



UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Simulation-Based Online Scheduling of a Make-to-Order Job Shop

David Krige

Student Number: 14056186



*Thesis presented in partial fulfilment of the requirements
for the degree of Master of Science of Industrial
Engineering*

At

Stellenbosch University

Study Leader: Mr. J. Bekker

December 2008

DECLARATION

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 2 October 2008

Copyright © 2008 Stellenbosch University

All rights reserved.

ABSTRACT

Scheduling is a core activity in the manufacturing business. It assists with efficient and effective utilization of capital-intensive resources and increased throughput, thus increasing profitability. The focus in this thesis is on scheduling of manufacturing orders in a make-to-order job-shop enterprise. It is widely accepted that manufacturing of large volumes and production with as few as possible product variants is the most cost-effective business approach, but the need for low volume, once-off engineering parts will always exist.

Many approaches to scheduling exist, including translation of a scheduling problem to a Travelling Salesman analogue, while Discrete-event computer simulation is well established as a means to assist with scheduling. Simulation is appealing in the manufacturing environment, as it can realistically imitate dynamic, stochastic processes while being descriptive in forecasting the future. In this thesis, the development and testing of a simulation-based scheduler is described. The scheduler was developed for, and in collaboration with a South African make-to-order job-shop enterprise. A supporting information system was also developed and it is required that the enterprise changes some of its business processes if this scheduler is implemented.

The scheduler considers the status of the enterprise each time a new order is received, and the current schedule is reviewed and may be revised at such a point in time, making it a real-time scheduler. Several classic scheduling dispatching rules and –measures were incorporated in the scheduler. These include First-in First-out, Earliest Due Date, Longest Processing Time, Shortest Processing Time, Smallest Slack and Critical Ratio (dispatching rules), while the performance measures are Makespan, Earliness, Lateness, Average Flow Time and Machine Usage.

The proposed scheduler has been verified and validated using test data and designed confidence building tests, and its performance was also compared to an actual, historical schedule. The functioning of the scheduler is finally demonstrated using a stochastic test environment. The scheduler has generally performed satisfactorily and should be implemented as the final phase of this project.

Skedulering is een van die kardinale aspekte in 'n vervaardigingsonderneming. Dit kan verseker dat kapitaal-intensiewe bates effektief gebruik word om die deurset van die onderneming te verhoog en sodoende die winsmarge te vergroot. Hierdie tesis fokus op die skedulering in 'n vervaardigingsonderneming wat klein hoeveelhede onderdele volgens bestellings vervaardig. Dit is 'n welbekende feit dat die vervaardiging van groot hoeveelhede onderdele, en van klein verskeidenheid, die mees koste-effektiewe benadering is, maar die behoefte aan klein hoeveelhede spesiaal-ontwerpte onderdele sal altyd bestaan.

Daar bestaan reeds 'n groot verskeidenheid skeduleringstegnieke, wat die omskakeling van die skeduleringsprobleem na die bekende "Travelling Salesman" probleem insluit. Diskreet-gebeurtenis rekenaarsimulasie is 'n bekende tegniek wat skedulering ondersteun. Simulasie is 'n aantreklike tegniek in die vervaardigingsektor aangesien dit dinamiese en stogastiese prosesse realisties kan naboots en oor die vermoë beskik om aanvaarbare vooruitskattings te doen. Die ontwikkeling en toetsing van 'n skeduleerder wat gebaseer is op simulasie word beskryf in hierdie tesis. Die skeduleerder was ontwikkel vir 'n Suid Afrikaanse onderneming wat 'n vervaardig-volgens-bestelling werkswinkel bedryf. 'n Inligtingstelsel is ook ontwikkel om die skeduleerder te ondersteun. Daar word verwag dat die besigheid van sy besigheidsprosesse verander indien die skeduleringstelsel geïmplementeer sou word.

Elke keer wanneer 'n nuwe bestelling ontvang word, ondersoek die skeduleerder die huidige status van die besigheid, asook die huidige skedule en kan op daardie oomblik in tyd 'n nuwe skedule ontwikkel, wat dit 'n intydse-skeduleerder maak. Klassieke skeduleringsreëls soos kortste vervaardigingstyd, langste vervaardigingstyd, vroegste sperdatum, eerste-in eerste-uit, kleinste surplus en kritiese verhouding is ingesluit in die skeduleerder, terwyl modeluitsette soos vervaardigingstydperk, vroegheid, laatheid, gemiddelde vloeytyd en masjienbenutting dien as prestasiemaatstawwe.

Die voorgestelde skeduleerder is geverifieer en gevalideer deur middel van geskepte data en toetse wat ontwikkel is om vertrouwe in die skeduleerder op te bou. Die skeduleerder is ook getoets met werklike geskiedkundige data om die skeduleerder se uitkomst te vergelyk met 'n werklike skedule. Die funksionaliteit van die skeduleerder is finaal gedemonstreer in 'n stogastiese toetsomgewing. Oor die algemeen het die skeduleerder goed presteer en moet geïmplementeer word as die finale fase van hierdie studie.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to the following people and organisations whose contributions made this work possible:

- Mr. J. Bekker for his boundless guidance, support, devotion and patience.
- Dr. F.K. Krige for his financial support throughout my studies.
- Department of Industrial Engineering of the University of Stellenbosch for the bursary that made my post-graduate studies possible.
- Daliff Engineering for their willingness to participate and assist in this study.

TERMS OF REFERENCE

The work in this thesis originated from a need in the Department of Industrial Engineering at Stellenbosch University to establish a simulation-based scheduler for research purposes. A particular requirement was that the scheduling be done online, which implied an application environment in which the status changes frequently. It was thus stated that, because manufacturing is a cornerstone of industrial engineering, it is preferred that the application be done in a manufacturing environment, and in particular in a manufacturing job shop.

The specific assignment was to develop a *platform* for an *online scheduling mechanism* for a *manufacturing process* using *discrete event simulation* in combination with an *information system*. The mechanism must continuously monitor the current status of the manufacturing job shop. Using the information about the current status, the scheduling mechanism must suggest a schedule that could be followed to achieve the organizational objectives best, using typical scheduling rules and -performance criteria. Figure 1 illustrates the assignment in detail.

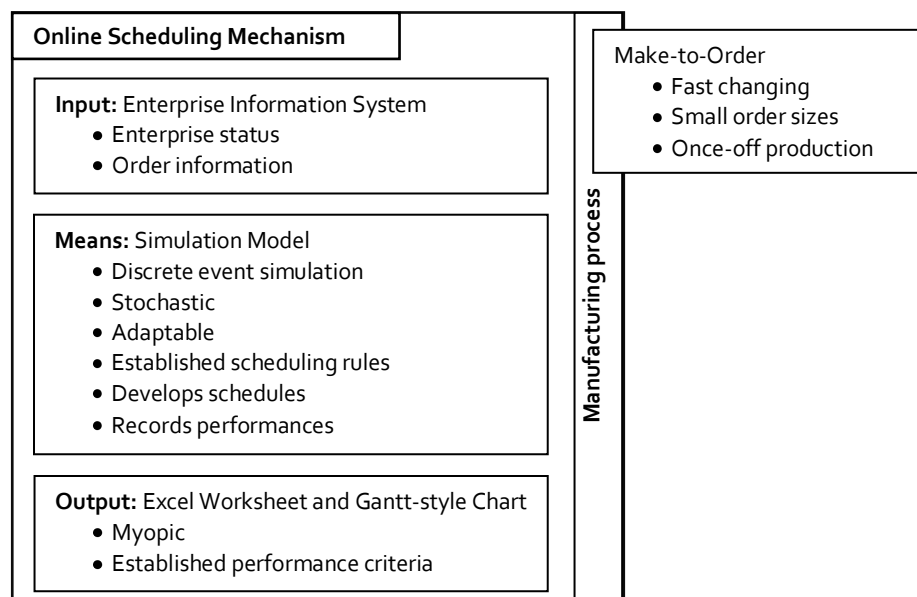


Figure 1 Outline of assignment

An information system that manages information about the current enterprise status and orders had to be developed to provide input for the scheduling mechanism. This information should be used by a simulation model of a job shop that adapts itself to the current shop floor status. Having assumed that status, it must develop different schedules by implementing established scheduling rules while simulating the processing of waiting

orders. The simulation model must imitate the flow of orders through the stochastic processes of the enterprise when compiling these schedules. The simulation model must evaluate the performance of each scheduling rule and write the schedules and their performances to an Excel worksheet. The schedule with the best myopic performance, measured by established performance criteria, must be determined and a resulting Gantt-style chart should be developed for the schedule, as well as a detailed task list containing expected starting and finishing times per part of an order.

