Factory Orchestration with Genetic Algorithm

Simon Torka, Yacine Zahidi, Friedrich Mangelsdorf and Antonia Düker, *Group 1*

***Abstract* — The genetic algorithm for factory orchestration is needed due to its good results and flexibility. We present a genetic algorithm matching a 3D-printing factory setup and its surroundings so as to reduce the time needed and improve the quality of production. We programmed a factory setup to include multiple printers, conveyor belts and inspection stations. The makespan and the quality of the output are dependent on the complexity and the size of orders as well as inherent parameters of the printers. The genetic algorithm was tested with different weights in the fitness function, and different amounts of orders. The best results were achieved, when the weights of quality and makespan were nearly balanced.**

I. Introduction

In 2017 a 3D-printing smart factory was simulated in the ROS-environment as a use case scenario of the CHARIOT project at the DAI-Labor, AOT, TU Berlin. Chariot is “an IoT middleware for the integration of heterogeneous entities in an urban smart factory” [5]. The use case scenario serves for researching on data usage for optimizing production flow and controlling a smart factory. The simulated factory consists of 3D-printers, a conveyor belt system, quality inspection units, transportation robots and storage location, all of various amounts. A 3D-object being printed runs through the above mentioned components.

The existing architecture consists of a knowledge manager agent for triggering and handling requests, a conveyor belt motor component, a 3D-objects component and a cloud component. Through a shell, the user can interact with the knowledge manager agent. The 3D-objects component extracts features of 3D-objects from ".stl" files, writing them into a ".csv"-file. The conveyor belt motor component generates the data, writing them into a ".csv"-file. This model can be initialized several times to simulate various amounts of motors. In the cloud component, data storage and computer performance is provided.

As input from multiple users, the 3D-printing factory gets a various amount of 3D-object production requests. The users’ requests need to be orchestrated on the devices of the factory. That means in detail, for each job request the decision of which 3D-printer. Regarding manufacturing, making this decision considering an entire list of job requests is called scheduling. Arranging job requests and allocating them over time on resources of the factory aims at optimizing the cost function, which depends on e.g., production time and quality.

The problem is a production request like printing, transporting or inspecting. Output should be orchestration through heterogeneous devices like a scheduled list for the orders which maps models to printers, conveyor belts and inspection stations. The devices and services should be semantically annotated like in ontology for industry.

Our goal is the creation of ontologies for the system and its components, the service orchestration with the required communication and calling of appropriate services to get necessary information before scheduling like 3D model clustering giving information about the possibility of printing a model at a specific printer. Then the scheduling starts with quality and makespan prediction of each model and genetic scheduling based on a fitness function considering the quality and makespan of each model. The orchestration can start to execute the plan by printing the models at a 3D printer, convey them to an inspection unit with a conveyor belt and analyze the quality at an inspection unit. Each step logged in a database.

The Job Scheduling is a NP-hard problem. It can be solved with exact methods or heuristics. The most promising approach is the local search. One method is the genetic algorithm which optimizes scheduling with genetic operations like mutation, crossover, and selection. The genetic algorithm optimizes towards a fitness function which takes weighted costs into account.

We use a genetic algorithm with costs for makespan and quality of each chromosome. Makespan and quality are predicted and written into a database. The genetic algorithm uses the predicted makespan and quality as input and proposes chromosomes which are altered with genetic operations. Chromosomes are optimized towards a fitness function which considers the costs for the chromosomes makespan and overall quality.

The system is handled by the knowledge manager which uses the MAPE-K blueprint [8] in a hybrid form with a finite state machine. It has units to monitor, analyze, plan and apply actions to the managed system. While the orchestration uses threads to trigger the production line units, to get feedback like the quality and track all actions and information in the database.

To analyze the genetic algorithm, we build up a test system to get results about the impact of weighting, and the dynamics and runtime. We measured the overall fitness of each winning chromosome, the makespan, quality and setup. For the genetic algorithms’ performance we analyzed the runtime, the number of generations and the chromosome setup.

The runtime has a linear behavior increasing with the amount of orders. Weighting has also a small impact on the runtime, because of the necessity for the algorithm to perform further actions. Makespan and quality cost weighting also have an impact on the achieved quality and makespan; Overall, the quality peaked when the quality cost took 70% of the fitness. The makespan improved when the cost of the makespan got larger - and for 100% of fitness the makespan was the shortest, but from 40% of fitness the makespan was reasonably low. Both costs should be either balanced or quality could be calculated out.

In this paper, we will write about other works solving the job scheduling problem. We will describe our system setup in the methodology starting with the general framework, showing our system communication and thread structure, the knowledge manager handling the information exchange, and the orchestration handler which operates the execution. The scheduling itself is described with the quality prediction by a neural network and the execution units itself which are the printers, conveyor belts and inspection stations. Eventually, we will present our results and finish with conclusion.

II. Related Works

The following analysis of possible solution methods for the job problem is based on the paper [6] from 1996. It is divided into the sections exact methods, approximate approaches and heuristics. Another section explains the related work "Genetic Algorithm based on Some Heuristic Rules for Job Shop Scheduling Problem" from 2012 [7], which introduces a genetic algorithm our scheduling algorithm is based on.

A. Exact Methods

The exact methods include branch and bound and refinements to this method like lower bounds approach, branching and valid inequalities. Those are based on enumerations of all feasible solutions of a combinatorial optimization problem. This can be also understood as decision tree approach. The lower bounds method adds strong lower bounds to cut branches of the enumeration tree as early as possible. The branching approach makes use of decision tree method for a preemptive one- machine solution. And the valid inequality-method is a special branch and bound approach with removal of branches based on inequalities.

B. Approximate Approaches

Another approach to NP-hard problems takes approximation algorithms. Since, it is not always feasible to calculate the optimal solution, a near-optimal solution is chosen to solve the problem. One solution can be priority rules. Known from scheduling tasks in operating systems, those rules can be taken to solve the job scheduling problem. Those rules consider priorities or constraints and can be preemptive or non-preemptive. Our problem can consider non-preemptive rules. Some examples are the shortest operation time rule - the order with the shortest operation time is done first. Other rules are: the longest operation time, the longest remaining processing time, the shortest remaining processing time, the longest operation remaining processing time, Random, first come first served, the shortest processing time, the longest processing time, the longest operation successor, the smallest number of remaining operations and the largest number of remaining operations. Another approximation approach is the shifting bottleneck heuristic. This heuristic searches for the near optimal solution for each machine and finds thereby the bottleneck.

C. Heuristics

A common heuristic is the local search. This heuristic searches for a better solution by replacing the first solution by a neighbor. There are different algorithms to find the neighbor and different stop criteria. Heuristics always optimize to reach their target function. One example is the tabu search with simulated annealing. The taboo search sets prohibitions on already visited solutions and makes sure to find the global minimum with simulated annealing as stop criteria. Simulated annealing counts down from a preset number until the cold temperature is reached. The constraint propagation disassembles the problem into constraints and solves them sequentially.

The genetic based job scheduling takes methods from genetics to optimize the fitness. Because the local search creates multi-sets out of multi-sets, it can be seen as parallel search. The idea of the algorithm is to look for good properties. Operations from biology are used to create the different sets. Those operations are mutation, crossover, and selection.

Newer methods are ant colonization algorithm, Bee colony algorithm, particle swarm, Q-learning and reinforcement learning. Those papers were published since 2017.

We decided to use the genetic algorithm because it suits our problem very well. The algorithm is one of the fastest and gives a good solution in short time. The solution might not be the best solution, but it fits and is given in runtime.

D. A Genetic algorithm [7]

The paper "Genetic Algorithm based on Some Heuristic Rules for Job Shop Scheduling Problem" solves their specific job-shop scheduling problem with different approaches. Heuristic rules, self developed heuristic rules, a genetic algorithm with a first population based on heuristics, and genetic algorithm with a random first population are compared. As a result, the genetic algorithm with first population based on heuristics works best, the genetic algorithm based on a random first population works second best, but significantly better than the approaches with heuristics.

The genetic algorithm starts with reading the input data. Then the first population is generated and evaluated with a fitness function. Based on that evaluation the best solution is selected. The genetic operations' crossover and mutation produce offsprings, again evaluated by a fitness function. Either the Offspring is discarded, the termination criterion is achieved and the algorithm stops or the offspring is replaced with a member of the population and added to the list sequence of new populations. This behavior repeats from the best solution selection.

III. METHODOLOGY

A. General Framework

In the given smart factory setup (figure 1) a user can order product which are handled by the knowledge manager agent, which also controls all other units as central entity. Other units include the 3d objects component with the main task to extract features of the ordered products, the motor component generating data simulating a running conveyor belt, and a cloud component holding a database and a machine learning library where the motor data is analyzed by a k-means algorithm, and the 3d objects data give information about which printer fits for execution.

