35, 2017.

[5] W. Grzechca, "Manufacturing in Flow Shop and Assembly Line Structure", International Journal of Materials, Mechanics and Manufacturing, vol. 4, no. 1, pp. 25-30, 2015.

[6] W. Stevenson, P. Ness, "Study Guide for Use with Production/Operations Management," McGraw-Hill: USA, 1999.

[7] S. Pellegrinelli, F. Moro, N. Pedrocchi, L. Molinari Tosatti, T. Tolio "A probabilistic approach to workspace sharing for human–robot cooperation in assembly tasks," CIRP Ann.-Manuf. Technol., vol. 65, pp. 57–60, 2016

[8] R. Babiceanu and F. Chen, "Development and Applications of Holonic Manufacturing Systems: A Survey", Journal of Intelligent Manufacturing, vol. 17, Issue 1, pp. 111-131, 2006.

[9] W. Shen, Q. Hao, H. Yoon and D. Norrie, "Applications of agent-based systems in intelligent manufacturing: An updated review," Advanced Engineering Informatics, vol. 20, no. 4, pp. 415-431, 2006.

[10] W. Teahan, Artificial Intelligence – Agent Behaviour, 1st ed. BookBoon, 2010.

[11] A. Ordóñez, L. Eraso, H. Ordóñez, L. Merchan, "Comparing Drools and Ontology Reasoning Approaches for Automated Monitoring in Telecommunication Processes," Procedia Computer Science, vol. 95, pp. 353-360, 2016.

[12] A. Ordóñez, L. Eraso, H. Ordóñez, L. Merchan, "Comparing Drools and Ontology Reasoning Approaches for Automated Monitoring in Telecommunication Processes," Procedia Computer Science, vol. 95, pp. 353-360, 2016.

[13] "Drools Expert User Guide", Docs.jboss.org, 2017. [Online]. Available: https://docs.jboss.org/drools/release/5.2.0.CR1/drools-expert-docs/html_single/ .

[14] N. Kapoor and N. Bahl, "Comparative Study Of Forward And Backward Chaining In Artificial Intelligence", International Journal Of Engineering And Computer Science, 2016.