CONTENTS

Declaration.....	I
Abstract.....	II
Opsomming	III
Acknowledgements.....	IV
Terms of Reference	V
Contents.....	VII
List of Figures.....	IX
List of Tables	XIII
Notation and Abbreviations	XIV
1. Introduction	1
2. Problem Statement	3
3. Literature Review.....	8
4. Technical Description of the Participating Enterprise: Daliff Engineering	40
5. Architecture of the Proposed Scheduling Mechanism.....	44
6. Information System Design and Implementation.....	48
7. Description of the Simulation Model.....	62
8. Validation and Verification of the Scheduling Mechanism Functionality	80
9. Evaluation of the Scheduling Mechanism with Real-World Data.....	100
10. The Scheduling Mechanism in a Stochastic Environment	112
11. Conclusion	135
12. Further Developments on the Scheduling Mechanism Identified	139
13. References.....	142

Appendix I.	Data Dictionary	147
Appendix II.	Information System ASP Code	150
Appendix III.	Explanation of Subroutines in VBA Code	179
Appendix IV.	Simulation Model Code	187

LIST OF FIGURES

Figure 1 Outline of assignment	V
Figure 2 Thesis Road Map	2
Figure 3 Graphical illustration of frequent planning.....	5
Figure 4 Generic FMS (Banks [5])	10
Figure 5 Type of shop according to production environment	12
Figure 6 Simple Job-shop problem with 4 jobs (Sadeh [12])	14
Figure 7 P class belongs to NP class.....	16
Figure 8 Illustration of NP-hard	17
Figure 9 Illustration of NP-complete	17
Figure 10 Job-shop problem presented as travelling salesman problem.....	19
Figure 11 Branch-and-Bound example	20
Figure 12 Simulation in perspective.....	34
Figure 13 Schematic for the on-line planning/control process using real-time simulation (Banks [5])	36
Figure 14 General framework and functional elements of an on-line simulation model (Drake and Smith [56]).....	38
Figure 15 Order processing at Daliff Engineering	41
Figure 16 Top level architecture of the scheduler	45
Figure 17 Simulation model architecture	47
Figure 18 ERD of the information system.....	49
Figure 19 Relation Data Structure of the Information System	51
Figure 20 Relational Data Structure containing associative entities	51
Figure 21 DFD for quote generation process	53

Figure 22 DFD for add new customer process	53
Figure 23 DFD for quote status change process.....	54
Figure 24 DFD for report process	54
Figure 25 DFD for add material process	55
Figure 26 Child diagram for the quote generation process	56
Figure 27 Homepage screenshot	57
Figure 28 Add material screenshot.....	57
Figure 29 Generate quote screenshot	58
Figure 30 Assign part information screenshot	59
Figure 31 Adding operations to an order screenshot	59
Figure 32 Add new customer screenshot.....	60
Figure 33 Update quote to an order screenshot.....	60
Figure 34 Order structure	63
Figure 35 Entity structure	64
Figure 36 Diagram of entity creation and attribute assignment component of the simulation model	69
Figure 37 Diagram of machine station and queuing component of the simulation model.....	70
Figure 38 Diagram of the part assembly section of the simulation model	71
Figure 39 Diagram of the statistic recording section of the simulation model	72
Figure 40 Model logic structure	72
Figure 41 Example of a schedule that is developed in the output file.....	76
Figure 42 Typical average flow time comparison bar chart.....	77
Figure 43 Typical average total lateness comparison bar chart	77

Figure 44 Typical average total makespan comparison bar chart	78
Figure 45 Typical average total earliness comparison bar chart.....	78
Figure 46 Typical usage comparison bar chart	78
Figure 47 <i>Scenario 1A</i> : Resulting Schedules	84
Figure 48 <i>Scenario 1B</i> : Resulting Schedules.....	86
Figure 49 <i>Scenario 2A</i> : Resulting Schedules	89
Figure 50 <i>Scenario 2B</i> : Resulting Schedules.....	91
Figure 51 <i>Scenario 3A</i> : Resulting Schedules	94
Figure 52 <i>Scenario 3B</i> : Resulting schedules	98
Figure 53 The Evaluation Methodology.....	101
Figure 54 Actual schedule followed.....	106
Figure 55 Proposed schedule for the FIFO rule	106
Figure 56 Proposed schedule for the SPT rule	107
Figure 57 Proposed schedule for the LPT rule.....	108
Figure 58 Proposed schedule for the EDD rule	108
Figure 59 Proposed schedule for the SS rule	109
Figure 60 Proposed schedule for the CR rule	109
Figure 61 Proposed schedule for the Mixed rules	110
Figure 62 Uniform distribution	114
Figure 63 Triangular distribution	115
Figure 64 Processing times input	116
Figure 65 Random number generation by means of the inverse transform	117
Figure 66 Time-line of order arrival.....	120

Figure 67 Different schedule performances of Order set 1.....	121
Figure 68 Proposed FIFO schedule for order set 1	122
Figure 69 Different schedule performances of Order set 2	124
Figure 70 Proposed CR schedule for order set 2.....	125
Figure 71 Different schedule performances of Order set 3.....	127
Figure 72 Proposed SPT schedule for order set 3.....	128
Figure 73 Different schedule performances of Order set 4.....	130
Figure 74 Proposed CR schedule for order set 4	130
Figure 75 Different schedule performances of Order set 5.....	132
Figure 76 Proposed CR schedule for order set 5.....	133

LIST OF TABLES

Table 1 Milling Machines	42
Table 2 Turning machines	42
Table 3 Fields of the Input Record	63
Table 4 Description of Entity Attributes	65
Table 5 Structure of recorded entity information	75
Table 6 Structure of recorded part information.....	76
Table 7 Order Composition for Validation	81
Table 8 Processing times for <i>Scenario 3A</i>	92
Table 9 Comparison of schedules.....	104
Table 10 Order set 1	120
Table 11 Order set 1 after 12.5 hours	122
Table 12 Order set 2	123
Table 13 Order set 3	126
Table 14 Order set 4.....	129
Table 15 Order set 5	131

NOTATION AND ABBREVIATIONS

The notation and abbreviations that are used throughout this thesis are stated in this chapter. The notation scheme used is shown first.

The subscripts used are as follows, the object they refer to is stated next to them:

- j refers to a job or part
- i refers to an operation
- k refers to a machine

The following notations are associated with job j :

- Operation ($O_{i,j}$) – operation i of job/part j
- Processing time ($p_{i,j,k}$) – processing time of operation i of job/part j on machine k
- Part due date (D_j) – the promised date that delivery of job/part j would be made to the customer
- Operation due date ($d_{i,j}$) – the due date of operation i of job/part j
- Release date (r_j) – the date on which job/part j joins the system

The abbreviations used and their meanings are as follows:

- FIFO – First in first out, the operations are processed in the order they enter the system
- SPT – Shortest processing time, the operation that has the shortest processing time is processed first
- LPT – Longest processing time, the operation that has the longest processing time is processed first
- EDD – Earliest due date, the operation with the earliest due date is processed first
- EST – Earliest start time, the operation with the earliest start time is processed first
- SS – Smallest slack, the operation with the smallest time difference between finishing its predicted processing time and its due date is processed first
- CR – Critical ratio, the operation that has the biggest remaining processing hours to hours to due date ratio is processed first
- TSP – Travelling Salesman Problem
- JSP – Job-Shop Scheduling Problem

1. INTRODUCTION

The road map of this thesis is illustrated in Figure 2 and consists of five phases. In each phase certain topics are discussed, which form the chapters of this thesis. The road map is included at the start of each chapter with the applicable topic emphasized on the diagram, thus providing context to the reader.

In the first phase, the research problem is stated, which in essence is to develop a scheduler that could assist a make-to-order job-shop with real-time scheduling. A literature survey on manufacturing shops and their scheduling is also included to better understand scheduling in make-to-order job-shops.

The second phase comprises the conceptual design, which includes a description of the participating local enterprise and the design of the proposed architecture for the scheduling mechanism. The architecture is the backbone of the scheduling mechanism developed in this thesis.

Phase three contains the detail design of the scheduling mechanism that was developed according to the proposed architecture. This phase includes the design and implementation of the enterprise information system that serves as input for the simulation-based scheduler. The information system enables the user to enter order information, and saves the information in a database from which the simulation model is run. The design and implementation of the simulation model that initiate the scheduling are also included in this phase. The simulation model automatically configures itself according to the production and shop floor status through the use of Visual Basic for Applications code.

The next phase conducts the testing of the scheduling mechanism through validation and verification. The confidence building tests that were designed to test the scheduler are included and described in this phase.

The final phase of this thesis comprises a conclusion, stating the evaluation process of the scheduling mechanism and its results. Historic data was used to compare the resulting schedules to the actual historic schedule. The functionality of the scheduling mechanism in a stochastic environment is also discussed. A conclusion is drawn from the results of the evaluation process and future work that can be done to enhance the current scheduling mechanism is stated.