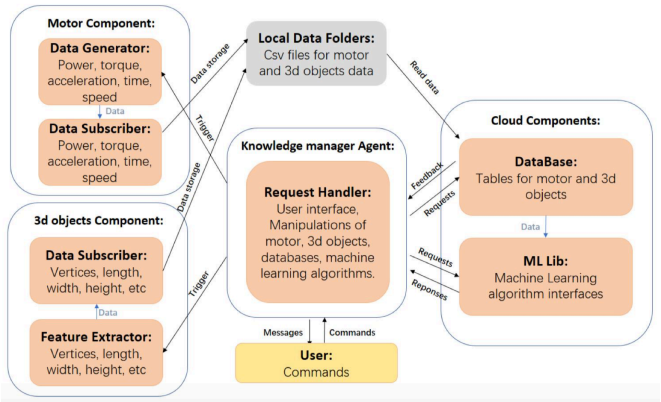

Fig. 1. Past System Setup [10]

Our system expansion includes ontologies for executing units, a super user setting the configuration data, a configuration note for system setup, an orchestration agent handling the execution units, an elaborated database, a genetic algorithm scheduling the orders, and executing units: dynamic number of printers, conveyor belts and inspection stations, setup in rows (figure 2) which not only execute the printing and transport, but also give feedback about the measured quality.

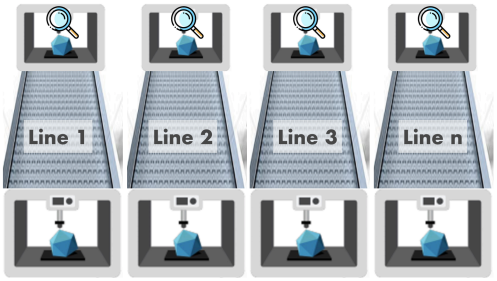

Fig. 2. Assembly lines with printer, conveyor belt and inspection station

To introduce each system component, a complete run through is explained. This is based on three general system phases: configuration, ordering, production.

Before the production starts, the factory is set up by the super user. Parameters settings are amount of production lines (printer, conveyor belts, inspection stations) and printer ontology parameters (maximum build speed, complexity speed, material-type, -color, -capacity, and quality over time behavior).

When the factory is configured and started, the standard-user starts ordering random elements. If those are new to the system, the 3d features are extracted by the 3d objects component. In the cloud, a prediction starts to forecast the quality (neural network) and makespan for each product on each printer. Based on this prediction the scheduling is done by the genetic algorithm.

Now the production can start. The production line gets its jobs from the orchestration handler, which is a part of the knowledge manager agent. The orchestration handler has control threads for the printers, conveyor belts and inspection stations which handle the ROS nodes of the executing units. In the production line, each product is printed, transported and inspected. From the inspection, the quality feedback is logged in the database.

In all steps the knowledge manager acts according to a hybrid MAPE-K model [8] and a finite state machine as a central unit to hold the knowledge and monitor, analyze, plan, and execute it via units.

All system components can be seen in figure 3, while a detailed image of the architecture and classes can be found in the appendix.

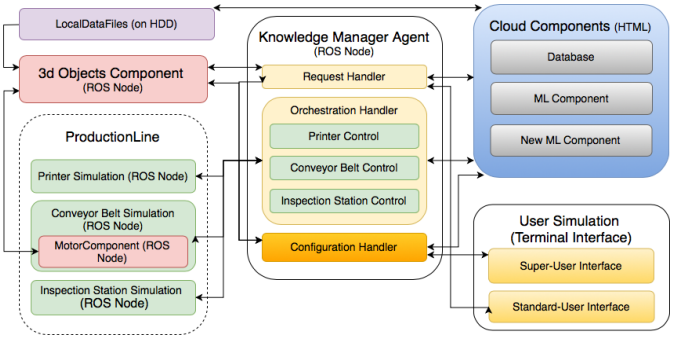

Fig. 3. System Architecture - more detailed Architecture in Appendix

B. Communication

ROS nodes "are processes that perform computation … [and] …communicate with each other by passing messages." [9] In our system the communication is between the knowledge manager and the other ROS nodes. In the following section, the communication flow is described. The flow chart representing this communication can be found in the appendix. The flow chart and this description are sorted by three central use cases: configuration, order process, and production.

Configuration

To configure the factory, the super-user logs in and has several setup options; First, option is creating a new JSON-configuration file, or the user uses an existing file, or an existing database, in that case no JSON-file is needed. If a new database is created, training for the printer-selection algorithm has to be done. All factory parameters are logged into the database. Other units like printers, conveyor belts and, inspection setup are setup with their corresponding parameters.

Ordering Process

To place an order, the user can send a new STL model file or use the existing ones. If an unknown model is sent, the 3d object data have to get extracted and stored in the database; Afterwards, the printer-selection algorithms runs to select the possible printer for the new model. Also this information is logged into the database.

Our standard-user is a simulation always ordering random models, that exist in the database. The order is logged in the database and the quality and makespan prediction run, which are triggered by the knowledge manager; Afterwards, the knowledge manager triggers the genetic algorithm to schedule the order. While the scheduling, all models are flagged as "in scheduling". The result is logged, and an internal handover from the knowledge manager to the orchestration handler takes place. All threads are set up, the main thread in the knowledge manager holds dynamic information about printers, conveyor belts and inspection units.

Production

Finally the production starts, which is handled by the orchestration handler. Models from the received scheduled list are sent to the printers in the given order; As soon as the printing of a product is requested, the "in printing" flag is set in the order backlog. The printer prints the product which is simulated by formula 2. When the product is finished, it is flagged "printed" in the order backlog and handed over to the conveyor belt in that assembly line, as soon as this is available; Again, at the conveyor belt the product is flagged "in transport", and the conveyor belts needs one to two seconds, before it is finished and flagged "transported". The orchestration handler triggers the inspection station as soon as it is available. There the quality is analyzed with formula 3 and flagged "in inspection"; When finished, the model is flagged "inspected".

If one of the flags "in printing", "printed", "in transport", "transported", "in inspection", or "inspected" is set, the model cannot be rescheduled. If a model is flagged "in scheduling" it cannot be printed, transported or inspected.

The rescheduling takes place, if new orders come in and all models that are not flagged are rescheduled as well.

C. Knowledge Manager

Included in the knowledge manager are following units: request handler, orchestration handler with printer control, conveyor belt control, inspection station control, and the configuration handler. The knowledge manager handles information exchanges from and to ROS nodes, sets components in action, handles the user interaction and database logging, the printer selection, makespan and quality prediction; Furthermore, the knowledge manager gives the signal to schedule, takes scheduled orders and ramps up the production. Within the production, the knowledge manager not only hands each product from station to station but also gets the feedback and logs it into the database; Thereby, the knowledge manager is divided into sub threads, to coordinate different tasks. The thread structure of the knowledge manager can be seen in the appendix.

D. Genetic Algorithm

Goal of the genetic algorithm is the scheduling of incoming orders by optimizing the mapping of each operation in one order to a printer. The genetic algorithm with input and output could be described as in figure 4.

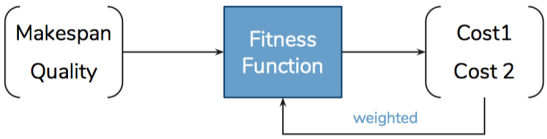

Fig. 4. Input, Output of Fitness Function

Input of the system is a matrix with quality and makespan of models for each printer. Since the scheduling takes place before the production, the values are predicted. For the makespan a formula is used for calculation, the quality is predicted with a neural network. A chromosome is a solution with a sequence of jobs. Job defines a mapping of model to printer. Gene can be seen as one of *n* jobs. Population is the set of random start chromosomes, while evolutions count the number of genetic operations. If a model is mapped to one printer, the other operations of the job (transport on conveyor belt and inspection) are executed by the printers' assembly line: printer1 - conveyor belt 1 - inspection station 1.

To prepare the genetic algorithm the job matrix is loaded. A list with forbidden items is generated, although not used yet. If the printer have maximum sizes they can print or other restrictions, the forbidden list takes those into account by recognizing if the makespan or quality cannot be predicted or is equal zero; Afterwards a list is loaded with jobs from which only one can be executed. This represents by now one model which is printed by only one printer and not several times. Based on that, the first population is created. Each chromosome randomly selects one item from each "only-one-allowed"-list without elements from the "forbidden"-list.

Based on the description of [7], the genetic algorithm starts with calculating the fitness of each chromosome of the population and selecting 40% of the strongest. Next the genes are crossed-over at a random crossover point, to combine two chromosomes. The fitness is calculated to get 40% of the worst chromosomes. For mutation, the genetic algorithm searches for the longest and shortest makespan and saves the printer names and longest order; Now, the order with the longest makespan is replaced with the shortest makespan.

