Agent-based Decentral Production Planning and Control: A New Approach for Multi-resource Scheduling

Martin Krockert, Marvin Matthes and Torsten Munkelt
Faculty of Computer Science, Dresden University of Applied Sciences, Friedrich-List-Platz 1, Dresden, Germany

Keywords: Multi-resource, Agent-based, Decentral, Job-Shop, Production, Scheduling, Planning.

Abstract: Manufacturing jobs commonly require more than one resource in order to equip machines with tools and process jobs. To achieve a feasible production plan and control its execution in an agent-based decentral production we developed a new approach presented in this paper. We introduce a negotiation procedure, based on the job priority and overlapping time slots across all resources. In addition, we provide simulative evidence that our approach is superior, in terms of time-based key performance indicators, to commonly used queuing procedures and the approach provides a more stable production under uncertain customer order arrivals and deviating processing times.

1 INTRODUCTION

Looking at today's productions, it is clear that due to the increased demand for a higher variety of products and a shorter takt time, the production characteristics of flexibility, robustness and scalability becomes more and more important. Nevertheless, while creating more flexibility, constraints of the production must be considered to obtain a feasible production plan. In the field of flexible production planning researchers have already proved to find suitable production plans by using decentralization and aspects of self-organization. (Rohloff, 1993; Gehlhoff and Fay, 2020) Moreover, decentralized production planning and control (dPPC) is able to outperform centrally planned and controlled production especially if uncertain processing times are included (Munkelt and Krockert, 2018).

However, previous publications mostly assume that only one machine is needed for one job. In reality, machines may have to be equipped with tools by experts and monitored or operated by workers during the production process (Miao and Zou, 2015). In many publications, material dependencies are also not taken into account to gain a feasible production plan.

In this paper, we introduce a new multi-resource scheduling approach for agent-based decentralized production planning and control, by handling time slots from multiple resources, which can represent employees, tools or machines. For this purpose we extended the protocol (FIPA TC C, 2002; Guizzi et al., 2019) to become interruptible by any participant upon disruptive events like longer processing times or canceled jobs. Furthermore, we combine those obtained time slots to a best matching time slot that is unique for each resource. Upon this time slot, each resource plans its own schedule. Thus, for the first time, we enable decentralized production planning and control for an unlimited number of parallel required resources for one job and demonstrate how this production performs compared to a production using common queuing methods.

The paper starts with a problem classifaction and a review of related work. Afterwards we introduce the general concept of our multi-resource approach, followed by a detailed description of the scheduling procedure. Subsequently, we provide results of our empirical study and close the paper with our conclusion and an outlook on further work.

2 PROBLEM DESCRIPTION

2.1 Literature Research

Production planning and control can be done by either dispatching or scheduling mechanisms. While dispatching mechanisms mostly refers to a queuing approach using priority rules, scheduling mechanisms provide strict start and end times for each operation to be processed (Gehlhoff and Fay, 2020). Produc-

Krockert, M., Matthes, M. and Munkelt, T.
Agent-based Decentral Production Planning and Control: A New Approach for Multi-resource Scheduling.
DOI: 10.5220/0010436204420451
In *Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS 2021) - Volume 1*, pages 442-451
ISBN: 978-989-758-509-8

tion planning and control by dispatching rules are well known and researched. For example, (Frazier, 1996) published an evaluation study comparing 14 priority rules within a manufacturing cell, considering due, setup and processing times. (Montazeri and van Wassenhove, 1990) compared 20 rules for a flexible manufacturing system, considering material relations and transportation times. (Kim, 1990) compared dispatching rules considering alternative routings. Even the combinations of priority rules using fuzzy logic are researched by (Grabot and Geneste, 1994). All this research points out that the results depend heavily on the production environment and which objectives the production shall achieve. Due to the ongoing advances in the development of the Industrial Internet of Things (Jeschke et al., 2017) and a high product variety demand by customers (Mourtzis et al., 2012), production systems change towards decentral production systems. (Toivonen et al., 2017) summarized the advantages of decentral over central production control to higher flexibility, faster decision-making process, clearness of responsibilities as well as up to date information. Besides that, the possibility of immediate reactions to disruptions in the production process makes the decentral production planning and control superior to a centrally planned production (Munkelt and Krockert, 2018). Planning a set of jobs on a finite set of resources in a fixed order is considered as a job shop scheduling problem (JSP), which is well known and proven as np-hard (Lenstra and A. H. G. Rinnooy Kan, 1978). The problem has been extended multiple times to consider transportation (JSPT) (Nouri et al., 2016), dual resources constraints (Dhiflaoui et al., 2018; He et al., 2016), and alternative routes through the production, known as flexible job shop problem (FJSP) (Chan et al., 2006). The literature research for JSP including decentral production leads to agent-based approaches such as (Shen et al., 2006; Akkiraju et al., 2001; Adhau and Mittal, 2012; Gu et al., 2018). Studies that consider decentral planning and control are only available for project management, which is considered a multi-project planning problem (MPSP) like (Adhau and Mittal, 2012), who are using a Multi-Agent-System utilizing negotiation and cooperation between agents. The approaches in the studies of MPSP are not able to solve FJSP, due to different domain constraints. In this paper, we provide an approach for decentral production planning and control where multiple resources are required to process a job.