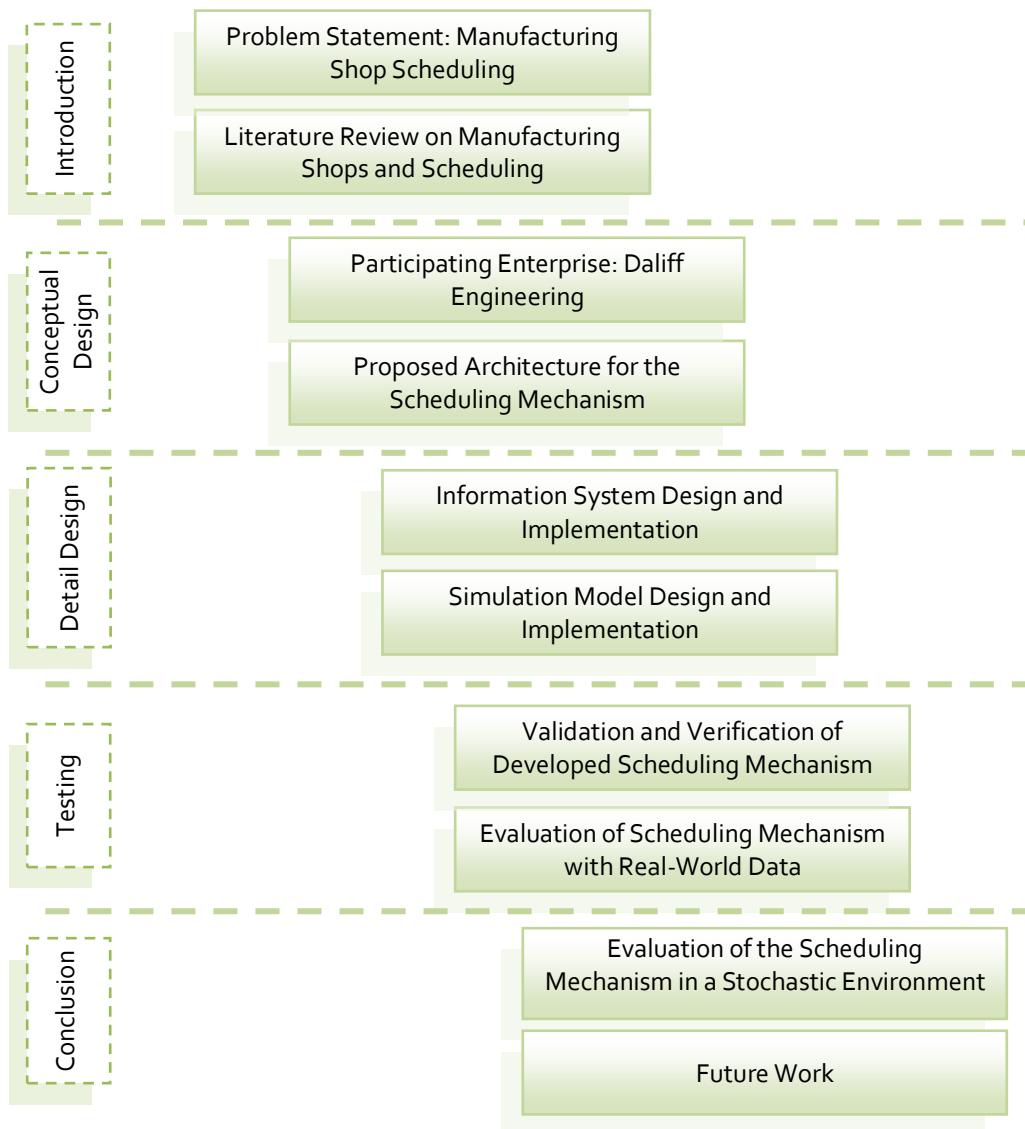
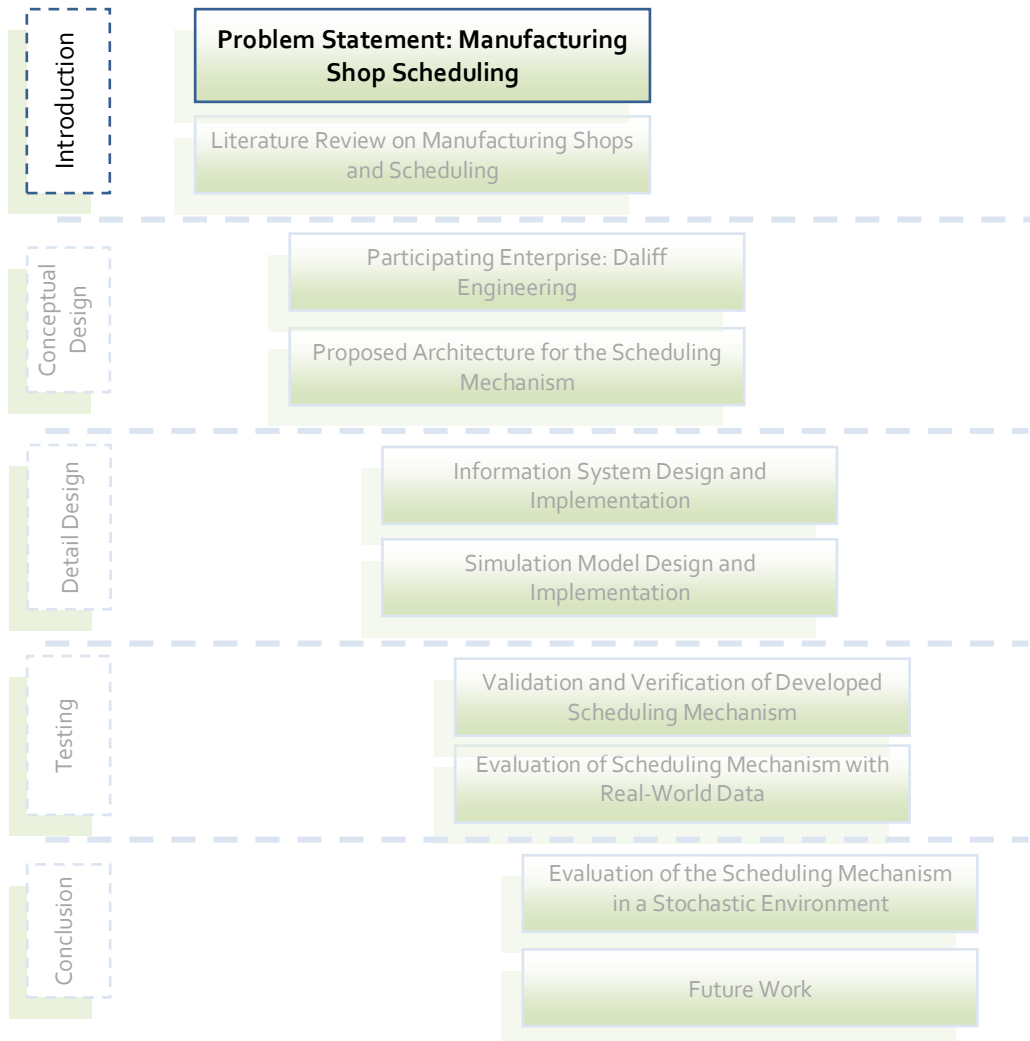


Figure 2 Thesis Road Map

2. PROBLEM STATEMENT

This study originated, as most research studies do, from a problem. Finding the solution to the problem or optimizing existing solutions is the drive force behind any research. This chapter states the problem behind this study and the context of the chapter is shown below.



Scheduling of manufacturing shops has been a highly researched area as it contains aspects that still needs to be resolved or optimized and plays an important role to the success of a manufacturing shop. Several different types of manufacturing shops exist and therefore also sub research sections. Before any research into manufacturing shop scheduling can be commenced, the particular environment of the manufacturing shop that will participate in this study needs to be stated, to be able to scope the research effort.

Daliff Engineering is a manufacturing shop that produces custom designed parts according to customer orders. Orders arrive at a high frequency and mostly require the production of small quantities of parts. Parts are usually once-off designs that are seldom reproduced in the future. The processing times are fairly predictable since CNC machines are used in production.

It is an unpredictable production environment as the future of order arrivals is unknown, while the chances of system disturbances are also high as machines fail and employees get sick. Determining a schedule in such a dynamic environment is challenging as it has to happen regularly and is very time consuming each time. The efficient scheduling of a manufacturing shop is of cardinal importance to the success of the shop, not only in terms of market gains, but also in service record and image. Scheduling therefore needs to be taken very seriously if a manufacturing shop wants to become a benchmark in their specialization field.

Daliff Engineering understands the importance of scheduling and was therefore willing to participate in this study that strives to develop the best scheduling mechanism for their manufacturing shop. Whilst doing this, the study also aims to add a functional methodology to the research area of manufacturing shop scheduling.

The scheduling mechanism should be able to sufficiently predict the future to generate a schedule that is practical, efficient and easily followed. Simulation is suggested as prediction model. The scheduling mechanism should use the enterprise information system as an input for its scheduling component. The information system should be of such nature that the current system state is correctly presented to enable the scheduling mechanism to imitate the manufacturing shop sufficiently.

As a result of the dynamic nature of the manufacturing shop under investigation, the scheduling function should have a frequent planning capability. This characteristic is seen as the most important element to achieve the best schedule for the manufacturing shop with its high order arrival rate, therefore it will be discussed in detail.

2.1 FREQUENT PLANNING CAPABILITY

Frequent planning can be described using Figure 3, which illustrates frequent planning on a single machine only. A red star represents an instance where a new order arrives in the

system, a circle represents an operation and a yellow star an instance where a snap shot of the current state is needed as input for the scheduling function. The operations are colour-coded to enable the reader to distinguish which operations are related to which part, and also where the new operations of a new order are scheduled. This will become clear as the figure is described further.