Those originated offspring is evaluated to get the best chromosomes and the genetic process is repeated until the 15 evolutions are reached or if five runs in a row have approximately the same fitness.

$$FitnessFunction = w_1 * Makespan + w_2 * (1 - QualityMeasure)$$

Formula 1, Genetic Algorithm Fitness Function

The fitness function adds the makespan and overall quality of one chromosome with adjusted weights. Those weights are used for testing to see the impact of each measure. Both weights together always equal to one. The makespan is the production time of a chromosome, which is calculated by adding each printers' production times and selecting the

longest of those. By now, the quality measure calculates the product of all job-qualities; Thereby, all qualities are considered and low qualities strongly affect the result. A better and recommend method is the quality average with the standard deviation; Therefore, the results would be better comparable in the fitness function, since for the product method, the results reach a value with the power of minus 5.

E. Quality Prediction

The concept of quality prediction or regression, in our case, stemmed from a simple yet annoying problem: once an object is printed by one of the printers of the factory, it has already been printed with a certain quality percentage that will be later determined by the inspection stations; we wish, however, to be able to somehow predict that quality on each of the available printers to optimize a cost function for the genetic algorithm.

Since each printer has its own ontology with parameters that play a role in the quality of printing (formula 3), and each object has its own properties of complexity and size, the quality determined by the aforementioned inspection stations must be a function of the complexity and size of the printed object as well as the ontology of the printer that printed the object itself. This is where Deep Neural Networks (DNNs) come in. They are considered to be universal function approximations, and even if they are sometimes considered a little overkill for prediction purposes (other simpler architectures like SVMs or logistic regression can be used) they have been shown to outperform most of the regression methods for regression tasks.

The quality prediction is achieved through the use of a Feed Forward Deep Neural Network (DNN) consisting of 2 layers of 15 units (neurons) each. An overview of the data that our DNN will train and be tested on is useful to understand how and why the architecture of the Quality Prediction is the way that it is. The data can be seen in figure 5.

```
little preview of loaded data :
   Unnamed: 0  Complexity           Size   Quality0   Quality1   Quality2   Quality3
0           0    221160.0   49127.777477  61.261013  67.611974  48.559090  54.910051
1           1    126864.0  210000.000000  76.840756  80.249508  70.023253  73.432005
2           2    130104.0   24394.650469  77.233862  80.975814  69.749959  73.491910
3           3     83913.0  499906.736039  82.820313  84.609694  79.241551  81.030932
4           4     22608.0  647576.611196  92.727114  92.548137  93.085068  92.906091
```

Fig. 5. DNN training data.

Since each printer has its own ontology, each printer will have a different quality prediction even though the same object (Complexity and Size) is considered. In other words, the quality function that the DNN learns to approximate will be different from printer to printer. For *n* printers, *n* DNNs are required. Each with the architecture following in figure 6.

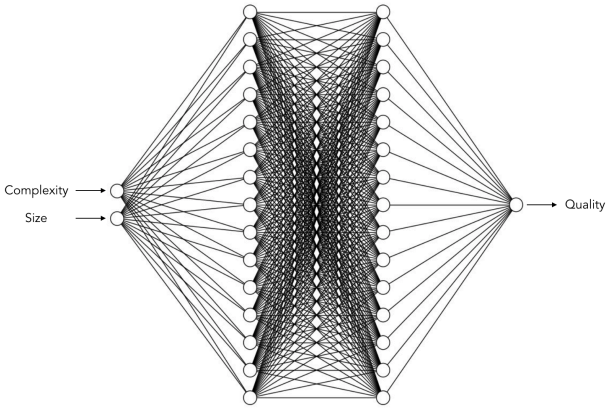

Fig. 6. DNN architecture, generated using *matplotlib*.

Writing the code for a deep neural network from scratch, would have been to tedious; therefore, the API *keras*, with the framework *theano* running as a backend, was used to build and customize our network in a modular way. To split the data, and pre-process it, *pandas* and *scikitlearn* have been used. Last but not least, *numpy* for the vector and matrix representations and *matplotlib* for graphical representations were used.

F. Production Line

The arrangement of the executing units of the production line are one printer, one conveyor belt and one inspection station in one line. When a product is printed, it does not leave the specific line (figure 2). Each unit has its own number of service threads in the terminal, depending on the number of assembly lines. Those threads permanently hand over status feedback by ROS messages.

Printer

Each printer has a dynamic status, which is permanently transmitted with the following options: ready, printing, defect, and refill. Ready means the printer is waiting for a job; printing means a product is printed at the moment; the defect state is reached when the communication to the printer is down or the printer has an error - this is not simulated yet; refill state means the printer is refilled with material.

Static parameters are the printers ID, the maximum speed in x, y, z direction, and complexity, material parameters such as color, capacity, and type, which can be "PLA" - polylactide a thermoplastic and biopolymer, or "PHA" - polyhydroxy-alkanoate a bioplastic, or "ABS" - acrylonitrile butadiene styrene a terpolymer. Important for the quality feedback is the printer alpha value; Finally, the last parameter is quality vs time which has no implemented function yet.

The printing process itself, loads order information, calculates print time and simulates printing by sleeping. The print time is dependent on the vertices and the speed for complexity, a "size-factor" which takes the printing speed for each dimension into account, and a random factor as it could be expected in a real factory. One hundredth of real time results in the simulation time.

$$PrintTime = \frac{\frac{vertices}{maximumComplexitySpeed} * sizefactor * speedfactor * random(0.8...1)}{100}$$

Formula 2, Printer Print Time Formula

Conveyor belt

The possible dynamic states of the conveyor belt are ready, running and defect. When the conveyor belt gets a product, it transports the product for a random time between one and two seconds, before going back to the ready state.

Inspection station

The inspection stations can be ready, inspecting or defect. When an inspection station inspects a product, it calculates the quality with the following formula:

$$Quality = alpha * (1 - \frac{complexity}{maxComplexity}) + (1 - alpha) * (1 - \frac{size}{maxSize})$$

Formula 3, Inspection Station Quality Formula

G. Database

The system has a SQL database in the cloud, consisting of seven lists; At the first, all model parameters are saved: the model list. Second list has all printer parameters: the printer list. The information from the printer selection algorithm are stored in the "best printer list" which name does not perfectly fit the content. In the order backlog list, scheduling informations are stored like the scheduled flags, the mapped printer with a queue number, or the reached quality. The user list holds information of the user like ID, name and password. Not used yet is the printer history list, which should hold historic information of each model. This list can be implemented and used for further training algorithms. Finally, the ProductionList was planned to hold dynamic data of the production, which was implemented into the order backlog; Therefore, the production list became obsolete.

IV. RESULTS

A. Test Setup

Despite the various tasks we had, the test setup will mainly test the genetic algorithm in regard to performance, successfulness, and impact of weighting. The orchestration with execution, feedback, and the system ontologies are used for testing and can be seen by running the system.

We chose the metrics overall fitness of each winning chromosome, makespan, quality and chromosome setup. For the genetic algorithms' performance we analyzed the runtime, the number of generations and the weighting.

For testing, we changed two variables: the amount of orders, and the weights of the costs in the genetic algorithms' fitness function. The amount of orders starts with 5 and increases to 10 and 15 orders. The weights can be between zero and one, and add to one. One weight increases by 0.2 - the other weight decreases in the same step size - in total they are one. Each setup was tested five times to have comparable results; Additionally, a weight step size of 0.1 was tested for 20 orders with one run.

Performance will be analyzed by looking at the runtime for different weightings, number of orders, and number of generations which is defined by the algorithm. We know, that the runtime is not fast, since we are running the system on a virtual machine, but it should change with the numbers of orders and not with the weighting. If the number of generations changes, this should have an impact on the runtime.

To see that the algorithm is working and giving dynamic results, the setup of chromosomes is analyzed. It is expected, that the setup is different for a run with the same parameters as number of orders and weight. Also nearly the same fitness should be reached.

The overall success is measured by looking at the overall fitness, because our fitness is not giving a real indication of the chromosome, we look at the quality and makespan individually and the overall fitness with different weights.

The accuracy of the predicted quality and makespan can be compared with the reached results.

Input range depends on the users' order. The user simulation orders randomly *n* products from 105 models; Also, the factory can be set up with *m* assembly lines.

B. Results

The runtime has a linear behavior increasing with the amount of orders, but is also slightly dependent on the setup of the test environment, and the weighting. For the amount of orders it makes sense, that the algorithm takes longer time for genetic operations and fitness calculation. Those algebraic operations can take some runtime of the computer. The number of evolutions stayed fifteen. This is a parameter that could be changed in the future, but for now it shows, that the algorithm does not reach optimization before; Therefore, we do not know, how the number of evolutions affect the runtime.

The runtime as in figure 7, is displayed in seconds for different orders with 15 evolutions. A result of 25 seconds for each run with only 5 orders is not the result, we were looking for, but the numeric result of the runtime is highly dependent on the computers' speed. This was tested in a virtual machine; therefore, the result was presumed.

