Proceedings of the Business and Industry Simulation Symposium (ASTC 1999).
April 2000. Washington D.C. [SCS]. ISBN 1-56555-199-0

PRODUCTION SCHEDULING OF COMPLEX JOBS WITH SIMULATION.

Jaap A. Ottjes and Hans P.M. Veeke,
Sub Faculty of Mechanical Engineering and Marine Technology, Fac. OCP
Delft University of Technology
Mekelweg 2, 2628 CD Delft, the Netherlands
e-mail: J.A.Ottjes@wbmt.tudelft.nl , H.P.M.Veeke@wbmt.tudelft.nl

Key words: manufacturing , job shop, project planning, intelligent agents, discrete simulation.

ABSTRACT

A simulation approach is presented for planning and scheduling a flow of complex jobs for job shop like production systems. Machines may have their own specific restrictions and properties such as relative production speed, set up characteristics and scheduling rules. A production job consists of a set of tasks represented by a directed activity network in which an activity is defined as a single task to be processed on a machine. As a consequence a machine may also be an assembly station. The task duration may be stochastic having any probability distribution. The task flow and the task selection for scheduling is governed by agents: Each machine group acts as an agent combining tasks and machines. Each task acts as an agent guarding the proper network sequences of its job by determining when it is ready for releasing and, after that, controlling its own scheduling priority. A job acts as an agent repeatedly updating critical path analysis for its tasks. For that purpose a separate critical path simulation model is used. Crucial for the modelling the "process approach" used. The model is generic with respect to job shop configuration e.g. number of machine groups and number of machines per group and also with respect to the job configuration.

INTRODUCTION

Job shop production planning and scheduling is generally considered to be complex . This is caused by the large variation of job composition, the unpredictability of the job stream and stochastic execution times of tasks. Much work has been done in this field important issues being to minimise the time span for a well known set of jobs to be executed (Chu, 1998), (Pezzella 2000) or to improve scheduling by batching (Potts, 2000). In (Sikora, 1997), a multi-agent framework is presented in which agents are used for ensuring the orderly operations and concerted decision making among components of the manufacturing system.

In this work we use straight forward simulation for planning and scheduling a stream of jobs which may have stochastic execution times.

The Factory

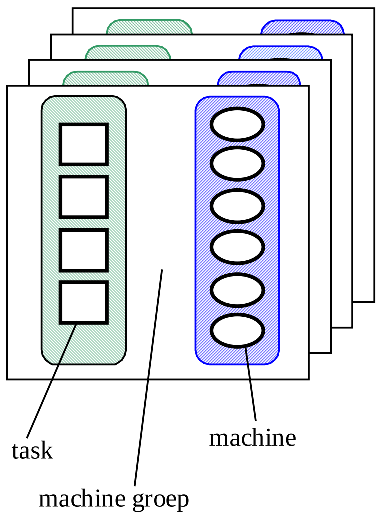

Figure 1. the factory, consisting of a number of machine groups each with a set of tasks to be executed and a set of machines or work stations.

The factory consists of an number of machine groups each containing a set of machines with equal function. The individual machines however may differ in specifications such as relative speed and set up times. Every machine group owns a set containing the released tasks to be done in this group.

The Jobs

A production job consists of a set of tasks: the taskSet. A task represents an operation to be executed on a machine of a specific machine group. The sequence of tasks of one job is represented by a directed activity network. That implies that a task can also be an assembly operation. A task owns an execution time which may be stochastic. The realisation of an execution time of the task on a specific machine is obtained by drawing the execution time from the execution time distribution multiplied with the speed factor of the machine and adding the current set up time required. Transportation tasks are to be modelled by introducing a machine group with "transport machines" and incorporating the transportation tasks into the task network structure.

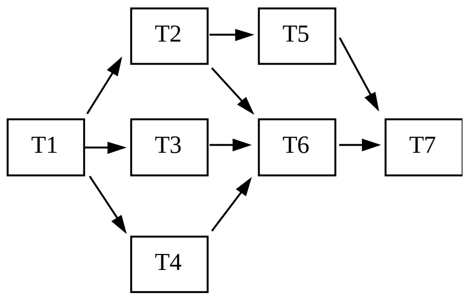

Figure 2. Example of the task structure of a job. It is modelled as a directed activity network.