2.2 Planning of Lobs that Require Multiple Resources at the Same Time

In our production, a job consists out of a set of operations that require the same combination of resources to be processed. How these operations are assigned to the job is beyond the scope of this paper. Therefore, in our production, a job is defined by a start, a duration, a priority, and a combination of resources that are considered as capability. Jobs that require the same capability can be processed in sequence without changing that capability (Sarkar and Šormaz, 2019). Changing a capability affords time that is not productive. Our goal is to find a feasible schedule for jobs on the required resources while maintaining a high capacity utilization and a high adherence to delivery dates. Each job is routed through the production by planning time slots based on a given priority rule. As shown in Table 1, there are four different combinations of resource types, and more than one resource of a resource type may exist in a combination. But, as our literature research points out, there is a lack of decentral planning algorithms that are capable of scheduling jobs which require two or more resources for setup and/or processing. One basic approach to apply queuing mechanics in production is described by (Liu and Sycara, 1996). This can be extended for multi-resource planning and control by grouping jobs requiring the same resources to be setup/processed. For each job taken from the queue, the production waits until all resources are ready to start setup/processing. However, this causes idle times on resources. These idle times lead to time slots, which are not filled by other possibly less prioritized jobs. Furthermore, queueing systems do not intend to fill these time slots with other jobs. In comparison to the queuing-based production control, we present an approach that aims to enable flexible and robust multi-resource scheduling in an agent-based production.

3 OUR APPROACH TO MULTI-RESOURCE PLANNING

3.1 Structure of Our Decentral Production Planning and Control

Our decentral production and control operate event-driven by utilizing a Multi-Agent-System, to allow immediate (re)planning and control based on the feedback from production. This ensures the production

can react to changes immediately which is superior to central planning algorithms(Munkelt and Krockert, 2018). The production utilizes an agent-based approach, in which workers and machines are represented as Resources-Agents, and production orders are represented as Production-Agents. In addition, we utilize Hub-Agents to allocate jobs to each resource, and Job-Agents to keep track of all activities that are related to their assigned job. This agent-based approach is presented in more detail in our previous publications (Munkelt and Krockert, 2018). To prove our concept of decentral multi-resource production planning and control we implemented the approach into our production by extending the interaction between Hub-Agents, Job-Agents, and Resource-Agents to enable multi-resource planning and control. To allow jobs to be assigned to different resource combinations, we assign a required capability to each job. This capability can be fulfilled by different capability providers (Sarkar and Šormaz, 2019). Hereby, each capability provider is a unique combination of resources. While resources may be assigned to multiple capability providers, resources can be used only by one capability provider at the same time. How the resource is used is defined by the element called 'setting', that connects the resource and the capability provider. This type of capability assignment allows the alternative routing of jobs through production, to provide high flexibility on how the job can be processed, i.e., we assign a required capability to create a hole with a specific diameter to a job, and a capability provider holds the information how the hole can be achieved and what resources are required. For instance, there can be multiple capability providers that are capable to create that hole either by drilling, cutting, or punching it.

Table 1: Setup cases.

	Resource type	Overlapping Setup | Process
1	Machine	
2	Operator	
	Machine	
3	Operator	
	Machine	
	Worker	
4	Machine	
	Worker	

3.2 Job Synchronization by Utilizing the Job-Agents

To perform jobs on multiple resources at the same time, it is required to keep track of the resources and synchronize their jobs to recognize when a job is finished and a new job can start. The synchronization sequence is shown in Fig. 1. Once a job arrives, the

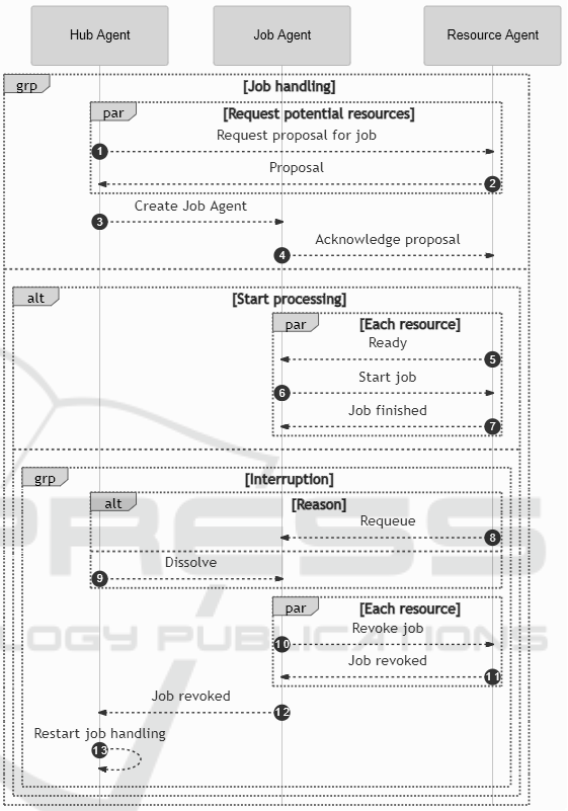

Figure 1: Messages during the processing of one job:
grp: is a grouping box
alt: is a box for alternative occurrences
par: is a box for send/recive by many participants.

Hub-Agent starts to (1) request a proposal from each Resource-Agent that is required to process that job. Each Resource-Agent is responding (2) with a proposal containing all available time slots, calculated based on the priority of the request. Once all proposals are received, the Hub-Agent determines the best time slot combination from them and creates a Job-Agent (3), which is a dedicated agent for one job. The Job-Agents are responsible for forwarding the acknowledgment to each selected resource (4) and handle all events send by the participating Hub- and Resource-Agents, such as ready signals and interrupting events, like a delay in the previous job. When a Resource-Agent becomes ready to process a job