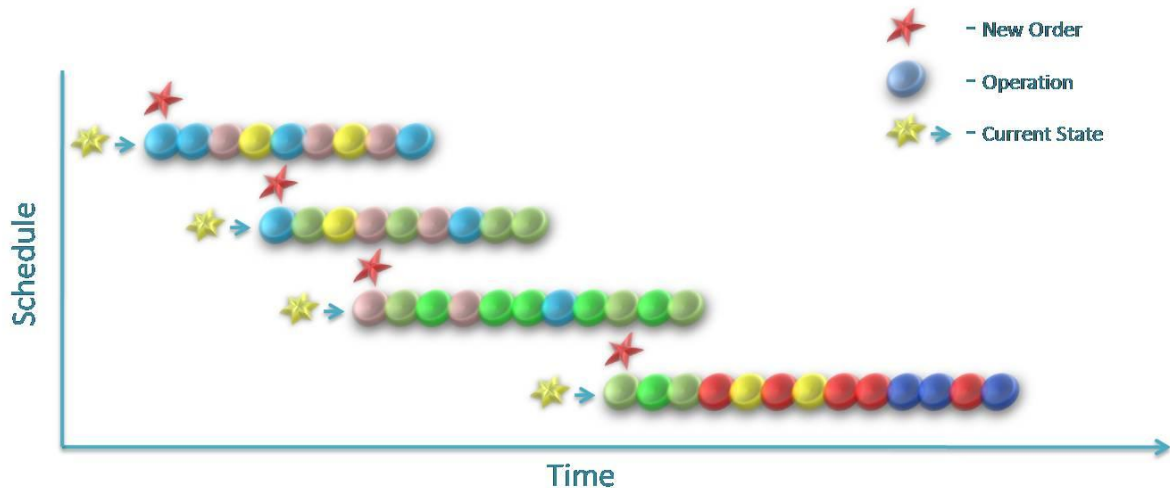


Figure 3 Graphical illustration of frequent planning

When a new order arrives in the system, a schedule must be compiled that incorporates the manufacturing of the operations related to the new order whilst also scheduling the operations of orders that were already in the system. Some of the operations of the orders that were already in the system, could by the time the new order arrives in the system already be processed. These completed operations of the orders already in the system must be excluded from the new schedule. It is also possible that processing on some of the orders already in the system is currently under way as the new order joins the system. It is important to schedule these operations to be processed first in the new schedule, as they are already being processed and pre-emption is not allowed, i.e. an operation already started can not be stopped and rescheduled for production at another time. These situations imply that a snap shot of the current system is needed as an input when scheduling is commenced.

For the purpose of explanation, the current state of the system is assumed to be empty when the first order arrives, represented by the first red star in time. The first order has three different parts, with their operations each distinguished by different colours, for example, the part that is represented by blue has four operations.

When the new schedule has been generated, it is implemented and production follows the sequence of operations suggested by the schedule as time elapse. At any given point in time, another new order can arrive in the system, resulting in the need to determine a new schedule. This is illustrated by the second red star, which indicates the instance of a new order arriving in the system at a certain point in time. The second yellow star represents the instance of the current system state used as input for the new schedule. At the particular point in time when the new order arrived in the system, the first four operations of the previous schedule have been completed. These four operations must thus not be included in the next schedule, for example one of the four operations that were finished is a yellow operation and there is only one yellow operation in the second schedule instead of two as in the previous schedule. The operations of the part required by the new order is included in the second schedule, its operations are represented by the dark green circles. It can be seen that the new operations have not just been added at the end of the existing schedule, but a whole new schedule has been generated.

This process must be repeated every time a new order arrives in the system. The frequency of the arrivals of new orders determines the frequency of schedule changes, hence the need for frequent planning capabilities in the scheduling mechanism for this study.

Computing a schedule by hand is often time-consuming and hard work. Repeating the process of schedule generation in a job-shop that has a high order arrival rate can become an impractical task, especially if it must be done by hand. In this study simulation is suggested as a means to generate schedules. Simulation-based scheduling is briefly discussed in the following section, a detail discussion will follow in the literature review.

2.2 SIMULATION-BASED SCHEDULING IN THIS STUDY

Simulation can be used as a modelling tool that can sufficiently imitate the real world. A simulation model can thus be used to imitate the processes of the job-shop system under study, the flow of parts through the system and subsequently generate a schedule. Previous work using simulation as scheduling mechanism is stated and discussed in the literature section.

The simulation model should configure itself in terms of ordering of the orders currently in the system, this will be dictated by the currently selected dispatch rule. The model should then be further configured when a simulation run is started to reflect the current state of

the system. The model can then be run to predict the flow of orders through the system. The act of scheduling is executed when an operation on a part is assigned to a machine and possibly queued at the machine. For each scheduling rule investigated, the queue disciplines are set according to the rule.

Instead of calculating a pro-active schedule by hand a reactive schedule is developed by using simulation. The reactive schedule will be evaluated and promises to be more efficient, as simulation imitates the flow of a system very close to the real world.

2.3 CHAPTER SUMMARY

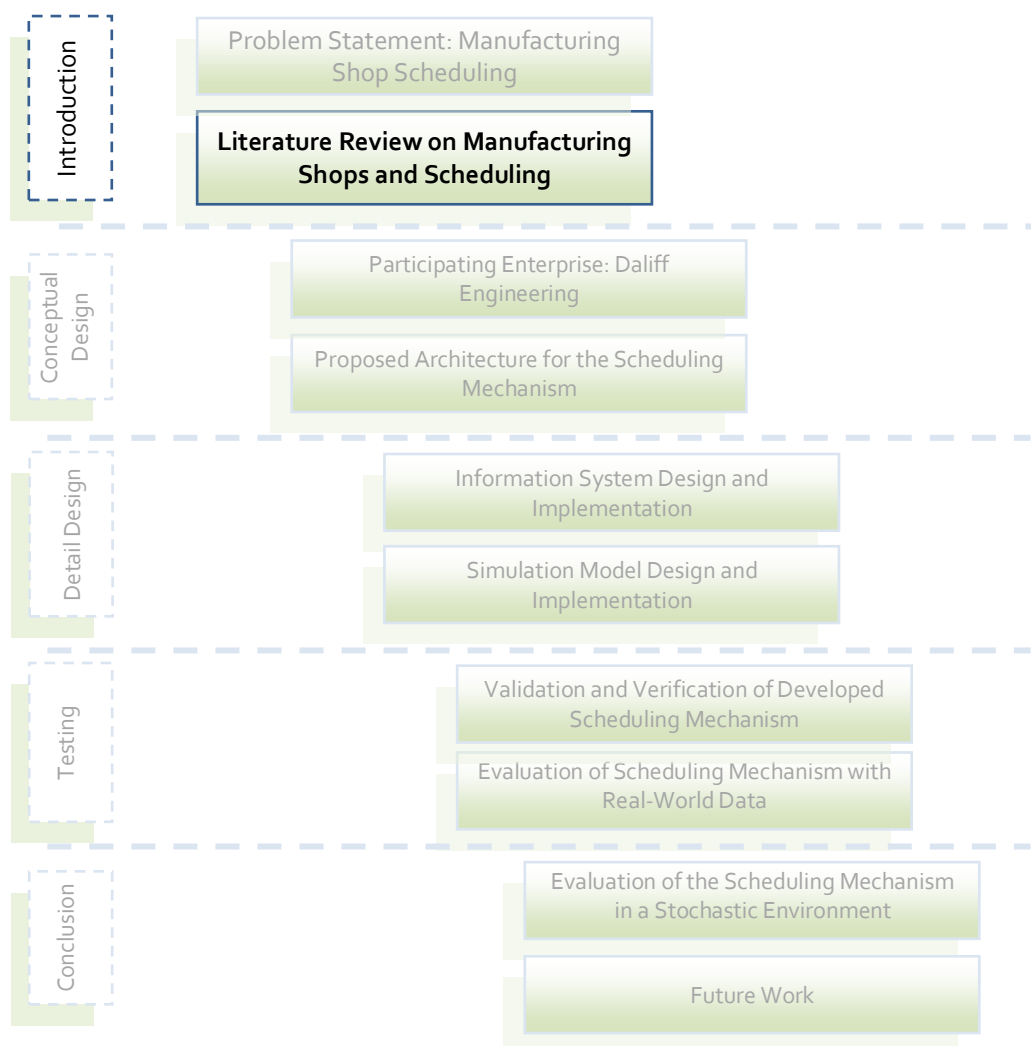
This chapter states the problem that is addressed in this study. It is stated that a scheduling mechanism should be developed to schedule a job-shop. The characteristics that the mechanism should have are stated, with the frequent planning capability discussed in more detail as it is seen as the most important element that the scheduling mechanism should have.

Simulation is suggested as a means to build a scheduling mechanism with the characteristics that were stated. A brief overview of simulation based scheduling is included in this chapter, while a detail discussion is included in the literature review.

In the following chapters, this problem will be addressed, a scheduling mechanism developed and implemented, and finally it will be tested.

3. LITERATURE REVIEW

Real-time simulation-based scheduling and control of a manufacturing shop have, over the years, grown into a very popular research area. Many researchers are studying this area and several different methodologies, frameworks and implementations have been developed and derived. The literature review in this chapter examines the work of several researchers on this topic. The context of this chapter is part of the introduction phase of this thesis, see below.



This chapter starts by describing the different types of manufacturing shops in which scheduling is of cardinal importance, the first being flexible manufacturing systems.

3.1 FLEXIBLE MANUFACTURING SYSTEM (FMS)

Originally, automated manufacturing processes were rigid, fixed and designed for a specific product. However, as manufacturing processes and production demand changed over time, flexibility of manufacturing processes became a requirement. The demand for greater product variety, high quality, affordable prices and fast turnover pushed manufacturers to develop processes that could cope with the fast changing demand. As a result flexible manufacturing systems (FMSs) were developed (Womack *et al.* [1], Groover [2]).

Shnits *et al.* [3] defines a FMS as a manufacturing system that is comprised of automated hardware such as CNC machines, mini-load storage systems, and automated guided vehicles or complex conveyors for material handling.