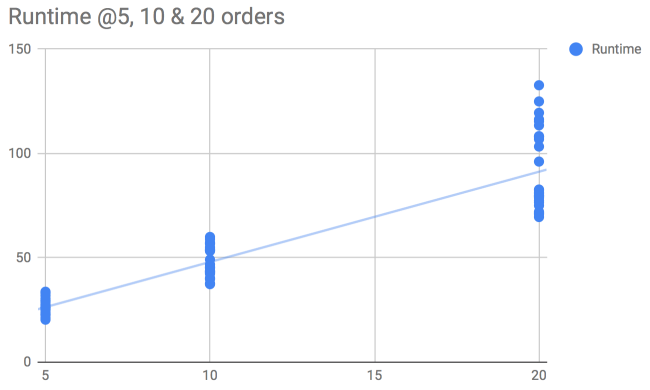

Fig 7. Runtime of genetic algorithm on a virtual machine with 5, 10 and 15 orders, each 15 evolutions, 5 times each weight. Results in 90 runs.

Weighting has also a small impact on the runtime, because of the necessity for the algorithm to perform further actions.

Setup of the chromosomes for five orders, weight 1 is 40%, tested in five runs can be seen in the following figure 8.

	Run 1		Run 2			Run3			Run 4			Run 5	
P1													
P2	O5	O2	O5			O5			O5			O5	
P3	O3		O4			O4			O4	O3	O2	O3	O4
P4	O4	O1	O1	O2	O3	O2	O1	O3	O1			O2	O1

Fig. 8. Schedule in five runs with the same parameters. Qualitative drawing - size does not correspond to the length of a job.

An interesting note is, that the makespan and quality of the best chromosome is nearly the same: makespan for all winning chromosomes is 14.664; the quality had a range of 0.008 and was between 0.099 and 0.107.

Quality and makespan are the indicators of the algorithms' success. The following diagrams show the makespan and quality cost for 5, 10, and 15 orders, with weighing from zero to one in 0.2 point steps, and 5 runs for each combination. We can see, that makespan does not significantly increase from 5 to 10 orders, but does double from 5 and 10 to 20 orders. No change between 5 and 10 orders is an unexpected result and can be explained with the figure x above. It is visible, that there are free spots left that can be used for additional orders. When 20 orders are placed, all printers have longer queues. Although when makespan weights 20% of fitness, a low result of 39 seconds is possible, but all other runs do not reach that production time.

The impact of weighting is visible for the makespan, which is best when low, because that results in a shorter production time. With increasing weight, the deviation drops, and with it the makespan itself. For 20 orders, the deviation is larger at every weight compared to the deviation for 5 and 10 orders, which is shown in figure 9.

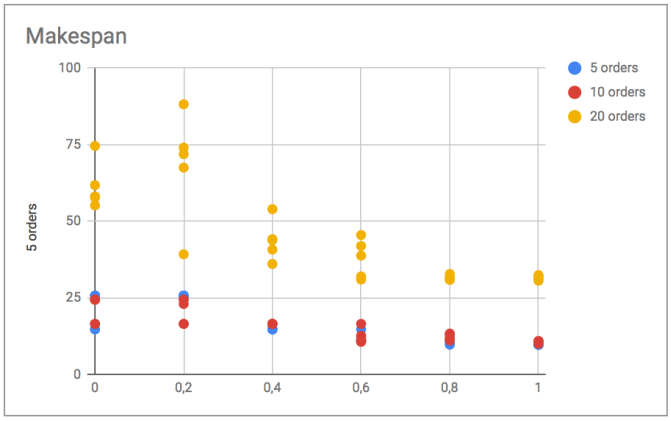

Fig. 9. Makespan, for five runs each for five, ten, twenty orders with weights ranging from zero to one.

Quality cost is difficult to measure because we multiply quality of all orders; Therefore, the result is suited for fitness and comparison to other chromosomes, but not to make a meaningful statement out of it, or compare it to results with a different number of orders. In figure 10 we scaled the quality cost of the runs with 5 and 10 orders, to find trends. In general the quality is better, when the quality is higher weighted (smaller w1 means a higher weight for quality cost). Although the best result is not achieved, when the quality is the only measure, but when the ratio is 80/20. Another lead is the high variance for runs with 20 orders. But since those numbers are scaled, it must be added, that the values before scaling where in the range of micros. The results are shown in figure 10.

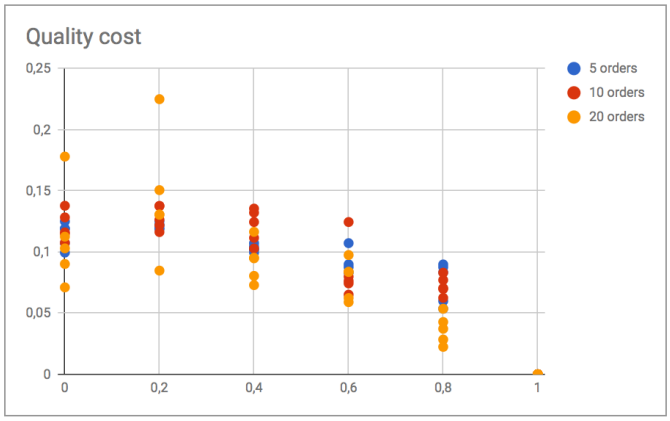

Fig. 10. Quality cost (scaled) for different weights.

To analyze the algorithms' performance, we calculated an overall fitness for 20 orders and weights from zero to one in 0.1 steps. Figure 11 is a qualitative measure in the means to comparability, but not fitness in the sense of percentile adaption to the system. Higher fitness value means higher adaptation to the system, this is reached either when the weight is balanced, or when the weight is one, which means only makespan is considered and quality is calculated out. We had several ways to cross-check this result, since there are different ways to analyze quality and makespan. Another method we chose, was the average and standard deviation of the cost factors. All methods led to the same result, that balanced weight increases the fitness and especially to get reasonable results for quality and fitness. Another outcome is that for the exact balance of 50/50 for the weights, the result has a local minimum instead of an expected maximum.

Fig. 11. Overall fitness for 20 orders with changing impact.

Accuracy for quality predictions measurements turned out to be highly difficult, because the neural network was low on training data; Therefore, we used mock data for training and testing. Since the quality prediction is learning the printer alpha values and is based on formula 3 without the random

factor, this worked out for our setup. With this environment, the error rate was lower than one percent.

C. Discussion

We have a working system, that runs relatively slow in a virtual machine. We developed a genetic algorithm, that works by combining the quality and cost of an order at an assembly line and maps orders to those line optimizing for weighted cost and makespan. When this weight is nearly balanced, (like 60/40 or 40/60) the algorithm works best.

We described our performance problems already, and stated, that it highly depends on the computer that is used. For the genetic algorithm we would recommend erasing all debugging output, if the system should be used in a real production.

The high variance for the makespan, especially for 20 orders, shows optimization potential. This could be improved by using heuristics for the first population or an elaborate termination criterion. With those improvements, the algorithm would get faster results, which happens when having start heuristics, because not as many genetic operations need to be done, to get the final fitness. For the termination criterion it would mean, that the algorithm would stop earlier, if a good fitness is reached, or no improvements can be done anymore. But an improved termination criterion could also lead to longer calculation, if a good fitness is not reached within 15 evolutions, it would not stop; therefore, the variance would decrease. The approach we used with problems and recommendations can also be found in [7].

An accuracy result of over 99% was achieved for prediction, which is not realistic. The neural network should be trained on real models, and the model set divided into 70% training data and 30% test data; therefore, we would need about 1000 models, to reach a similar accuracy. When the system is adapted to a real 3d printing assembly line, the quality formula should be reevaluated, to have realistic results. This could be done, by at first scheduling without the quality and learning from the quality results, which are logged in the order backlog. In other words, the deep neural network works well at predicting dummy data.

One main part of our system was analyzing the impact of the weight parameters in the fitness function. Those parameters should be balanced and not only for testing. This balancing should be done by the genetic algorithm itself.

The genetic algorithm maps the orders to printers based on quality and makespan and gives a chronology of these order. This chronology is random and not based on any calculation. If the chronology should be considered by the algorithm, a due date (deadline) for each order should be introduced. This would make the scheduling problem more difficult, but also closer to heuristic solutions (e.g., minimum-slack-time-rule, or minimum-due-date-rule). This could be implemented by start heuristics based on those rules for the first populations, a parameter in the fitness function, or additional lists similar to the "forbidden-list" or "only-one-allowed-list".

Measurements of precision and overall success showed that we reach reasonable makespan and quality values, but the measurement could be improved, by comparing the genetic algorithm to either an optimal solution (for 5 orders possible, for more orders inconvenient) or different heuristics, or rule approaches. Examples for this approach is given in [7]: computational results are evaluated between common heuristics, self-developed heuristics, a genetic algorithm with a first population based on a heuristic, and a genetic algorithm with a random first population.