Approach

The approach may be characterised as simulation based planning and scheduling using agents. Machine groups, machines , jobs and tasks are equipped with relatively simple processes and methods to assure the right task sequences and to accomplish completion times as close as possible to pre set due times. These due times are supposed to be known from a global planning for example using an ERP application. Essential for the easy straight forward modelling is the use of process oriented modelling. The process oriented modelling was first used in the program language "Simula". after that the technique was used in several simulation packages. Especially the object oriented approach appears to be suitable for this process simulation. (Healy, 1997), (Robert, 1998). The model presented here is being implemented in the simulation package Tomas (Tool for Object-oriented Modelling And Simulation), based on Delphi (Veeke, 1999), the description of which is given in another paper proposed for this conference. It provides a simulation class called "TomasElement" with all the necessary simulation features. Simulation components are descendants of this class. Further Tomas allows distributed modelling. First we will go into some important aspects of process oriented modelling and define pseudo language for important simulation commands.

PROCESS ORIENTED MODELLING

The job shop model will be explained in terms of a process oriented model in pseudo code. The process oriented approach used in this work can be summarised into two steps:

<u>Step 1:</u> decompose the system into relevant classes of components, preferably patterned after the real world components of the system. A class is characterised by its attributes. Attributes may be of any data type. The state of each instance of a class is defined by the state of its attributes. An instance of a class will be called a component.

<u>Step 2:</u> distinguish the “living” component classes and provide their process description. A process description governs the dynamic behaviour of each instance of the component class.

A process defines the dynamic behaviour of a component. In the discrete event two types of activities are distinguished: Activities which consume no system time, for example the determination of the next task to be processed, and activities consuming system time, for example the actual processing of a task. In the process description we use “***hold***” t to indicate that a component needs t time units to carry out an activity. If such a hold statement is encountered in the process description, the process halts until time t has passed and then continuos its process. In other words the process is waiting for a specific time event. Analogue to that it is possible for a process to wait for a state event e.g. for a specific condition to be fulfilled. In pseudo code this is written as: ***"standby"*** while/until condition. Another time consuming statement in a process description is

"passivate" meaning that a component becomes passive when this statement is encountered. A passive component can only be (re)started be a ***"(re)activate"*** command given from another process.

Because there may be several components active at the same time a sequencing mechanism is necessary to synchronise the activities and to manage the event calendar. This mechanism is to be supported by the simulation package used.

Additional features are queues or sets which may contain components and, in case of stochastic behaviour, distributions, modelling for example inter arrival times or execution times. Queues, sets as well as distributions may be used as attributes of component classes. Different models, running at the same time on one or more computers, may communicate via messages. This can be done synchrone in case the models use the same time base or asynchone if model use different times. The latter will be used in the communication between the job shop model and the critical path planning model. The interaction between two models is characterised by two commands: ***sendMessage*** and ***waitMessage.*** If during model execution a component calls *sendMessage*, then all models receive the message and decide if they are addressed. If a model is addressed it checks all of its components whose last action was a *waitMessage* and takes proper action. In next paragraph the job shop and its dynamics are described in terms of a process oriented model.

MODELLING

In the job shop model the actual scheduling and processing of the jobs is simulated.

Table 1: Classes of the job shop model

class	process
machineGroup	determine task-machine combination and start machine
machine	execute tasks
job	monitor and arrange critical path planning of job.
Task	take care of timely release and own priority

The job shop model uses a separate simulation model taking care of the critical path analysis of jobs. It will be referred to as the CP model.

In the job shop model the component classes are defined in table 1.

In the job shop model the current time is called shopTime

The CP model has its own system time called *planTime*, starting with 0 for each planning session. The classe are summarised in table 2.

Table 2: Classes of the critical path model

Class	process
PlanAgent	control critical path analysis for a job.
PlanTask	take care of timely release and simulate own execution

Both models communicate via messages. Next all classes of both models will be elaborated in terms of class descriptions with attributes and process descriptions in pseudo code. Cyclic processes have a repeat command at the end meaning that they have to restart their process from the beginning. If attributes need to be qualified by their owner this is done by dot notation, for example: taskToProcess.execTime means execTime of taskToProcess. Comments are put between braces {}.

Machine Group

The machines are grouped in machine groups. All machines belonging to the group are contained in the machineSet which is an attribute of a machine group. Idle machines also belong to another set: the machineFreeSet. Each machine group owns a taskToDoSet in which all tasks to be scheduled are contained and a reference to the next machine to start. A machine group owns a process in which the assigning of tasks to machines is controlled. This may be done in various ways. An obvious way is to combine task in the taskToDoSet with the highest priority and the most suitable machine in the machineFreeSet. In the example below the machine with the smallest set up time is chosen. In this context the highest priority is mutually determined by the tasks themselves using their critical path data (see task description). In general any decision rule may be implemented

<u>machineGroup</u>

- machineSet
- machineFreeSet
- tasksToDoSet
- nextTask
- machineToStart
- PROCESS