Typical characteristics of a FMS as stated by Shnits and Sinreich [4] are as follows:

- The FMS is capable of manufacturing a large, but finite, variety of part types.
- Each part type needs to go through several operations in a predetermined order, based on technological constraints.
- Each of these operations can be performed by several machines subject to the availability of the appropriate tooling.
- The processing time of an operation may differ from machine to machine.
- Production orders of the different part types arrive randomly or according to some production requirement list.
- Handling and transferring of parts in the FMS are done in single units (on single unit-load pallets).
- Each work order in the FMS occupies a single resource at any given point in time (for example a machine, a material handling device, an input/output buffer or a central storage buffer location).
- Each machining centre can operate on only one work order at a time.
- There is no pre-emption.
- Tooling change times and load/unload time are included in part type processing time.
- Processing time for each part type operation on each machining centre is known and fixed.
- Work order due dates are known and fixed.

- Machines can break down at random.
- Transportation time between the central buffer and the machining centres is constant for all part types. Material handling devices are available whenever required.

Davis in Banks [5] described a generic FMS by using a schematic diagram shown in Figure 4. Jobs enter the system at the entry mechanism, and the job entities receive a part type from the entry mechanism. This mechanism specifies a probabilistic distribution for inter arrival times between job entities.

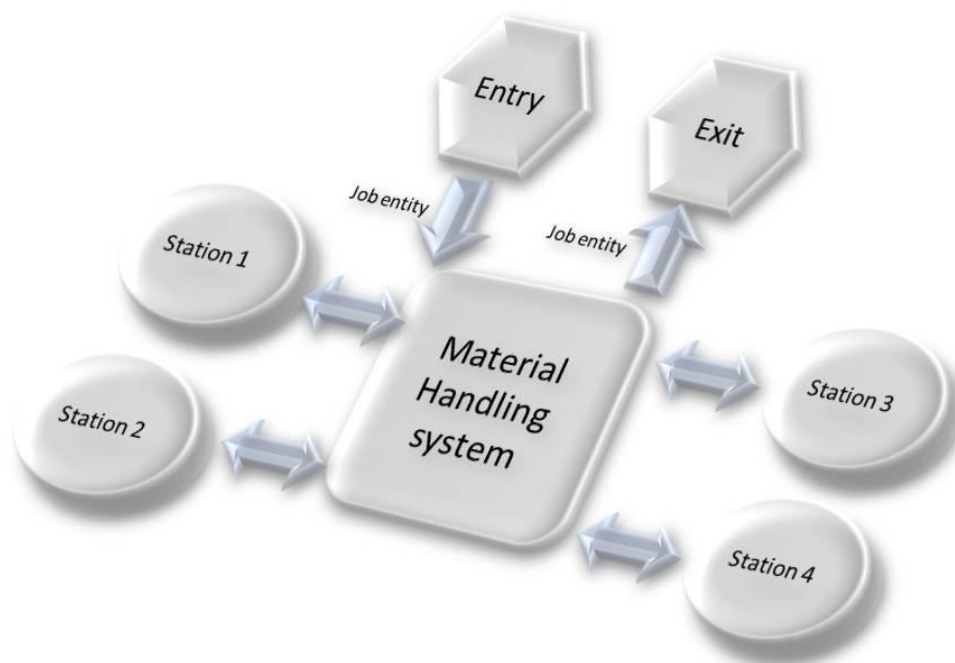


Figure 4 Generic FMS (Banks [5])

Once the job entity is in the FMS, it will follow a certain sequence through the stations. The sequence depends on the given part type of the job. The entity will typically be placed in a queue when arriving at a station and when a machine and operator become available, a setup operation will occur. Setup time is usually predefined and included in the operation time. When setup is finished, the operator is freed and the operation continues.

The machine is unloaded and freed when the operation is finished. The entity will join the output queue of the workstation, where it waits to be moved to the next workstation. The

exit mechanism will free the entity from the system once the processing of the entity is finished.

The flexibility of a FMS is mainly due to versatile machines that enable alternative and flexible routing in the system (Byrne and Chutima [6]). Alternative routing implies that work can be distributed evenly on the different machines, offering better system robustness and utilization. The production of a wide variety of products is also possible due to alternative routing and versatile machines. It is obvious that the correct scheduling of the manufacturing processes is essential to achieve high system efficiency and productivity.

FMSs are in general more sensitive to system disturbances than conventional manufacturing systems because of tighter synchronization, system integration, and interdependencies among automated components. Hence, they require an immediate response to changes in system states (Kim [7]). Real-time scheduling is required and it is known to be a tremendous task, especially as a result of the dynamic environment of a FMS (Tung *et al.* [8]).

A manufacturing shop that relates to a FMS, but does not have that much of a flexible environment, is the job-shop which is subsequently discussed.

3.2 JOB-SHOPS

Job-shops are almost similar to a FMS. The biggest difference between a job-shop and a FMS is the alternative routing capabilities of a FMS. In a job-shop the machines are not as versatile as these in a FMS and the routing options of a job-shop are thus limited.

Hopp and Spearman [9] further define a job-shop as "*Small lots are produced with high variety of routings through the plant. Flow through the plant is jumbled, setups are common, and the environment has more of an atmosphere of project work than pacing*". In general, job-shops specialize in a certain field which requires special skills.

Pinedo [10] clarifies the difference between a job-shop and a flow- and open shop respectively. A job-shop has fixed routes for jobs which differ from job to job, where in a flow shop the routes of jobs are fixed and the same for each job. In a flow shop the machines are set up in series and the jobs flow in the same direction through the series. The flow of jobs in a job-shop follows the route that is assigned to it. Open shops have machines that can do all the operations. The routes are thus not fixed and are not defined according

to the job. The flow of jobs is thus dynamic and adjusted to suit a schedule. Open shops and FMSs are basically the same.

Figure 5 shows which type of shop is applicable in different production environments. The flow shop is best suited when mass production of common parts is needed. When a great variety of customized parts, but only a few of each, must be manufactured a job-shop is applicable. A FMS is appropriate to produce a significant number of parts with a certain degree of variety.

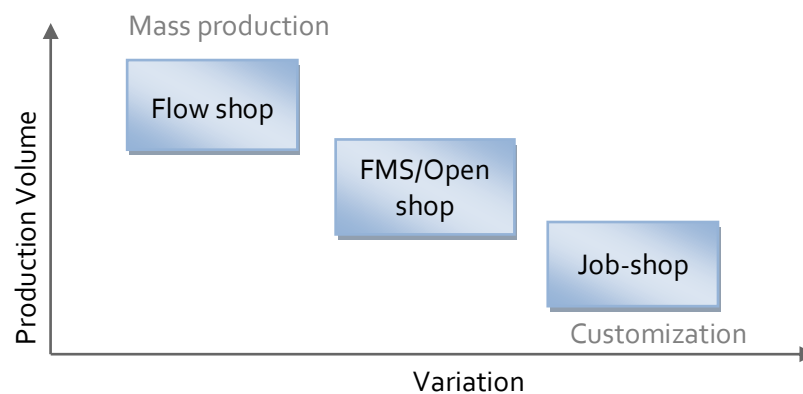


Figure 5 Type of shop according to production environment

The size of orders and type of parts of an order determines the type of job-shop. One such type of job-shop is the make-to-order job-shop in which orders drive the production, where part routing and processing times are determined by the orders. The order sizes are typically small and the parts are custom manufactured and are seldom manufactured again.

Having a perspective on the type of manufacturing shops and their relation to the production environment, the scheduling of a job-shop is subsequently discussed.

3.3 JOB-SHOP SCHEDULING PROBLEM

The manufacturing shop under study is a make-to-order job-shop, and scheduling of the shop must be done, as required by the problem statement. The investigation of job-shop scheduling is therefore included in this study.

Perregaard and Clausen [11] define the job-shop scheduling problem (JSP) as follows. Where n jobs are to be processed on m distinct machines, each job is composed of a set of operations of different time length. These operations are to be processed in an order

defined in the process plan of the part. Each machine is only able to process one operation at a time and once an operation is started, it cannot be interrupted, i.e. pre-emption is not allowed.

Leung [13] gives a more mathematical description of the job-shop scheduling problem:

- A set J of n jobs J_1, J_2, \dots, J_n has to be processed on a set M of m different machines M_1, M_2, \dots, M_m .
- Each job J_j consists of a sequence of i_j operations $O_{1,j}, O_{2,j}, \dots, O_{i_j,j}$ that must be scheduled in this order.
- An operation can only be processed on a specific machine among the m available ones.
- Pre-emption is not allowed and machines can only handle one operation at a time.
- Operation $O_{i,j}$ has a fixed processing time $p_{i,j}$.
- The objective is to find an operating sequence for each machine to minimize the makespan $C_{max} = \max_{j=1,n} C_j$, where C_j denotes the completion time of the last operation of job J_j ($j = 1, \dots, n$)

Sadeh [12] states that in manufacturing, jobs typically have release dates and due dates. The release date specifies a date before which the job cannot start and a due date that a job should ideally be completed. In a make-to-order environment due dates correspond to delivery dates.

According to Sadeh [12], job-shop scheduling is a Constraint Satisfaction Problem (CSP) or Constraint Optimization Problem (COP). The constraints that must be satisfied are these of precedence, capacity, release dates and due dates. The precedence constraints ensure that the job follows the process route that was assigned to it. The capacity constraints exist to prevent allocation of multiple operations to the same resource at the same time. The release date and due date constraints determine the possible time frame in which an operation can be executed. The release date stipulates the date before which a job can not start due to practical reasons, where the due date is the date by which a job should be finished. It is possible that the due date constraint is not met, but some sort of penalty will then occur.