Another starting point for improvement of overall success is the quality cost calculation. By now the quality cost is the product of all order qualities, which considers small qualities as well as high qualities. But this makes the quality cost non-comparable and not meaningful, since the cost gets smaller with every order. It is still possible to compare the qualities with each other, but it does not make a statement of the overall quality. A better approach is the average quality minus the standard deviation; Therefore, the overall quality can be determined with discordant values decreasing the result.

V. Conclusion

We built a system that orchestrates, schedules, and executes a smart 3D-printing factory. The orchestration uses ontologies and the knowledge manager, to handle requests, monitor the managed elements, analyzes the statuses, plans the next steps, and triggers all actions. A combination of a deep neural network and a genetic algorithm are doing the scheduling. The deep neural network predicts the quality of the objects to be printed on the different available printers. Our genetic algorithm schedules and maps the incoming orders to a printer-assembly line based on the predicted quality and makespan. Each printer-assembly line consists of one 3D printer, one conveyor belt and one inspection station.

With this setup, we achieved our objectives; Finally, we tested our system by analyzing the genetic algorithm in detail. Runtime and results of the genetic algorithm are properly working. The best results were achieved, when the weights of quality and makespan were nearly balanced. Further recommendations to improve the algorithm are: erasing the debugging output, compare to different heuristics, build first population on heuristics, and change the calculation of quality costs.

We noticed, that for the genetic algorithm, the neural network and the simulation a fast PC is needed and using a virtual machine leads to long simulation and test time; Therefore, running the system in a virtual machine is slow, but the best solution for a fast setup. Since, the only fully running system was in a virtual machine, we needed long time for evaluation and debugging. Also, errors in used libraries and the framework were hard to find.

Enhancements for the smart factory are: addition of graphical simulation of the production line, inclusion of k-means motor analyzation with conveyor belt units, maximum dimension for printers to create a more realistic environment, product due date to enhance the genetic algorithm, involvement of quality/time parameter, and adaption of the select-printer algorithm to the environment with several

printers; Finally, the printing history could be used to build better learning algorithms.

REFERENCES

[1] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng und T. Darrell, DeCAF: "A Deep Convolutional Activation Feature for Generic Visual Recognition", Berkeley, CA, USA: UC Berkeley & ICSI, 2014.

[2] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone und A. Beghi, "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach", IEEE Transactions on Industrial Informatics (Volume: 11, Issue: 3), 2015.

[3] L. Kavrak, P. Svestka, J.-C. Latombe und M. Overmars, "Probabilistic roadmaps for path planning in highdimensional configuration spaces", IEEE Transactions on Robotics and Automation (Volume: 12, Issue: 4), 1996.

[4] Bundesministerium für Bildung und Forschung, "Digitale Wirtschaft und Gesellschaft - Industrie 4.0", [Online]. Available: https://www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html. [Accessed: 29 May 2018].

[5] C. Akpolat, D. Sahinel, F. Sivrikaya, G. Lehmann und S. Albayrak, "CHARIOT: An IoT Middleware for the Integration of Heterogeneous Entities in an Urban Smart Factory", in s Federated Conference on Computer Science and Information Systems, Prague, Czech Republic, 2017.

[6] J. Blazewicz, W. Domschke und E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques", European journal of operational research, Bd. 93, Nr. 1, pp. 1-33, 1996.

[7] Boushaala, Amer A., Mohamed A. Shouman, and Somaia Esheem. "Genetic Algorithm based on Some Heuristic Rules for Job Shop Scheduling Problem." *Proc. of the International Conference on Industrial Engineering and Operations Management, July*. 2012.

[8] Kephart, Jeffrey O., and David M. Chess. "The vision of autonomic computing." *Computer* 1 (2003): 41-50.

[9] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3. No. 3.2. 2009.

[10] Stefan Klemencic, Peng Qian, Ruan Chongwei, "Big Data in Industrial IoT (IIoT)" Applications of Robotics and Autonomous Systems, February 2018, TU Berlin.