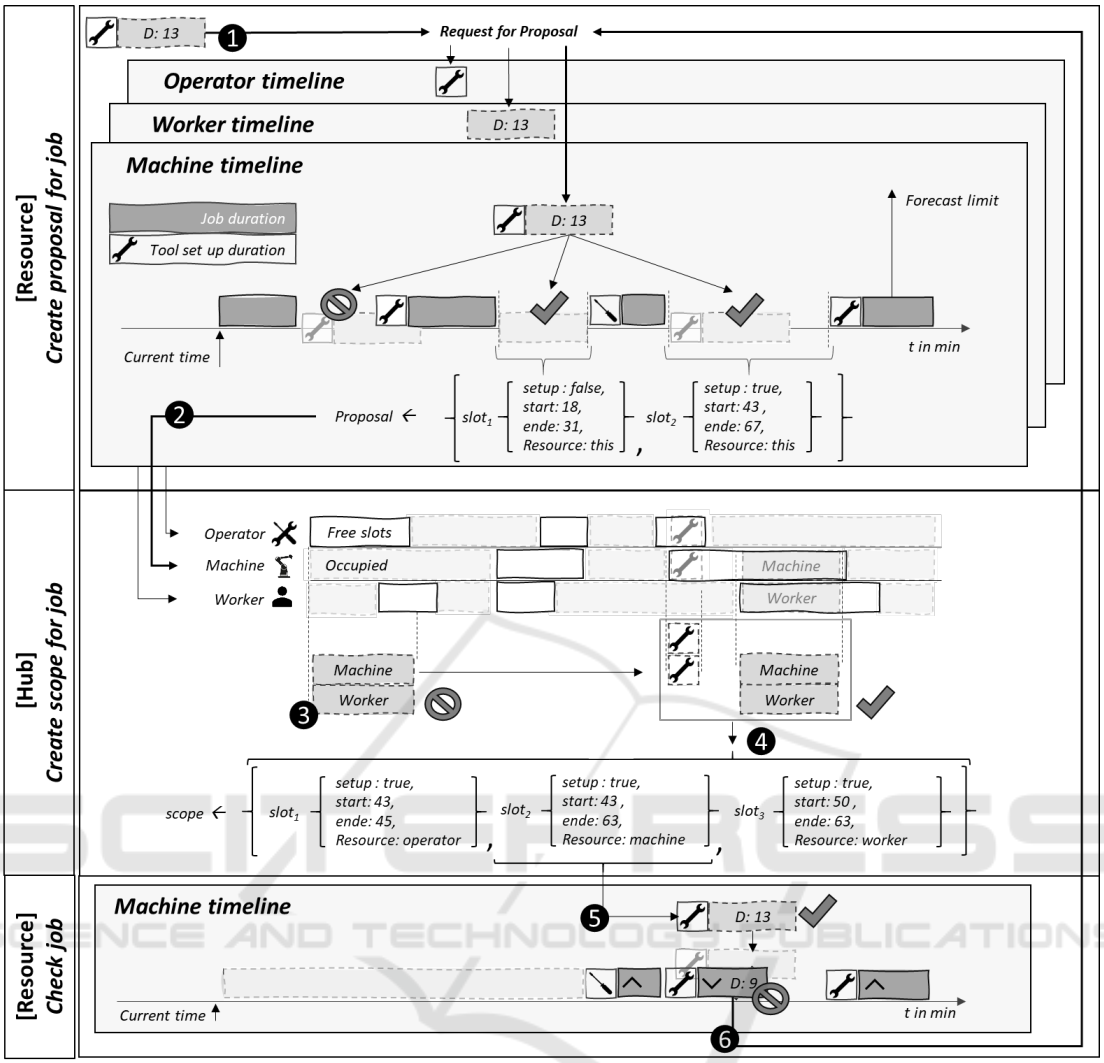

Figure 2: Overview from request proposal for job to obtained scope for job.

the Resource-Agent sends a ready to process message (5) to the Job-Agent. When all resources are ready and all material requirements are met, a job can start and the Job-Agent will send a start signal (6) to all Resource-Agents. Each Resource-Agent will finally return with a job finished message (7). Upon the finished message, the Job-Agent will be dissolved. However, if any unforeseen events occur, the affected agent can request requeue (8) or dissolve (9) from the Job-Agent. That request triggers the Job-Agent to revoke (10) the job from all assigned resources' schedules. If the processing has not started yet, the resources will remove the job from their schedule and acknowledge the revoke (11). Because the job still has to be processed, the procedure will start over until it finally succeeds with step number (7).

4 SCHEDULING PROCEDURE

4.1 Procedure Overview

The scheduling sequence for a job consists of three procedures illustrated in Fig. 2. The first procedure to find and 'create a proposal for job' containing a viable time slot is shown in the first lane in Fig. 2 using a machine as an example. The procedure starts with a request for proposal (1), which is created and sent by the Hub-Agent to all Resource-Agents defined by the required capability of the job. Depending on the required capability, a resource is used for setup only, for processing only or for both tasks. By receiving the request for proposal for the given job, the machine determines a suitable time slot based on its local

knowledge about its own scheduled jobs and the setup requirements for the requested job. In the next step, all possible time slots for the job are wrapped into a proposal (2). The created proposal is sent back to the Hub-Agent and the procedure 'Create scope for job' shown in the second lane starts, as soon as all proposals for that job are received. The Hub-Agent maps the time slots (3), selects the first time slots that match all requirements and merges them into one scope that will be assigned to the job (4). In the last procedure 'Check job' shown in the third lane, the Hub-Agent sends a job acknowledgment containing the scope to all selected resources, as mentioned in section 3.2. Then, each resource has to check if the slot still fits into the current schedule. This check is necessary because of the asynchronous processing of requests and the possibility of changes in the time between the proposal for that job and the acknowledgement of the job. Hereby all overlapping times from scheduled jobs are compared with the acknowledged job. If there are higher prioritized jobs scheduled the acknowledged job or the lower prioritized job will be sent, to restart the scheduling sequence.