Sadeh [12] uses the schematic in Figure 6 to explain the job-shop scheduling problem. In this problem, there are four jobs on five machines. Each node in the figure represents an operation and is labelled with the name of the operation ($O_{i,j}$), where j is the j -th part and i is the i -th operation, the k -th resource required is indicated by R_k and the duration of the operation is simply shown by a number n . The arrows represent the precedence constraints and the broken lines the capacity constraints. This example assumes that each resource can only do one operation at a time, hence the capacity constraint. If there is more than one operation competing for a resource, all but one have to wait, as they cannot be processed at the same time.

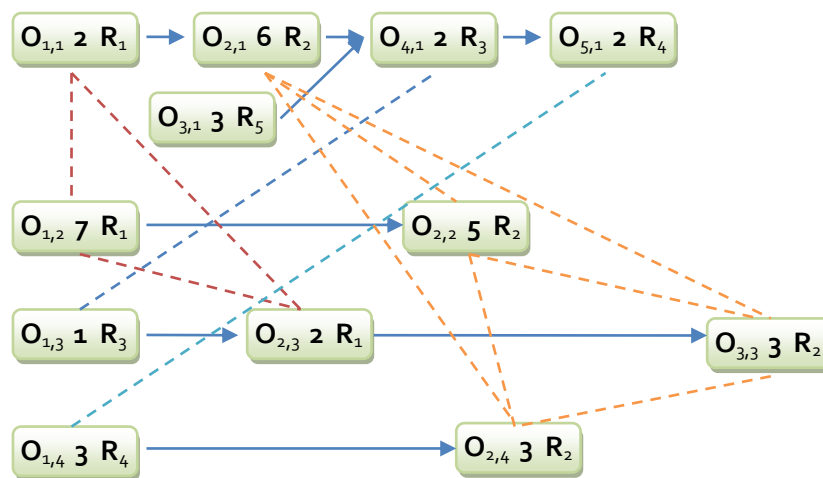


Figure 6 Simple Job-shop problem with 4 jobs (Sadeh [12])

The constraints can be described by referring to Figure 6. Operation $O_{1,3}$ has to be performed before operations $O_{2,3}$ and $O_{3,3}$, hence the precedence constraint. Operations $O_{1,1}$, $O_{1,2}$ and $O_{2,3}$ all have to be performed on resource R_1 , hence the capacity constraint.

Sadeh [12] further states that when some solutions are preferred rather than another, the job-shop problem becomes a COP with an objective function to optimize. Several scheduling performance criteria exist; the specific selection being determined by management structures. A thorough investigation into performance criteria will be discussed later in this study.

The complexity of the job-shop scheduling problem is determined by using the classic travelling salesman problem shown in Figure 10. First the complexity classification of a scheduling problem needs to be discussed.

When a new scheduling problem is developed, an algorithm is required that is more efficient than the normal enumerative search. It is often possible that even with a lot of effort, no efficient algorithm could be found, in which case the theory of complexity can be applied to state that no efficient algorithm could possibly exist.

The definition of the complexity theory is quite technical, and its complete explanation is beyond the scope of this study. A brief description will now be given, starting by describing time complexity.

A measure to quantify the complexity of an algorithm is to refer to the time it takes to execute the algorithm. The running time of an algorithm is measured as a function of the size of its input (n), an algorithm with a larger input size will take longer to solve. In complexity theory the running time of an algorithm is stated in terms of the growth rate of the algorithm. The growth rate is denoted by the notation $O(\cdot)$.

Suppose an algorithm A has a running time $T(n) = O(g(n))$, where $T(\cdot)$ is the running time of the algorithm and $O(\cdot)$ is its growth rate. The algorithm A will be a polynomial-time algorithm, if $g(n)$ is a polynomial function of n , and if $g(n)$ is an exponential function of n , the algorithm A is an exponential-time algorithm. For example, if $T(n) = O(3n^2)$, then the algorithm is a polynomial-time algorithm, on the other hand, if $T(n) = O(2^n)$, the algorithm is an exponential-time algorithm. By hand of the example Leung [13] gives, it can be shown that it is undesirable to have an exponential-time algorithm as the growth of an exponential function is much faster than a polynomial function. According to his example, consider an algorithm with running time $T(n) = O(2^n)$. Computers currently execute one trillion instructions per day, thus if $n = 100$, the algorithm will take more than 30 billion years to run on a current computer (Leung [13]).

The complexity of problems can be classified according to certain complexity classes, the main two classes being **P** and **NP**. **P** is the class of problems that can be solved in time proportional to a polynomial of the input size, i.e. problems with a time complexity of $O(g(n))$, where $g(n)$ is a polynomial function. With any problem belonging to **P**, it is known that it can be solved quickly. Linear programming and determining if a number has a prime are examples of commonly known problems that also belong to **P**. **NP** is the class of all problems that can be solved using a non-deterministic algorithm in time proportional to

a polynomial of the input size. A non-deterministic algorithm works on the concept of guessing a solution and then verifying it.

The difference between problems in the **P** class and problems in the **NP** class can be described as follows. Problems in the **P** class can be solved in polynomial time using a deterministic algorithm, whereas proposed solutions, derived from a non-deterministic algorithm, to problems in the **NP** class can be verified in polynomial time using deterministic algorithms.

P class, together with other complexity classes, belong to the **NP** complexity class, see Figure 7 as illustration. The **NP** class is one of the most fundamental complexity classes which contains decision problems for which solutions can be checked and verified quickly and problems for which it is not known if a solution can be found in polynomial time.



Figure 7 P class belongs to NP class

The next complexity classes of importance are the complexity classes **NP-complete** and **NP-hard**, and can be defined as follows.

NP-hard is the class of problems for which any problem in class NP can be reduced to the specific problem in polynomial time.

NP-complete is the class of problems for which any problem in class NP can be reduced to the specific problem, which is also in the class NP, in polynomial time.

By referring to Figure 8 and Figure 9 a further explanation of these two classes can be given. The arrow represents the process of reducing a problem X (represented by the diamond), whose class it belongs to is known, to the problem P₁ that is being classified (represented by the star) in polynomial time. Figure 8 illustrates the **NP-hard** class, where X, that is known to be from **NP**, could be reduced to P₁ in polynomial time.

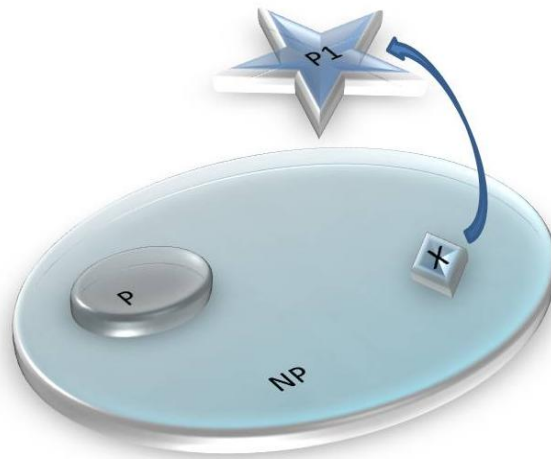


Figure 8 Illustration of NP-hard

Figure 9 illustrates the **NP**-complete class, where P_1 is known to be from **NP** and there exists a problem X from **NP** that is reducible to P_1 in polynomial time.

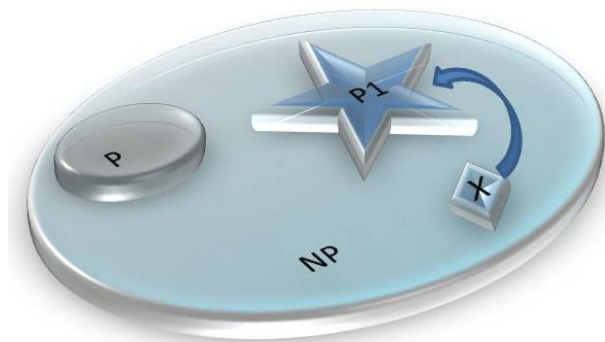


Figure 9 Illustration of NP-complete

Mathematically, it can be explained as follows. Let X and P_1 be two decision problems. We say that X reduces to P_1 in polynomial time, denoted by $X \propto_{\text{poly}} P_1$, if there exists an algorithm that in polynomial time transforms an instance of X , denoted by I_X into an instance of P_1 , denoted by I_{P_1} such that the answer of I_X is yes if and only if the answer to I_{P_1} is yes. Then the classes can be defined as:

A decision problem P_1 is **NP**-hard if $\forall X \in \mathbf{NP}, X \propto_{\text{poly}} P_1$.

A decision problem P_1 is **NP**-complete if $P_1 \in \mathbf{NP}$ and $\forall X \in \mathbf{NP}, X \propto_{\text{poly}} P_1$.

Leung states that the theory of **NP**-hardness only applies to decision problems, and seeing that scheduling problems are optimization problems, it seems that the **NP**-hardness theory is not applicable. By adding an additional parameter to any optimization problem and asking if there is a feasible solution such that the cost solution is smaller, or bigger, than the additional parameter, the optimization problem can be converted to a decision problem. This will further be explained looking at the symmetric Travelling Salesman Problem (TSP).