APPENDIX

System Architecture

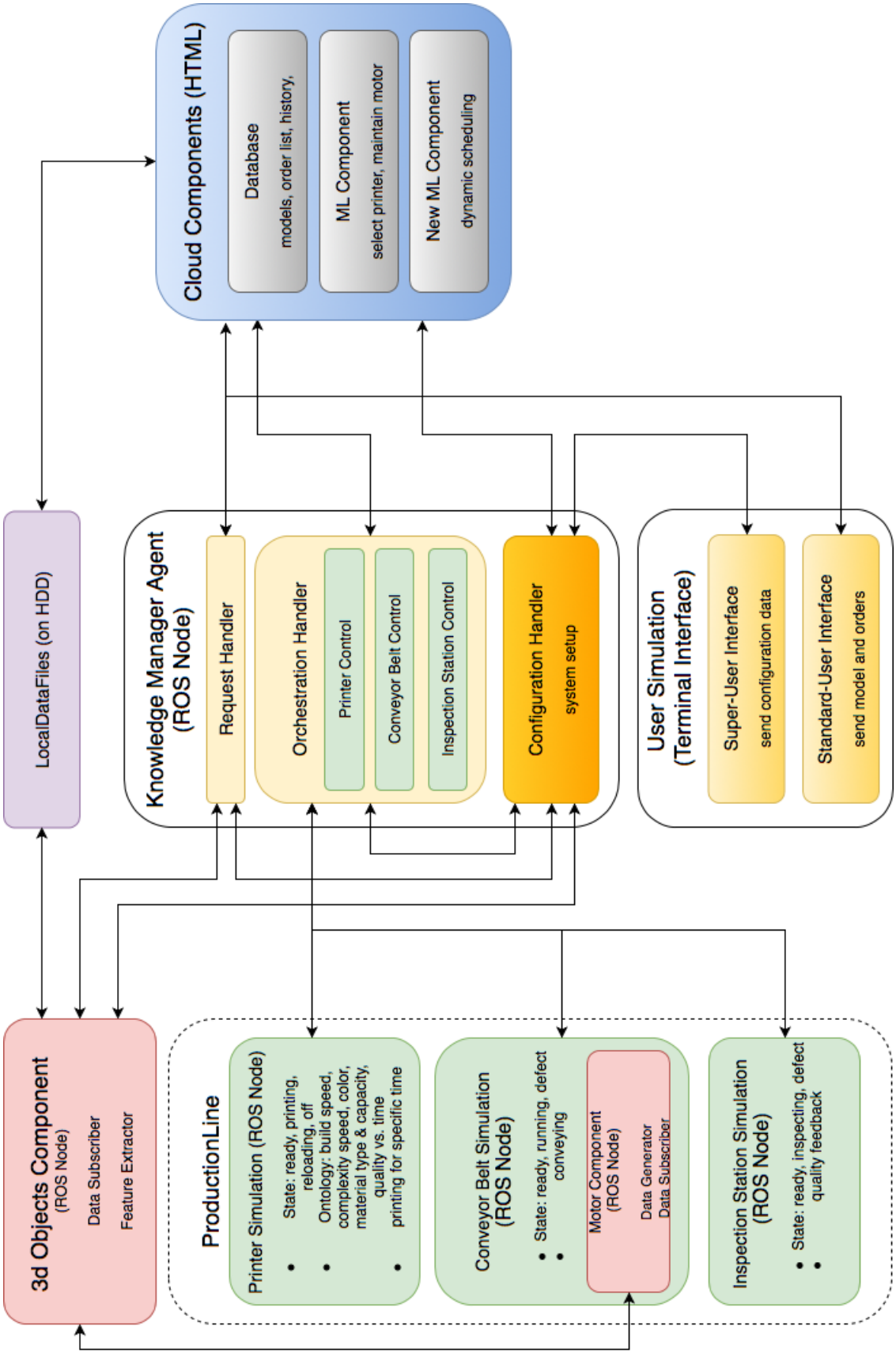

Fig. 12. Detailed system architecture

Class diagram

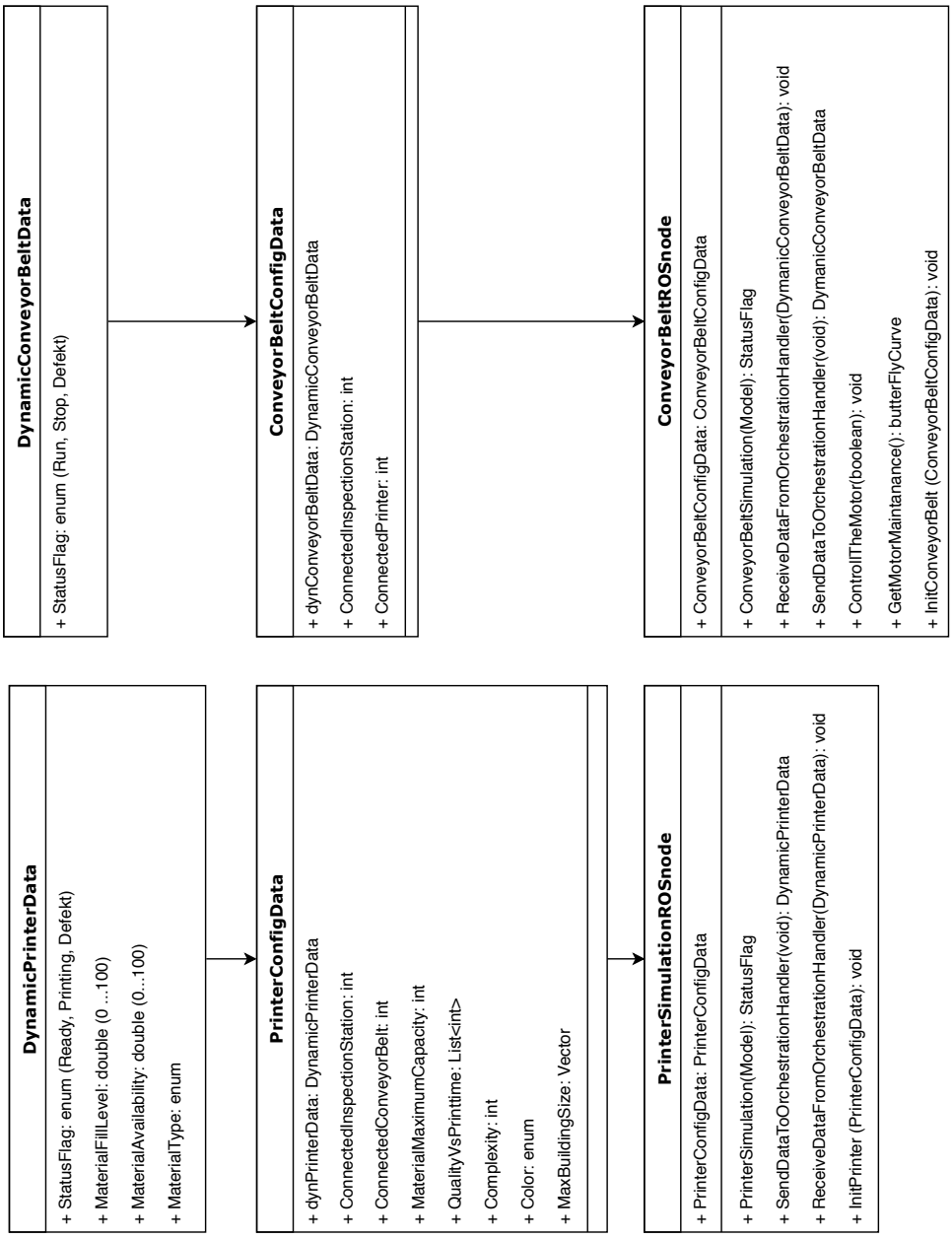

Fig. 13. Class diagram.