4.2 Proposal Procedure

During the sequence described in chapter 3.2 and 4.1, each resource creates a proposal *prpsl* based on its scheduled jobs and the priority of the requested job.

Table 2: Symbol definition.

Symbol	Definition
t	type $t \in \{0,1\}$
T	is a tuple of $\{t_{setup}, t_{processing}\}$
pt	previous task required with pt *in* $\{0,1\}$
s	start $s > 0, s \in \mathbb{N}$
e	end $e > s, e \in \mathbb{N}$
d	duration $d > 0, d \in \mathbb{N}$
p	priority determined by the priority rule
j, suc, pre	job is a column vector with $(p,s,e,d,c)^T$
J	is a set of jobs $\{j_1, \cdots, j_n\}$ ordered by s(j)
r	is a resource $r \in R$
c	capability is a set $\{cp_1, \cdots, cp_n\}$
cp	capability provider, is a vector with $\begin{cases} STG \leftarrow \{stg \in STG \mid stg(cp)\} \\ d \leftarrow d \in D \end{cases}$
$slot$	scope is a column vector with $(s,e,pt,r)^T$
$prpsl$	proposal is a set $\{slot_1, \cdots, slot_n\}$
scp	scope is an acknowledged set $\{slot_1, \cdots, slot_n\}$
stg	setting is a column vector with $(cp,r,scp,T)^T$

Algorithm 1: Request for proposal.

Input: $jtp \leftarrow$ *job to propose*
$r \leftarrow$ *this resource*
$J \leftarrow$ *scheduled jobs*

Result: $PRPSL$; // list of possible slots

Data: $mft \leftarrow$ *maximum forecast time* ; // latest end time $\leftarrow$ *current time*

1 $PRPSL \leftarrow \{\emptyset\}$; // emty store for slots
2 $s \leftarrow \max \in \{e(job(r)) \wedge time\}$; // earliest start
3 $d \leftarrow 0$; // save job duration
/* add setup time if required */
4 **if** $t_{setup}(c(cp(jtp)) = 1$ **then**
5 $d \leftarrow d(cp(jtp))$
/* add processing time if required */
6 **if** $t_{processing}(c(cp(jtp)) = 1$ **then**
7 $d \leftarrow d + d(jtp)$
/* get all jobs with higher priority */
8 $J \leftarrow \{j \in J | p(j) < p(jtp)\}$
9 **if** $J \neq \emptyset$ **then**
10 $pre \leftarrow Dequeue(J)$
11 $s \leftarrow e(pre)$
/* and check setup requirement */
12 $pt \leftarrow stp(pre) = stp(jtp) \vee t_{processing}(c(cp(jtp)) = 0$
/* then loop through remaining jobs */
13 **while** $J \neq \emptyset$ **do**
14 **if** pt **then**
15 $d_{temp} = d - d(cp(jtp))$
/* get next item from queue */
16 $suc \leftarrow Dequeue(J)$
/* check if the job fits between */
/* predecessor and successor */
17 **if** $d_{temp} \leq s(suc) - s$ **then**
/* create a new scope and add it to possible scopes */
18 $PRPSL \leftarrow PRPSL \cup \left\{ slot \leftarrow \begin{cases} s \leftarrow s \\ e \leftarrow s(suc) \\ pt \leftarrow pt \\ r \leftarrow r \end{cases} \right\}$
19 $pre \leftarrow suc$; // save predecessor as successor
20 $s \leftarrow e(pre)$
/* and check setup requirement */
21 $pt \leftarrow stp(pre) = stp(jtp) \vee t_{processing}(c(cp(jtp)) = 0$
end
/* add a final scope after the last job */
22 $PRPSL \leftarrow PRPSL \cup \left\{ slot \leftarrow \begin{cases} \leftarrow s \\ e \leftarrow mft \\ pt \leftarrow pt \\ r \leftarrow r \end{cases} \right\}$

Algorithm 1 describes how the Resource-Agent iterates through the scheduled jobs J and creates a scope scp upon a fitting time $slot$. Algorithm 2 shows how the Hub-Agent finds the best schedule out of all proposed scopes. Algorithm 3 shows the final check each resource does to ensure its availability. Table 2 introduces all variables used in all algorithms. Every time a Resource-Agent receives a request for proposal al-

gorithm 3.2 is initialized with a maximum forecast time, a time that a resource looks ahead to get valid time slots. Next, the algorithm sets the earliest possible start to the end of the current job or, if currently, no job is in process, to the current time. Furthermore, an empty set of time slots is initialized as *PRPSL* and the job duration temporarily set to zero (see lines 1-3). Succeedingly, the Algorithm summarizes the duration of the setup and processing depending upon the resource is used for either one or both (see lines 4-7).In the following step, the Algorithm separates all scheduled jobs that are more important than the requested job into a separate priority queue ordered by job priority (see line 8). If the created queue holds any jobs, the Algorithm dequeues the first job and replaces the earliest possible start (see lines 10-11), stores the job as predecessor and determines if a setup is required, because the predecessor job may differ from the current setup on the Resource (see line 12). If the queue is still containing jobs, the algorithms start to iterate through the remaining jobs. For each remaining job, the algorithm reduces the job duration by the defined setup time if no setup is required (see lines 13-15). Then the algorithm takes the next job from the queue as the successor to the current job and checks if the requested job fits between both jobs (see lines 16-17). If the job fits, the algorithm creates a new slot and adds that slot to the proposal (see line 18). After that, the algorithm continues by assigning the successor to the predecessor, sets a new start, and checks the setup condition for the next iteration (see lines 19-21). Lastly, if the queue is empty, the algorithm will terminate by creating the last scope and adding it to the list of possible time slots to ensure that at least one feasible time slot is returned as a proposal for each request (see line 22).