The TSP is an optimization problem that searches for a solution that minimizes the cost of the total travel of a salesman that needs to visit n cities exactly once and return to the starting city. The diagram on the left of Figure 10 illustrates the problem, the salesman having the task to visit each city indicated by the symbols \mathcal{A} , \mathcal{B} , \mathcal{C} , \mathcal{D} . There is a cost to travel between every city, the cost is the same for travelling from \mathcal{A} to \mathcal{B} as it is travelling from \mathcal{B} to \mathcal{A} making this a symmetric TSP. The costs for travelling for example from \mathcal{A} to \mathcal{B} are not necessarily equal to the cost of travelling from \mathcal{A} to \mathcal{C} , making the total travelling costs differ for different routes. The size of the solution space is $\frac{1}{2}(n-1)!$ for $n > 2$, thus making the size of the solution space for the problem in the figure three, meaning that there are three alternative routing possibilities. One would think that there are actually six routing options, they are as follows:

1. \mathcal{A} to \mathcal{C} ; \mathcal{C} to \mathcal{B} ; \mathcal{B} to \mathcal{D} ; \mathcal{D} to \mathcal{A}
2. \mathcal{A} to \mathcal{D} ; \mathcal{D} to \mathcal{B} ; \mathcal{B} to \mathcal{C} ; \mathcal{C} to \mathcal{A}
3. \mathcal{A} to \mathcal{C} ; \mathcal{C} to \mathcal{D} ; \mathcal{D} to \mathcal{B} ; \mathcal{B} to \mathcal{A}
4. \mathcal{A} to \mathcal{B} ; \mathcal{B} to \mathcal{D} ; \mathcal{D} to \mathcal{C} ; \mathcal{C} to \mathcal{A}
5. \mathcal{A} to \mathcal{B} ; \mathcal{B} to \mathcal{C} ; \mathcal{C} to \mathcal{D} ; \mathcal{D} to \mathcal{A}
6. \mathcal{A} to \mathcal{D} ; \mathcal{D} to \mathcal{C} ; \mathcal{C} to \mathcal{B} ; \mathcal{B} to \mathcal{A}

It is correct to believe that there are six routes, but they can be minimized to only three routes. Routes 1 and 2, 3 and 4, and 5 and 6 are respectively the same routes, but only completed in the other direction. As stated earlier, the cost for travelling from \mathcal{A} to \mathcal{B} is the same for travelling from \mathcal{B} to \mathcal{A} . The travelling costs for the routes that are actually the same, only in the other direction, are thus the same, leaving only three different possible routes to choose from.

The TSP can easily be converted to a decision problem by adding a parameter and stating that there must be decided if there is a solution that gives a smaller cost than the parameter. Looking at the TSP from a running time perspective, the size of the solution space can be described as $T(n) = O\left(\frac{1}{2}(n-1)!\right)$ which is an exponential-time algorithm with input n , making the TSP **NP**-complete.

As stated earlier in the discussion on complexity theory, by transforming a **NP**-complete problem to the problem that needs to be classified, the problem can be classified as **NP**-complete. Figure 10 represents the transformation of a **NP**-complete problem, hence the TSP, to the JSP.

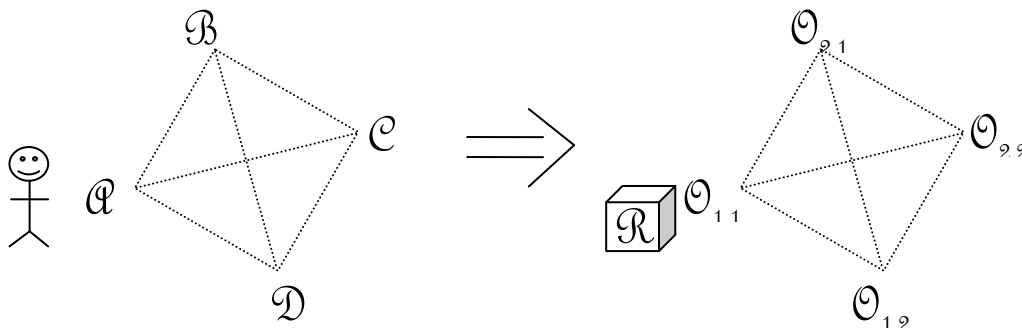


Figure 10 Job-shop problem presented as travelling salesman problem

If the job-shop has one machine, $m = 1$, the JSP can be reduced to a TSP where the machine is the salesman and the operations are the cities (Leung [13]). There are no travelling costs involved as the resource is allocated to the different operations. Other costs do exist though, which implies that the order in which the operations are processed will influence the objective function. Each operation for example has a certain processing time and due date, it is possible to have different schedules in which some operations are past their due dates. In this situation the objective will be to find an order of operations that has the minimum late jobs.

As it was previously shown that the TSP is **NP**-complete, it is evident that the JSP can also be classified as a **NP**-complete problem.

The complexity of the JSP is of such nature that current systems that have been developed can only solve problems with less than 200 operations (Perregaard and Clausen [11]). Another example of the complexity is that the classical job-shop problem, created by Fisher and Thompson [14], with 10 jobs and 10 machines took more than 25 years to solve

(Schutten [15]). Hopp and Spearman [9] indicated that for the classical 10-job 10-machine problem there are almost 4×10^{65} possible schedules, which is more than the atoms that there are in the earth, according to them. The JSP is the most difficult problem in the area of scheduling according to Pezzella *et al.*[16]. As the JSP has attracted a considerable amount of research, many techniques have been developed to solve the problem at hand. The branch-and-bound method and variations of it seem to be the most popular.

Branch-and-bound (see Figure 11 for an example) works on the concept of constructing a search that explores the space of feasible solutions. A search tree is dynamically constructed containing different branches, each containing nodes that represent partial schedules, to explore all possible variations of a schedule. A partial schedule is a schedule that contains jobs from the set of schedulable jobs, but not all of them, i.e. a schedule with only two jobs scheduled after one another, leaving the rest of the five jobs unscheduled. More and more jobs are included in the partial schedules as nodes are added on the next levels of the tree, until all the jobs are included in the schedules making them complete schedules. The number of jobs in a partial schedule is equal to the level number the node is in, for example a node in level two of the tree has two operations in its partial schedule.

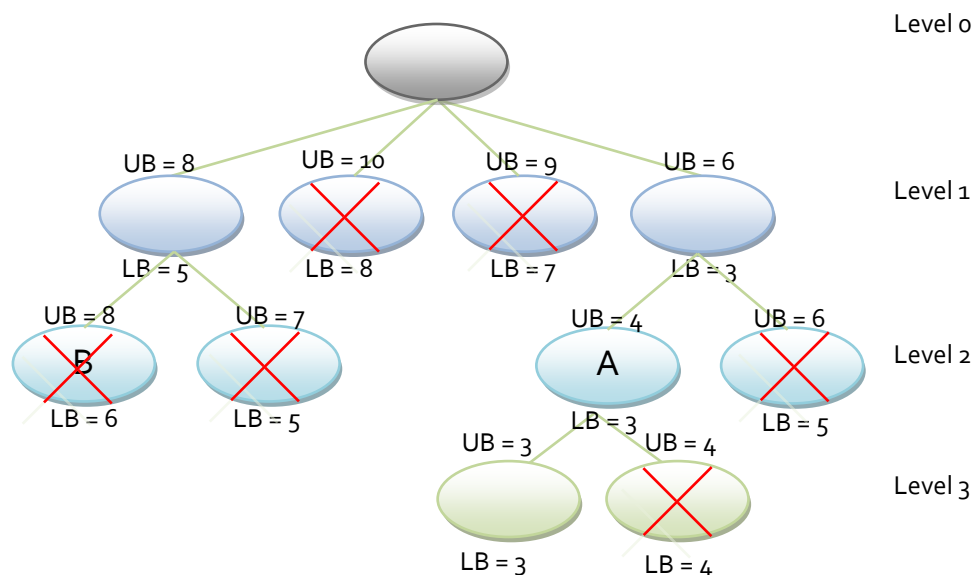


Figure 11 Branch-and-Bound example

The objective of the branch-and-bound method is to find an optimal schedule as fast and effortless as possible. This is achieved by eliminating a branch as soon as possible, i.e. eliminating at the highest possible level of the tree, by means of lower and upper bounds. An upper and lower bound are calculated for each node by a certain algorithm. In the case

of a minimization problem, the lower bound represents the best possible value for the objective function and the upper bound the worst possible outcome. In the case of a minimization problem, a branch is eliminated when its lower bound (the best result achievable) is higher than the upper bound (the worst result achievable) of another branch. In other words, a schedule is eliminated if another schedule exists that in the worst case delivers a better result (see node A in Figure 11) than the schedule would if it could achieve its best result (see node B in Figure 11).

The example in Figure 11 illustrates how branches are eliminated at the highest possible level of the tree. The two branches that are eliminated at level one each have a lower bound value of 7 and 8 respectively, which are both higher than the upper bound of 6 of another node. The rest of the branches are eliminated in similar fashion.

At each node, a set of jobs that can be scheduled (the jobs that satisfy their constraints, for example their release date has been reached) is selected and from this set, a branch is created for each job in the set. The branches form new nodes, containing partial schedules in which the particular job of the branch is scheduled first. For each of these newly added nodes, more branches are created for the jobs not included in that node's partial schedule. This continues until the set of jobs that can be scheduled is empty.

Having described the fundamentals of types of manufacturing shops and job-shop scheduling, the focus now moves to planning, scheduling and control of manufacturing processes.