<u>process of a machineGroup</u>

- *standby* while machineFreeSet is empty OR tasksToDoSet is empty
- call selfArrange of last task in tasksToDoset {tasks rearranges themselves according priority criterion}
- nextTask=first task in tasksToDoSet
- machineToStart = machine in machineFreeSet with smallest set-up time with respect to nextTask
- machineToStart.taskToProcess = nextTask
- remove nextTask from tasksToDoSet
- *reactivate* machineToStart
- repeat

Machine

A machine has a reference to its machine group and an reference to the task to be processed. Further all kinds of specific attributes might be attached such as a factor giving the relative production speed of the machine compared with the other machines of the group and set up information. A machine owns a process simulating the physical operation of the its tasks. This machine process is quite simple: after being activated for the first time, it enters the machineFreeSet and waits (is passive) until it receives its next taskToProcess and is activated by its machine group. It leaves the machineFreeSet, completes the assigned task and repeats its process by re-entering the machineFreeSet of its group. The execTime of the task may be composed of several parts such as set-up time and actual execution time. These times may be drawn from a distribution or obtained from a data base or input file with realistic manufacturing data. In case the model is used for real time scheduling the simulated machine may be substituted by a real one. Its execution time then is the real execution time of the real machine. Using simulation in this way is similar to the approach discussed in a multi-AGV control system. (Ottjes, 1996).

<u>machine</u>

- group
- taskToProcess
- machine specific attributes
- PROCESS

<u>process of a machine</u>

- enter group.machineFreeSet
- *passivate*
- leave group.machineFreeSet
- *hold* taskToProcess.execTime
- *reactivate* taskToProcess
- repeat

Job

A job to be processed on the job shop is modelled as an activity network, the activities being called “tasks”, see also figure 2. The tasks are present in the taskSet of the job. Further the job owns its due date and its earliestReadyDate. The latter is determined by a critical path analysis of the job assuming infinite production capacity and average task execution times. This analysis is controlled by the process of the job and repeatedly performed. For each planning session the job process provides the current states of the remaining task. The remaining execTimes of tasks, which are already in process on a machine, are estimated by interpolation. The replanningCriterion of the job may be periodic or trigged by the job state for example its tardiness.
The job process communicates with the critical path planning model, which itself is a simulation model with its own time base using messages. In terms of sequencing this means that the job shop model 'waits' until the *waitMessage* command has been completed. In the mean time the CP model performs the planning of the job and returns the results to the job process. The job then updates its ealiestReadyDate and all earlyStart and latestFinish data of the tasks which are still waiting for execution.

<u>job</u>

- taskSet
- dueDate
- earliestReadyDate
- replanningCriterion
- PROCESS

<u>process of a job</u>

- *standby* while replanningCriterion = false
- write shopTime and data of job and its remaining tasks to file ‘x’
- *sendMessage* ‘x’ to planAgent {pass the file name 'x'}
- *waitMessage* {wait for ready signal of planAgent}

- read updated job data and task data and update job and its tasks
- repeat

Task

A task has a reference to the job it belong to and a reference to the machine group it has to be processed on. Further it owns an execution time which may be stochastic. The task network structure is defined using two sets: the preTaskSet and the afterTaskSet. The preTaskSet contains all directly preceding tasks. A task only may be released if all tasks in its preTaskSet are ready. So task6 (T6) in figure 2 only may release itself if all tasks (T2, T3 and T4) in its preTaskSet are ready. The afterTaskSet contains directly succeeding tasks. In other words all tasks which are waiting for the completion of this task. If task 2 (T2) in figure 2 is ready T5 and T6, being the tasks in his afterTaskSet, may be released as far as T2 is concerned. T6 however has to wait for completion of T3 and T4 too. In this way any directed activity network can be represented. Each task owns a process which guards its proper sequencing and a procedure called selfArrange which is used to take the right position in the tasksToDoSet of its machine group. Further a task has a planned earliest start, latest finish and free float from the last critical path analysis performed on its job. In order to determine its production priority a priority function is added. As an example this function may return the sum of the jobslack + the freeFloat of the task. A very interesting way of defining mutual task priorities is using a fuzzy criteria as described by [Sakawa, 2000].

task

- job
- preTaskSet
- afterTaskSet
- execTime
- execGroup
- earliestStart
- latestFinish
- freeFloat {latestFinish-earliestStart- average execTime}
- function prioCriterion
- procedure selfArrange {take own position in tasksToDoSet according to prioCriterion}
- PROCESS

process of a task

- *standby* while preTaskSet is not empty
- enter tasksToDoSet of execGroup
- *passivate*
- leave all preTaskSets of the tasks in afterTaskSet
- terminate