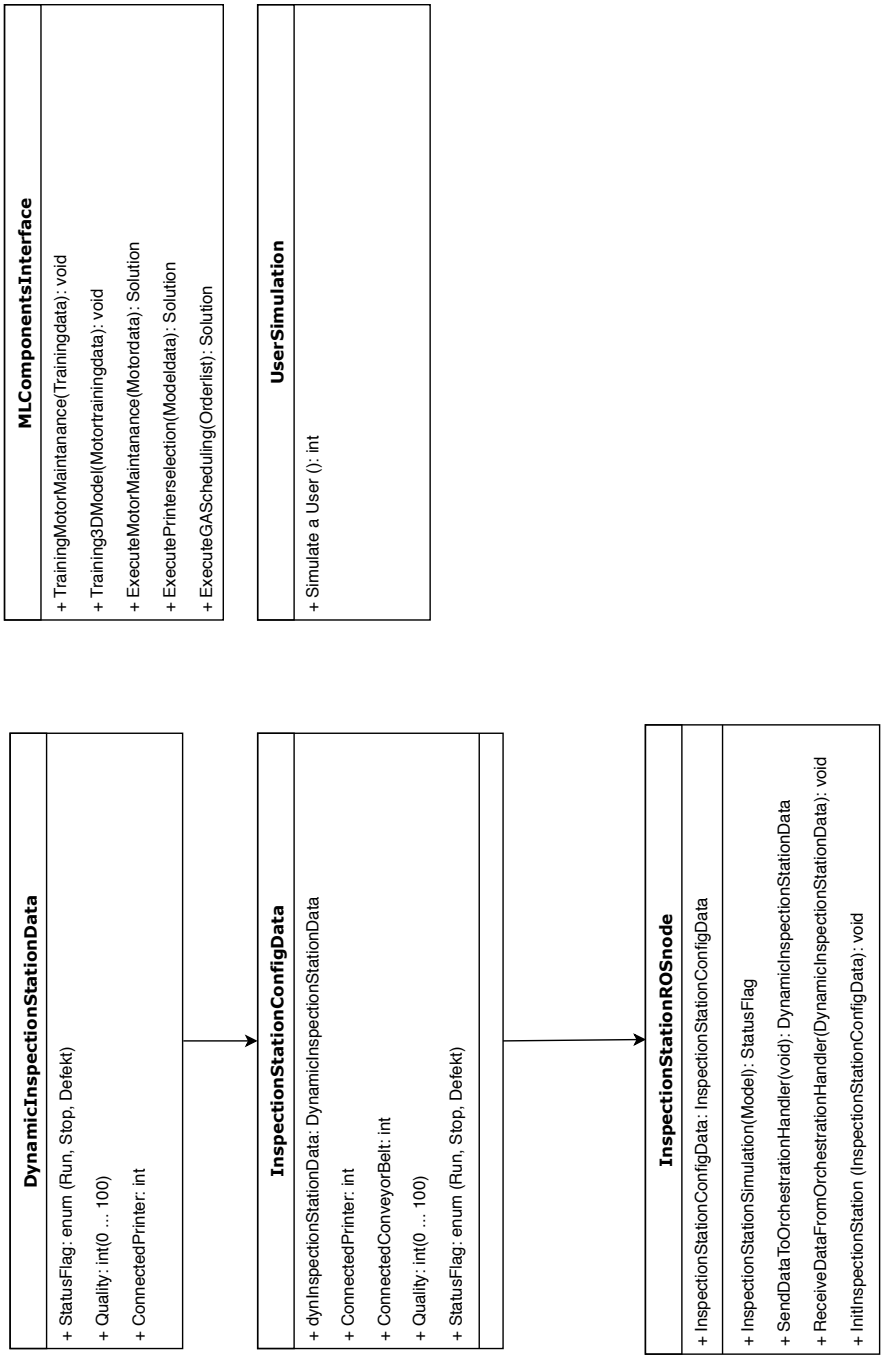

Fig. 14. Class Diagram

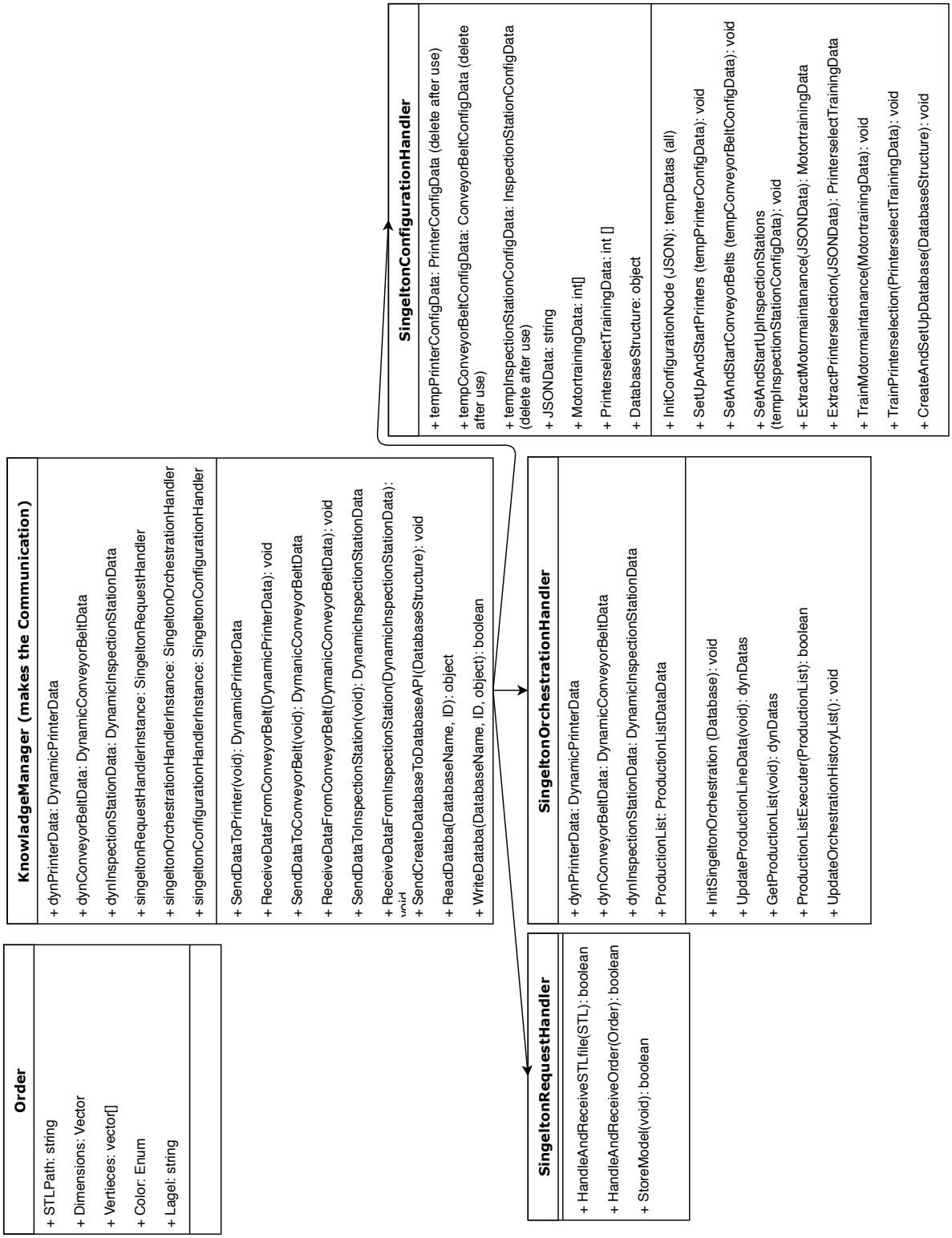

Fig. 15. Class Diagram

Database structure

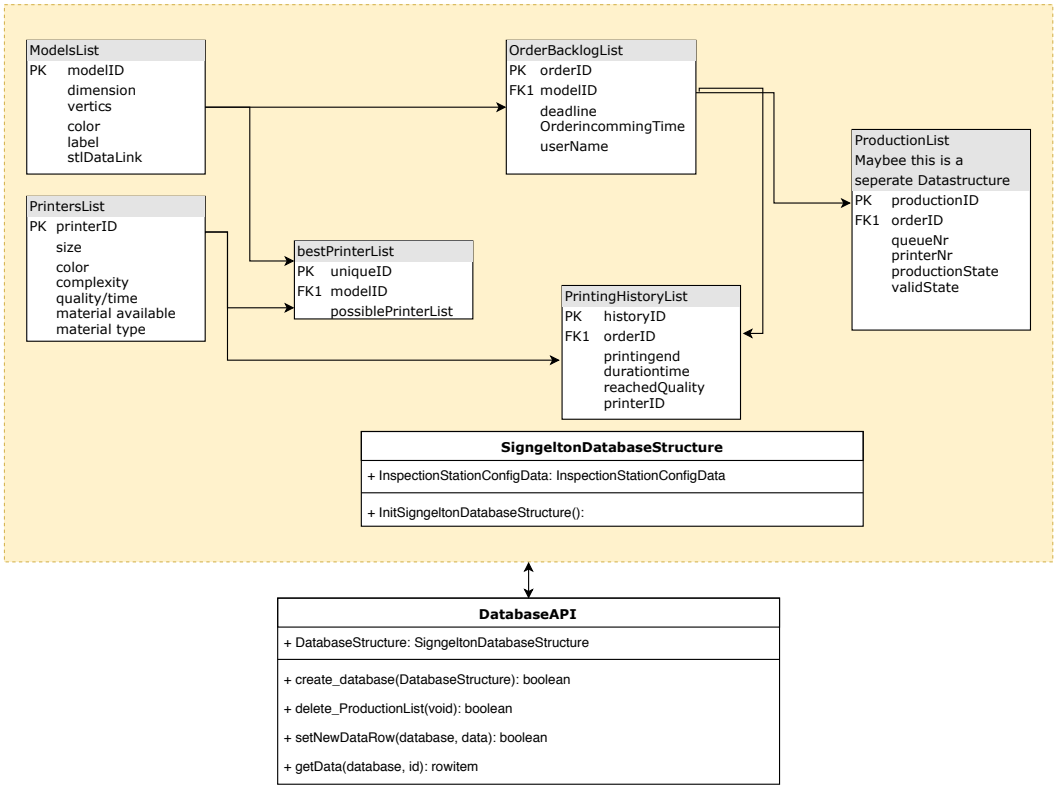

Fig. 16. Database structure

Sequence diagram

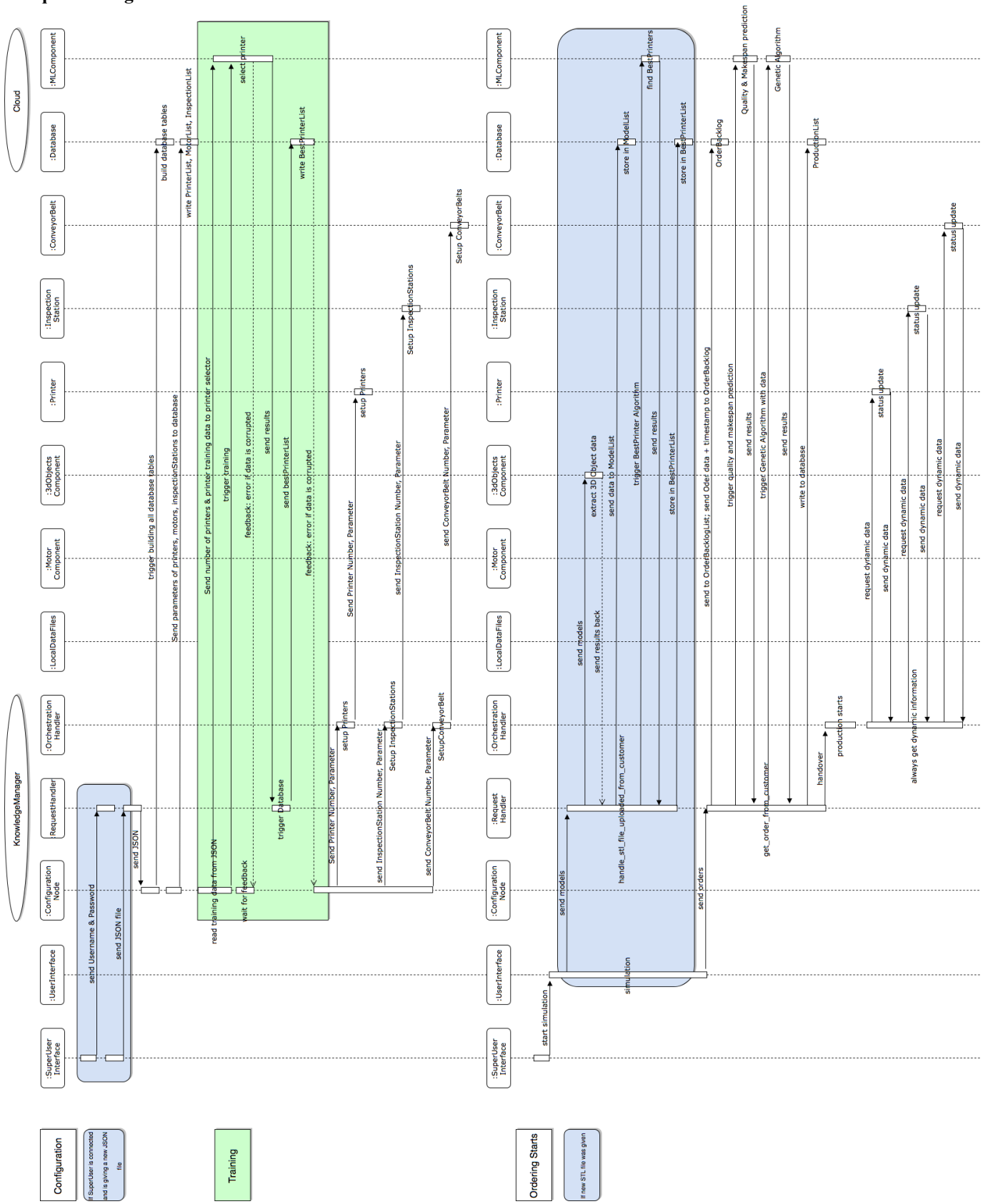

Fig. 17. Sequence diagram: Configuration and ordering process.