4.3 Create Scope

After receiving all proposals from each resource for the requested job, the Hub-Agent creates a scope for each setting of the capability provider from all received proposals. Thus, each setting contains one resource and the set of scopes containing the returned time slots. Depending on their purpose specified in the setting, a resource can be used either for setup or processing or both task types, setup and processing, see Table 1. To determine the best proposal, the Hub-Agent starts algorithm 2 for each capability provider by creating an empty set of scopes and seperates the list of settings into a list for setup and a list for process (see lines 1-3). Both lists are sorted ascending by the earliest start of the contained slots defined by $min(s(slot(scope(stg))))$. As some resources, like

Algorithm 2: Create scope.

Input: $cj \leftarrow$ *current job*
$ccp \leftarrow$ *current capability provider*

Result: *SCP as list of matching slots*

1 $SCP \leftarrow \{\emptyset\}$
2 $process \leftarrow \{stg \in STG \mid stg(ccp) \cap t_{processing}(T(x)) = 1\}$
3 $setup \leftarrow \{stg \in STG \mid stg(ccp) \cap t_{setup}(T(x)) = 1\}$
4 $first \leftarrow Dequeue(process)$
5 **while** $SCP \neq \{\emptyset\} \wedge slot(first) \neq \{\emptyset\}$ **do**
6 $nextSlot \leftarrow Dequeue(slot(first))$
7 **if** $process \neq \{\emptyset\}$ **then**
8 $slot_{comp} \leftarrow GetMatches(d(cj), comp, process)$
else
9 $slot_{comp} \leftarrow nextSlot$
end
10 $latestend \leftarrow e(slot_{comp}) - cd$
11 $SCP \leftarrow slot_{comp}$
12 **if** $slot_{comp} \neq \{\emptyset\} \wedge pt(slot_{comp})$ **then**
13 $slot_{comp} \leftarrow \begin{cases} s \leftarrow 0 \\ e \leftarrow latestend \\ pt \leftarrow pt(slot_{comp}) \\ r \leftarrow r \end{cases}$
14 $comp \leftarrow GetMatches(d(ccp), slot_{comp}, setup)$
15 **if** $slot_{comp} \neq \{\emptyset\}$ **then**
16 $les \leftarrow \max_{s \in slotes} \{s(S), e(slot_{comp}))\}$
17 $slot_{stp} \leftarrow \begin{cases} s \leftarrow les - d(cpp) \\ e \leftarrow les \\ pt \leftarrow pt(slot_{comp}) \\ r \leftarrow r \end{cases}$
18 **if** $slot_{comp} \neq \{\emptyset\}$ **then**
19 $esj \leftarrow \max_{s \in slotess} \{e(slot_{stp}), s(SCP))\}$
20 $SCP \leftarrow slot_{stp} \cup \begin{cases} s \leftarrow esj \\ e \leftarrow esj + d(cj) \\ pt \leftarrow pt(slot_{comp}) \\ r \leftarrow r \end{cases}$
else
21 $SCP \leftarrow \{\emptyset\}$
end
end
Function $GetMatches(d, slot_{match}, STG)$**:**
22 $slot_{reduced} \leftarrow \{\emptyset\}$
23 **while** stg_{temp} *in* $STG \wedge slot_{match} \neq \{\emptyset\}$ **do**
24 **while** $slot_{reduce} = \emptyset \wedge slot(stg_{temp}) \neq \{\emptyset\}$ **do**
25 $slot_{temp} \leftarrow Dequeue(slot(stg_{temp}))$
26 $slot_{reduce} \leftarrow Reduce(d, (slot_{match} \cap slot_{temp}))$
end
27 $slot_{match} \leftarrow slot_{reduce}$
end
return $slot_{match}$
Function $Reduce(d, SLOT_{compare})$**:**
28 $slot_{reduce} \leftarrow \{\emptyset\}$ $s \leftarrow \max_{slot} \in SLOT_{compare}\{s(slots)\}$
$e \leftarrow \min_{slot} \in SLOT_{compare}\{e(slots)\}$
29 **if** $e - s \geq d$ **then**
30 $slot_{reduce} \leftarrow slot \begin{cases} s \leftarrow s \\ e \leftarrow e \\ pt \leftarrow \exists\, pt(slot) \\ r \leftarrow r \end{cases}$
return $slot_{reduce}$

the machine itself, must be used for setup and processing, a setting from the resource appears in both lists. After creating the two lists, the algorithm takes the first setting from the list for processing (see line 4). Each of the time slots contained in the scopes has to be compared with the scopes from the other required resources to determine possible matching time slots for process and, if the scope for process requires it, an additional time slot for setup. While iterating through the list of time slots of the resource (see lines 5-21), possible time slots from the first resource and all other resources from the list of process for the given job duration are evaluated (see line 8), by calling the function: "*GetMatches*" with the jobs' duration, the time slot "*nextSlot*" and the remaining time slots from "*process*". *GetMatches* shrinks all given slots, by calling the function: "*Reduce*", until it finds the first matching slot or no slot is found (see line 22-27). *Reduce* returns a time slot that fits into both given time slots by calculating the minimum end and the maximum start of both (see lines 28-30). Depending on whether the function returns a possibly reduced time slot for all process resources or not, the algorithm also checks if a setup for the time slot is required (see line 12). If no setup is required and a time slot exists (see line 18), the algorithm creates the concrete time slot for all processing resources (see lines 19-20) and adds it to the returned scope. Otherwise, if a setup is required, the algorithm creates a temporary time slot and calls *GetMatches* again, this time with the temporarily created time slot and the settings from "*setup*". During the second call, the time slot must fit before the latest start of the process (see lines 10 and 14). If any possible matches for setup exist, the algorithm creates the concrete time scope for setup (see lines 16-17) and finally adds all time slots to the returned scope (see line 20). If no matches could be found, the algorithm takes the next scope from the first resource and tries to find matches until a match has been found or all scopes for the first resource have been checked. The result of Algorithm 2 can be either the matching scope or an empty result, which is the case if no suitable scope could be found. If Algorithm 2 returns a scope, the approach assigns the job to the capability provider with the scope containing the earliest start and acknowledges the job to the associated resources of the settings in the capability provider. If no capability provider can offer a scope for the requested job, the job idles until the Hub-Agent repeats the call for proposal. This repeated call is scheduled after the current time plus maximum forecast time.