Procedure selfArrange of task

- while prioCriterion < prioCriterion of predecessor in tasksToDoSet then change places
- if not first in tasksToDoSet call selfArrange of predecessor in taskToDoSet
- exit

PlanAgent and PlanTask

The next two classes form the CP model. It consists of a planAgent controlling the planning and a class planTask being very similar to the task class in the job shop model.

planAgent

- activeTaskSet
- PROCESS

The planAgent initiates and controls the planning run with the CP model.

process of the planAgent

- *waitMessage* {communication with jobs of the job shop model}
- read shopTime and data of job and its tasks from file 'x'
- create planTasks (+ attributes)
- put all planTasks into the activeTaskSet
- *activate* all planTasks
- *standby* until activeTaskSet is empty
- earliestReadyDate=shopTime+*planTime*
- reset *planTime* =0
- put all tasks back into the activeTaskSet
- *reactivate* all planTasks
- *standby* until activeTaskSet is empty
- write new job and task data to file 'x'
- *sendMessage* 'x'
- repeat

The planTask, like in the job shop model, waits until it may be released. Then, in stead of waiting for execution by a machine, it executes itself using its average execTime. If all planTasks are finished (activeTaskset = empty), the make span and the early start times are known. Then the planning process is performed vice versa, in other words the job is broken down. This gives the latest finish data for the tasks.

<u>**planTask**</u>
- preTaskSet
- afterTaskSet
- tempTaskSet
- averageExecTime
- earliestStart
- latestFinish
- PROCESS

<u>**process of a planTask**</u>
- put all tasks of **pre**TaskSet into tempTaskSet
- *standby* until tempTaskSet is empty
- earliestStart=shopTime+*planTime*
- *hold* averageExectime
- leave all tempTaskSets of the tasks in afterTaskSet
- leave activeTaskset
- *passivate*
- {now reverse simulation: break down the project}
- put all tasks of **after**TaskSet into tempTaskSet
- *standby* until tempTaskSet is empty
- latestFinish = earliestReadyDate-*planTime*
- *hold* averageExectime
- leave all TempTaskSets of tasks in preTaskSet
- leave activeTaskset
- terminate

CONCLUSIONS AND FURTHER WORK

A process oriented structure has been proposed for modelling job shop production of jobs which are represented as a directed activity network of tasks. Tasks govern their own release and priority claims, acting like agents. Each job guards its own progress and initiates, if necessary, a critical path planning action for its remaining tasks. For this purpose a planning model is proposed also based on simulation. Machine group agents assign machines to tasks and machines perform the physical task execution. The model is generic with respect to machine configuration and job composition. A machine may be an assembly station. Further the model is flexible with respect to specific machine properties, scheduling rules and stochastic task execution times. The model is being implemented to be used for off line developing and testing planning and scheduling rules and for on line controlling actual production.

REFERENCES

Chu, C. J.M. Proth and M. Wang. 1998 "Improving job shop schedules through critical pairwise exchanges." *International Journal of Production research* 36 (1998) 383-694.

Healy, J. R.A. Kilgore 1997. "Silk,: A Java-Based Process Simulation Language". *Proceedings of the 1997 Winter Simulation Conference*, IEEE,

Ottjes, J.A., F.P.A. Hogedoorn; "Design and control of multi-AGV systems: reuse of simulation software." *Proceedings of 8th European simulation symposium. Society for Computer Simulation Internationa Genoa* 1996, p. 461-465. ISBN: 1-56555-099-4

Pezzella, F and E. Merelli. 2000. "A tabu search method guided by shifting bottleneck for the job shop scheduling problem." *European Journal of Operational research* 120 (2000) 297-310.
Piscataway NJ

Potts, C and M.Y. Kovalyov. 2000. "Scheduling with batching: A review." *European Journal of Operational research* 120 (2000) 228-249.

Robert, C.A., Dessouky, M. 1998. "An Overview of Object-Oriented Simulation". *Simulation* vol:70:6, pp. 359-368. 1998.

Sakawa, M and R. Kubota. 2000 "Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms." *European Journal of Operational Research* 120 (2000) 393-407

Sikora, T.T. , M. Shaw. 1997. "Coordination Mechanisms for Multi-Agent Information Systems: Applications to Integrated Manufacturing Scheduling." *IEEE Transactions on Engineering Management*, Vol 44, (May 1997)

Veeke, H.P.M. and Ottjes, J.A. 1999. Problem oriented modelling and simulation, *Proc. 1999 summer computersimulation conference (SCS'99*) Chicago, Illinois pp.110-114, ISBN 1-56555-173-7