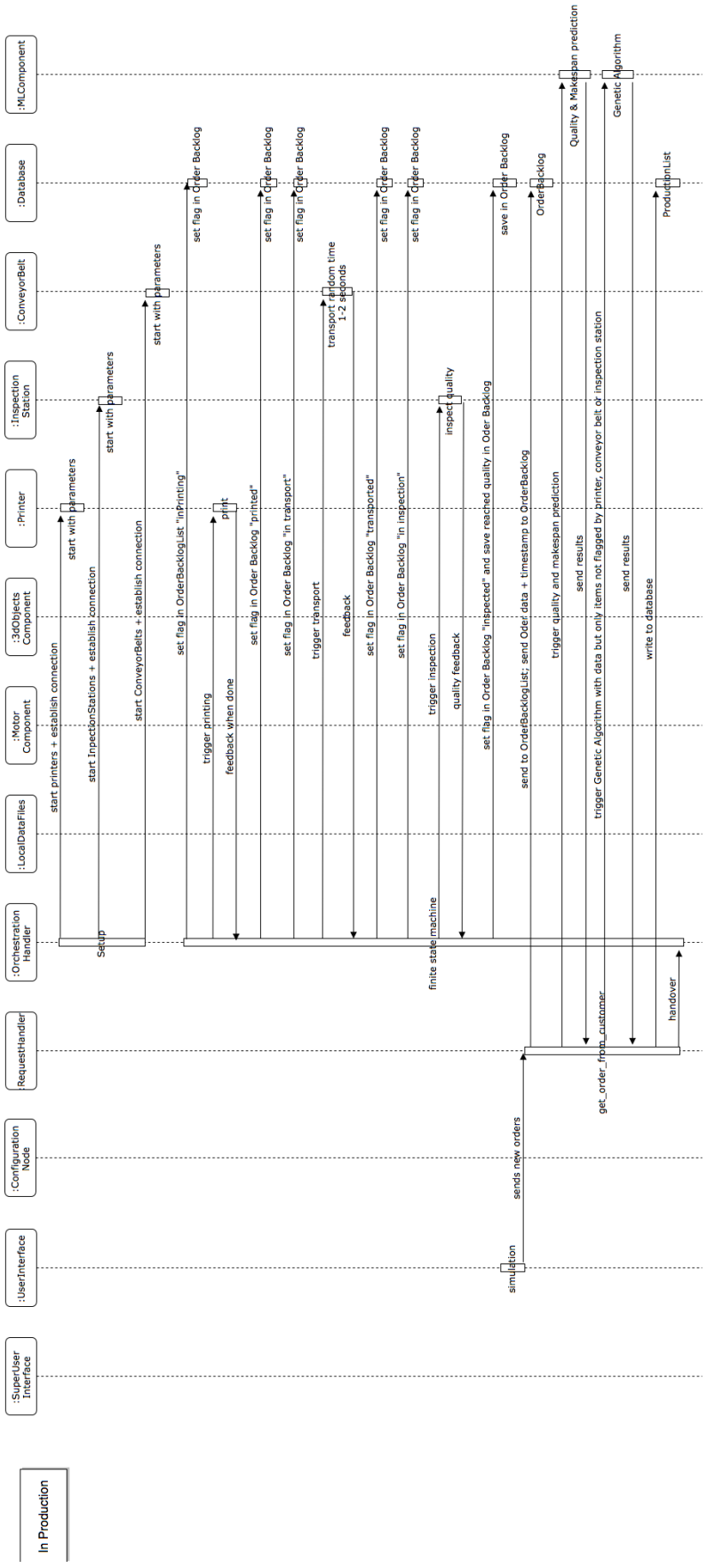

Fig. 18. Sequence diagram: Production process.

Thread Structure

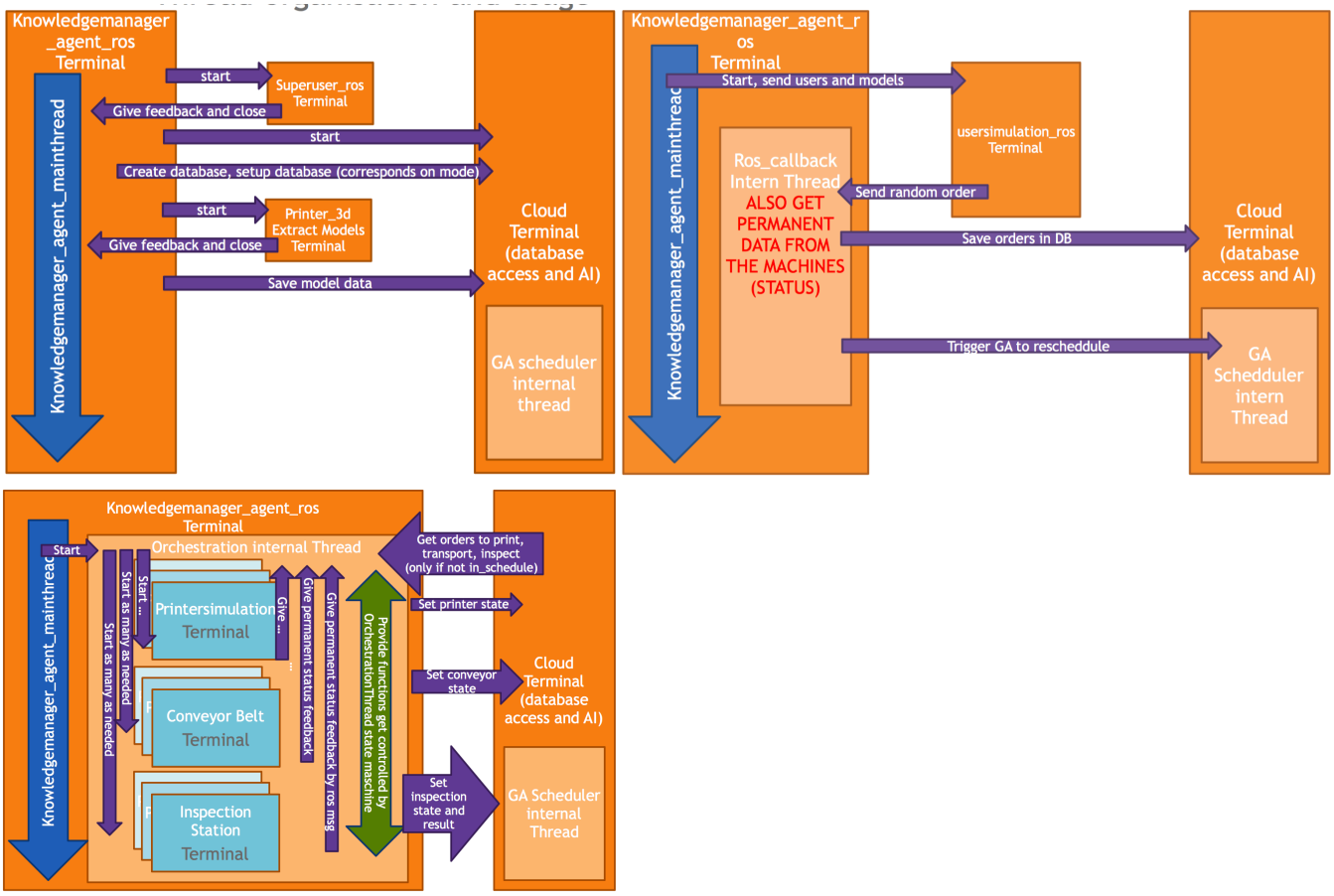

Fig. 19, 20, 21. Thread structure while in production

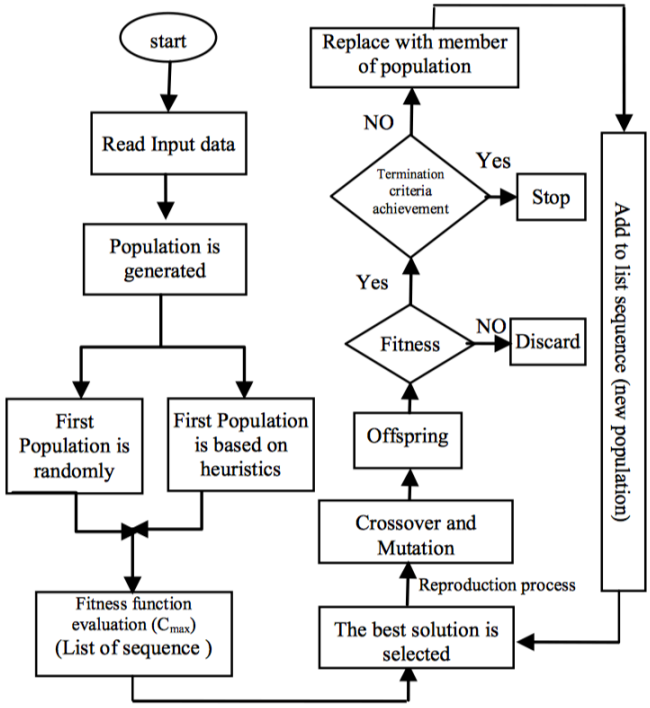

Fig. 22. Proposed algorithm flow chart by [7]