4.4 Check Acknowledged Job

After the resource has received a job acknowledgment for a suggested time slot, the resource must check if the time slot of that job fits into its current schedule, as mentioned in section 4.1. The algorithm 3 shows the procedure to check the acknowledged job and rejects the acknowledged job or any job from the scheduled jobs that have an overlapping time slot and therefore needs to be rescheduled because of a lower priority. For that purpose, the algorithm defines an empty set of $JOB_{requeue}$ (see line 1) and a set of $JOB_{overlap}$ containing all slots that overlap with the time slot from the acknowledged job (see line 2). If $JOB_{overlap}$ contains any higher prioritized jobs, the acknowledged job will be rejected and set to requeue (see line 4). Otherwise the job gets scheduled and, if there are any jobs in $JOB_{overlap}$, they will be requeued (see lines 6-7).

Algorithm 3: Check Acknowledged Job.

Input: $j_{ack} \leftarrow jobtocheck$
$J \leftarrow scheduled\ jobs$

Result: $JOB_{requeue}$; // jobs to requeue

1 $JOB_{requeue} \leftarrow \{\emptyset\}$
2 $JOB_{overlap} \leftarrow \{j \in J | (e(j) < e(j_{ack}) \wedge e(j) > s(j_{ack})) \vee (s(j) > s(j_{ack}) \wedge s(j) < e(j_{ack}))\}$
3 **if** $(\exists j \in JOB_{overlap} | p(JOB_{overlap}) < p(j_{ack}))$ **then**
4 $\quad JOB_{requeue} \leftarrow \{j_{ack}\}$
5 **else**
6 $\quad J \leftarrow J \setminus JOB_{overlap} \cup \{j_{ack}\}$
7 $\quad JOB_{requeue} \leftarrow JOB_{overlap}$

5 EXPERIMENTAL STUDY

5.1 Experiment Description

To evaluate our approach we implemented the algorithms of the proposed approach in our agent-based production planning and control. The production configuration is shown in Fig. 3.

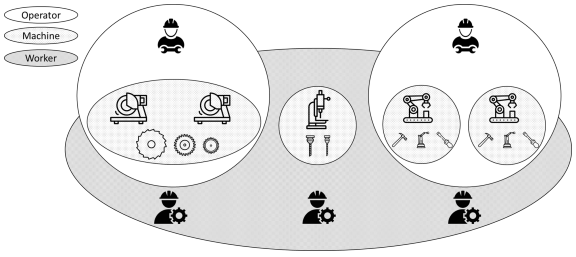

Figure 3: SSOP.

It includes three work centers with two sewing machines, one drilling machine and two assembly stations. Moreover, we introduced two groups of employees. The first group are operators, who are responsible for equipping tools onto machines. The second group consists of workers, who are responsible for processing jobs using one of the machines. With this configuration, we produce two products consisting of 15 assemblies with two to three operations each. Each operation requires a different capability with a different set of resources. The operations are grouped by their capability to jobs. This production configuration ensures that the machine resources have to change their setting frequently. For the evaluation, we simulated four weeks of nonstop production, where new customer orders arrive log normally distributed at the production. Additionally, we varied the processing time of the jobs to cover uncertainty in the production process to evaluate stability and flexibility. We compared our approach for a decentral production planning and control with a central queueing-based production planning and control and combine each of the production planning and control approaches with each of the following dispatching rules: FIFO, LST, MDD and SPT (see equations 1, 2, 3, 4).

$$
\begin{aligned}
t &= current\ time\ at\ the\ shop\ floor \\
j_i &= job\ for\ which\ the\ priority\ is\ determined \\
a_{ji} &= release\ (arrival)\ time\ of\ job\ j \\
&\quad on\ the\ shop\ floor \\
dt_{ji} &= due\ time\ of\ the\ job_j\ in\ the\ shop\ floor \\
d_{ji} &= processing\ duration\ for\ job\ j_i \\
j_{ik} &= all\ successor\ jobs\ k\ from\ job\ j_i \\
d_{jik} &= processing\ duration\ for\ all\ sucessor\ jobs \\
&\quad k\ from\ job\ j_i
\end{aligned}
$$

$$\textbf{L}east\ \textbf{S}ack\ \textbf{T}ime\ (LST) = dt_{ji} - t - d_{ji} - d_{ijk} \quad (1)$$

$$\textbf{M}odified\ \textbf{D}ue\ \textbf{D}ate = max\{dt_{ji}, t - (d_{ji} + d_{ijk})\} \quad (2)$$

$$\textbf{F}irst\ \textbf{I}n\ \textbf{F}irst\ \textbf{O}ut\ (FIFO) = a_{ji} \quad (3)$$

$$\textbf{S}hortest\ \textbf{P}rocessing\ \textbf{T}ime\ (SPT) = d_{ji} \quad (4)$$

To achieve comparability between both approaches regarding the underlying production environment, we implemented an additional behavior into our simulation by developing a new hub-agent that allows central queue-based production planning and control as described in chapter 2. Now, for the central queue-based production planning, the hub agent determines the next job with the highest priority and waits until all necessary resources are available. In addition, all jobs with the same setup requirements that are ready for processing are also assigned to these resources. This procedure is called exhaustive behavior and is well known to improve the overall performance of the production by reducing the number of setups and ultimately reducing the overall setup time. (Frazier, 1996) All simulation parameters we applied are summarized in Table 3.

Table 3: Simulation parameters.

Value	Unit	Description
28	days	simulation end time, asuming 24/7 worktime
2	days	settling time
1	hour	average customer order arival time
36	hours	average time from order placement to delivery
20 and 0	%	deviation of estimated operation processing time

5.2 Experimental Results

In order to evaluate our approach, we performed experiments based on all possible combinations of simulation types and priority rules. Combining simulation type decentral production planning and control and central queueing with the four priority rules, FIFO, LST, MDD, SPT results in eight different simulation experiments. By adding log-normally distributed deviations for the processing time of jobs, we simulate uncertainties to evaluate the flexibility and robustness of our algorithm in the presented production. Therefore the results are split into two sections and sum up to 16 experiments in total, where the top section shows all simulation results without deviation of the job's processing time, while the bottom section shows all results applying 20% deviation. Each experiment consists of ten simulation runs. While each simulation run has been using a different seed to create the sequence of order arrivals and delivery dates, each experiment has been using the same seeds, to share the same sequence of events for each experiment. The presented results are the arithmetic mean values over the simulation runs of each experiment based on the equations 5-10, which are common key performance indicators for production (Jodlbauer, 2007). The statistical error for all experiment results was below *0.5%*.

Makespan is calculated from production start until the job finishes.

$$Makespan = \frac{\sum_{j \in J} C_j - S_j}{|\{j \in J\}|} \quad (5)$$

Tardiness is the overdue of all jobs divided by the jobs that are overdue.

Table 4: Results.

Deviation in %	Simulation Type	Priority Rule	Makespan in min	Orders processed in pcs.	Orders over due in pcs.	Adherance to due in %	Tardiness in min	Lateness in min	Processing time in %	Setup time in %	Workload time in %
0	DPPC	FIFO	1470	596	17	97.08	81	-929	73.01	16.36	89.37
		LST	1480	596	36	93.96	130	-919	73.03	16.39	89.42
		MDD	1466	599	40	93.32	159	-926	73.43	16.22	89.65
		SPT	1756	596	84	85.92	388	-643	73.01	15.86	88.86
	Queuing	FIFO	2356	592	270	54.39	419	-43	71.96	9.77	81.73
		LST	2298	594	241	59.41	401	-101	71.94	9.77	81.71
		MDD	2307	593	238	59.87	445	-92	72.06	9.69	81.75
		SPT	4037	562	470	16.37	2006	1637	71.52	9.87	81.38
20	DPPC	FIFO	1621	596	47	92.11	200	-778	74.51	15.50	90.01
		LST	1603	596	53	91.11	222	-796	74.42	15.66	90.07
		MDD	1614	596	68	88.59	298	-785	74.38	15.72	90.09
		SPT	1950	596	146	75.50	488	-449	74.42	14.83	89.25
	Queuing	FIFO	2482	592	273	53.87	507	83	71.98	9.18	81.17
		LST	2476	589	308	47.66	505	76	73.01	9.25	82.26
		MDD	2502	588	292	50.32	572	102	72.98	9.27	82.25
		SPT	4142	551	483	12.34	2047	1742	72.71	9.34	82.05

$$Tardiness = \frac{\sum_{j \in J} max(0, due_j - C_j)}{|\{j \in J | (due_j - C_j) > 0\}|} \quad (6)$$

Lateness is the total late or earlyness divided by amount of jobs.

$$Lateness = \frac{\sum_{j \in J} C_j - due_j}{|\{j \in J\}|} \quad (7)$$

Productive time is the time a resource is generating value.

$$Productive\ Time = \frac{\sum_{j \in J} C_j - S_j}{|R| * T_r} \quad (8)$$

Setup time is the time a resource is blocked to set-up.

$$Setup\ Time = \frac{\sum_{e \in E} C_e - S_e}{|R| * T_r} \quad (9)$$

Workload is the total time a resource is bussy.

$$Workload = Setup\ Time + Productive\ Time \quad (10)$$

$$\begin{aligned} se &= simulation_{end} - simulation_{start} \\ j &= job\ j\ \in\ J \\ due_j &= due\ time\ of\ job\ j \\ C_j &= completition\ time\ of\ job\ j \\ S_j &= releasetime\ of\ job\ j \\ T &= total\ planned\ availability\ time \\ R &= all\ resources\ r \in R \\ k &= number\ of\ processed\ jobs \\ e &= tool\ exchange\ (setup)\ e \in E \end{aligned}$$

All experiments were able to keep the workload in the production around 82%. Only the DPPC is able to increase the workload to 90% due to more frequent set-ups, which results in a higher set-up time than the other methods. Although all experiments were able to complete 562 to 596 customer orders, the SPT queuing is clearly the least viable solution in terms of adherence to due. This confirms that SPT queues are not practical for continuously arriving jobs that may depend on each other. While the job duration does not deviate the applied time-dependent based queuing mechanics LST, MDD and FIFO show an adherence to due of about $54-60\%$ and drop to $47-53\%$ by considering 20% deviation. Only DPPC outperforms the queueing approaches in all values and shows its superiority. The dPPC was able to reduce the mean tardiness to 23% of the central queueing approach. Even the queuing LST, MDD and FIFO approaches with a lateness close to the due date were not able to compete with the dPPC in the other performance indicators. The makespan was also reduced to 56% by dPPC compared to the central queuing.

6 CONCLUSION AND OUTLOOK

The target of our research was to create an multi-resource scheduling solution in a highly diverse production characterized by uncertainty. For this purpose, we developed a negotiation protocol based on the contract net protocol, the job's priority, and the time slots of all resources as well as three algorithms, one to offer a proposal with free time slots on resources, one to evaluate the best matching time scopes from a given set of proposals and one to check the acknowledged scope. We deployed our approach onto our self-organizing production and we are able to verify our concept through various simulation runs with and without consideration of uncertainty in the production process. The experiments also showed a great improvement versus common central queueing approaches. An extensive testing and a comparison to

common centralized scheduling solutions is still outstanding and will be done in the near future. Next, we want to extend the solution to handle more than the two task types setup and processing steps, like loading and unloading materials for a job, and evaluate how the developed solution is performing with these new constraints.

ACKNOWLEDGEMENTS

The authors acknowledge the financial support by the German Federal Ministry of Education and Research within the funding program "Forschung an Fachhochschulen" (contract number: 13FH133PX8).

REFERENCES

Adhau, S. and Mittal, M. L. (2012). A multiagent based system for resource allocation and scheduling of distributed projects. *International Journal of Modeling and Optimization*, pages 524–528.

Akkiraju, R., Keskinocak, P., Murthy, S., and Wu, F. (2001). An agent-based approach for scheduling multiple machines. *Applied Intelligence*, 14(2):135–144.

Chan, F. T. S., Wong, T. C., and Chan, L. Y. (2006). Flexible job-shop scheduling problem under resource constraints. *International Journal of Production Research*, 44(11):2071–2089.

Dhiflaoui, M., Nouri, H. E., and Driss, O. B. (2018). Dual-resource constraints in classical and flexible job shop problems: A state-of-the-art review. *Procedia Computer Science*, 126:1507–1515.

FIPA TC C (03.12.2002). Fipa contract net interaction protocol specification.

Frazier, G. V. (1996). An evaluation of group scheduling heuristics in a flow-line manufacturing cell. *International Journal of Production Research*, 34(4):959–976.

Gehlhoff, F. and Fay, A. (2020). On agent-based decentralized and integrated scheduling for small-scale manufacturing. *at - Automatisierungstechnik*, 68(1):15–31.

Grabot, B. and Geneste, L. (1994). Dispatching rules in scheduling: a fuzzy approach. *International Journal of Production Research*, 32(4):903–915.

Gu, M., Gu, J., and Lu, X. (2018). An algorithm for multi-agent scheduling to minimize the makespan on m parallel machines. *Journal of Scheduling*, 21(5):483–492.

Guizzi, G., Revetria, R., Vanacore, G., and Vespoli, S. (2019). On the open job-shop scheduling problem: A decentralized multi-agent approach for the manufacturing system performance optimization. *Procedia CIRP*, 79:192–197.

He, J., Li, Q., and Xu, D. (2016). Scheduling two parallel machines with machine-dependent availabilities. *Computers & Operations Research*, 72:31–42.

Jeschke, S., Brecher, C., Song, H., and Rawat, D. B., editors (2017). *Industrial Internet of Things*. Springer Series in Wireless Technology. Springer International Publishing, Cham.

Jodlbauer, H. (2007). *Produktionsoptimierung: Wertschaffende sowie kundenorientierte Planung und Steuerung*. Springer Vienna.

Kim, Y.-D. (1990). A comparison of dispatching rules for job shops with multiple identical jobs and alternative routeings. *International Journal of Production Research*, 28(5):953–962.

Lenstra, J. K. and A. H. G. Rinnooy Kan (1978). Complexity of scheduling under precedence constraints. *Operations Research*, 26(1):22–35.

Liu, J. and Sycara, K. (1996). Multiagent coordination in tightly coupled task schedulingmultiagent coordination in tightly coupled task scheduling. *Proceedings of the Second International Conference on Multiagent Systems*, pages 181–188.

Miao, C. and Zou, J. (2015). Parallel-machine scheduling with time-dependent and machine availability constraints. *Mathematical Problems in Engineering*, 2015:1–6.

Montazeri, M. and van Wassenhove, L. N. (1990). Analysis of scheduling rules for an fms (flexible manufacturing system). *International Journal of Production Research*, 28(4):785–802.

Mourtzis, D., Doukas, M., and Psarommatis, F. (2012). Design and planning of decentralised production networks under high product variety demand. *Procedia CIRP*, 3:293–298.

Munkelt, T. and Krockert, M. (2018). An approach to a self-organizing production in comparison to a centrally planned production. In *Tagungsband ASIM 2018 – 24. Symposium Simulationstechnik*, pages 299–306. ARGESIM.

Nouri, H. E., Driss, O. B., and Ghédira, K. (2016). A classification schema for the job shop scheduling problem with transportation resources: State-of-the-art review. In Silhavy, R., Senkerik, R., Oplatkova, Z. K., Silhavy, P., and Prokopova, Z., editors, *Artificial Intelligence Perspectives in Intelligent Systems*, volume 464 of *Advances in Intelligent Systems and Computing*, pages 1–11. Springer International Publishing, Cham.

Rohloff, M. (1993). Decentralized production planning and design of a production management system based on an object-oriented architecture. *International Journal of Production Economics*, 30-31:365–383.

Sarkar, A. and Šormaz, D. (2019). Ontology model for process level capabilities of manufacturing resources. *Procedia Manufacturing*, 39:1889–1898.

Shen, W., Wang, L., and Hao, Q. (2006). Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 36(4):563–577.

Toivonen, V., Järvenpää, E., and Lanz, M. (01.11.2017 - 03.11.2017). Managing production complexity with intelligent work orders. In *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 189–196. SCITEPRESS - Science and Technology Publications.