3.4 PLANNING, SCHEDULING AND CONTROL OF MANUFACTURING PROCESSES WITH JUMBLED PART ROUTINGS

A clear distinction between scheduling, planning and control of a manufacturing process exists and each has its own objectives. Together they form a functional shop floor controller. Planning is responsible for providing production routes for individual parts to meet production requirements. Scheduling uses this list to identify the best schedule to perform the desired tasks, according to some performance criteria. The performance criteria constantly change with changes in system state and production requirements, making the scheduling function dynamic. Execution of the scheduled tasks is the responsibility of the controller. Before planning, scheduling and control of manufacturing

processes are discussed, the distinction among the three elements is shown by describing each individually.

3.4.1 Planning of manufacturing processes with jumbled part routings

As the part routings of all parts are not the same in a job-shop manufacturing plant, each part must have a unique production plan. The planning function will define a list of individual part processing routes; this list usually contains processing times, a sequence of operations, routing options, part quantities, and part due dates.

3.4.2 Scheduling of manufacturing processes

Baker [17] defined scheduling as the “*allocation of resources over time to perform a collection of tasks*”. Pezzella *et al.* [16] states that scheduling is one of the most critical concerns in the planning and managing of manufacturing processes.

Scheduling plays an important role in the overall operational control of many manufacturing systems by efficiently allocating various resources to competing activities. Scheduling in manufacturing systems is easily performed if there are no system disturbances and few production changes, because operations are computer controlled and setup and processing times are deterministic. In this case, offline scheduling would be sufficient.

Offline scheduling, as described by Leung [13], is done in a situation where all the production information is known when scheduling is commenced. All the information regarding the system that needs to be scheduled, like the number of jobs, their release- and due dates, their production plans, etc. is known. Techniques can thus be used to determine the best schedule that minimizes an objective function as the decision maker knows what the future holds.

Two situations in offline scheduling exist, namely deterministic and stochastic processing environments. When the exact processing times are known *a priori* it is a deterministic environment, which cause no difficulty in minimizing the objective function. When the processing times are not known in advance, but it is known to be drawn from one or more probability distributions, it is a stochastic environment. In the stochastic environment, the decision maker needs to find an optimized

schedule using information from probability distributions. The objective function can thus only be minimized in a stochastic sense, making it an expectation rather than a certainty.

A more complicated situation as described above exists when no production information is known *a priori* when the decision maker has to determine an optimized schedule. In this situation online scheduling is required. In an online scheduling model an objective function must be minimized with no information about what the future has in store. No information on the number of jobs that needs to be processed, their release dates, and not even a probability distribution of their processing times exists beforehand. All that the decision maker can do is to determine the best action to take every time a new job is released, which will result in a higher objective function value that could be obtained in the offline case.

Leung summarizes the differences clearly: "*Offline deterministic scheduling deals with the perfect information. Stochastic scheduling deals with some information that may be perfect and other information that is only distributional. Online deterministic scheduling deals with the least amount of information.*" Other than offline and online scheduling, there is dynamic scheduling that can be done in an offline or online scheduling model.

Most manufacturing processes require dynamic scheduling as order arrivals and machine breakdowns occur dynamically. Dynamic scheduling is defined by Church and Uzsoy [18] as "*scheduling that aims to update an existing schedule by reacting to the occurrence of unpredictable events*".

Artigues *et al.* [19] state that there are two types of dynamic scheduling: incremental and regenerative. Incremental scheduling leaves the currently scheduled operations as it is and adds the schedule for the new operations to the existing schedule. With the existing operations taking precedence priority, the lead-time of the new operations may be very long. Regenerative scheduling generates a new schedule for all the operations, new and existing. Operations that have already started are not included in regenerative scheduling.

It can be argued that regenerative dynamic scheduling and online scheduling has a close relation. In online scheduling the arrival of an order would cause the

development of a new schedule. If an arrival of an order can be seen as an unpredicted event in dynamic scheduling and a new schedule has to be developed, it can be seen as regenerative dynamic scheduling. In each type of scheduling, the same action will be taken, stating their relation.

3.4.3 Control of manufacturing processes

Control of a manufacturing process is done by a system controller, which is responsible for converting production requirements into specific instructions for the individual pieces of equipment and interacting with the equipment to implement the instructions (Wysk and Smith [20]). The system controller not only sends instructions to the equipment, but also controls the scheduler. When new scheduling decisions must be developed the controller commands the scheduler to calculate a new schedule. The controller and scheduler are inevitably integrated.

The scheduling and control of manufacturing processes has attracted a considerable amount of research, delivering many different scheduling and control methods. A cursory overview of a few methods will be discussed next, literature references could be used if further insight into a method is needed.

Yamamoto and Nof [21] suggest a scheduling/rescheduling method for real-time control which generates an initial schedule at the beginning of a work period, and schedule revisions are made when significant operational changes occur. Church and Uzsoy [18] analyse scheduling/rescheduling methods that treat the dynamic scheduling problem as a series of static problems that are solved on a rolling-horizon basis. Yih and Thesen [22] formulate the real-time scheduling problem as semi-Markov decision models.

There are several methods using artificial intelligence (AI) techniques for real-time scheduling and control. Maley *et al.* [23] conceptualize a closed-loop control structure for scheduling and control of a computer-integrated manufacturing system. Sarin and Salgame [24] developed an interactive, real-time, knowledge-based approach for dynamic scheduling. Maimon [25] proposes a three-level control system (scheduler level, communication level, and process sequence level), while Shaw [26] views scheduling as a process with two levels of decision making, assigning jobs to appropriate cells and scheduling jobs within each cell.

Inductive learning has been used, based on status data accumulated in a system knowledge base (Shaw *et al.* [27], Piramuthu *et al.* [28]). Neural networks were used as classifiers to create the knowledge base for the learning system (Sun and Yih [29], Soon and Desouza [30], Min *et al.* [31]). A genetic algorithm is incorporated in the learning mechanism using simulation for system training and evaluation (Jahangirian and Conroy [32]).

Simulation can be used for decision making and controlling in real-time scheduling. Applications include emulating real-time control systems, adaptive scheduling and planning, real-time displays of system status, performance forecasting, as well as actual implementation into a shop floor controller (Smith *et al.* [33], Jones *et al.* [34]). Real time scheduling is based on the real time simulation of the system, running in parallel with the actual operation of the system. It consists of making decisions in real time each time a conflict appears (Julia and Valette [35]). Dynamic use of the scheduling period and dispatching rules based on shop floor status analysis is suggested (Kim and Kim [36], Jeong and Kim [37]). Classifiers based on a knowledge base and neural networks are used to analyse shop floor status (Cho and Wysk [38]), while Ishii and Talavage [39] developed a special methodology to determine the future scheduling period.

Several researchers combined different techniques forming complex schemes. Wu and Wysk [40] developed a multi-pass scheduling algorithm that uses a combination of simulation and a learning system. The system learns from its historical performance and makes its scheduling decisions based on simulation of alternative combinations of scheduling rules.

Kim *et al.* [41] uses a combination of inductive learning and neural networks. Inductive learning solves the multi-criteria scheduling problem, while neural networks are used to classify training examples created by simulation runs. The system defines dispatching rules that are most appropriate for a given set of decision criteria in effect for the planning period and which are capable of handling alternative routings. A learning algorithm based on a combination of simulation and an intelligent agent is suggested by Aydin and Oztemel [42]. The study uses a single performance evaluation criterion (average tardiness) and does not support alternative routings.

Reisin-Fournier [43] proposes a look-ahead procedure that not only dynamically selects dispatching rules based on shop floor conditions at decision points, but also learns during the process.

There are studies that use a fuzzy method as a ranking mechanism for the system shop floor conditions or for the scheduling criteria. Kazzeroni *et al.* [44] propose a hierarchical process simulation driven by a fuzzy method. Fanti *et al.* [45] incorporate a fuzzy method and a genetic algorithm for solving a multi-criteria scheduling problem.

All of these scheduling and control methods make decisions based on a chosen performance criteria and dispatching rules. The following section investigates different performance criteria and dispatching rules and how they can be implemented.

3.5 PERFORMANCE CRITERIA AND DISPATCHING RULES

Different techniques have been developed to choose the performance criteria and scheduling rules to be implemented in a system. The decisions are made at various decision points based on the current system state. The general approach is to determine the performance criteria, choose a scheduling rule or a set of scheduling rules and estimate the performance of the chosen rule/rules in terms of the chosen performance criteria. Techniques differ in the sense of when and how the scheduling decisions are made, and how their performance are estimated. This section reviews some of these techniques, but performance criteria and dispatching rules are discussed first.

3.5.1 Performance Criteria Measures for Scheduling

The performance criteria describe the objectives of the system, and could be customer- or system- oriented. Customer-oriented performance criteria would ensure customer satisfaction and a good level of service. A typical objective is to minimize late deliveries. System-oriented performance criteria address system performance, such as minimizing work in progress and flow time. A change in external conditions (such as a change in market demands, organization objectives, the priority of orders, etc.) affects the system objectives as expressed by the scheduling criteria. A change of internal conditions (such as delays on the shop floor or machine breakdowns) affects part routing, dispatching rules, delivery dates and other control decisions.

Several different performance criteria exist and there are numerous ways that the criteria are implemented and used. Shnits *et al.* [3] compiled a list of the most used performance criteria throughout the literature. They are ranked in order of popularity, as follows:

- Minimum mean (or weighted or maximum) flow time, where flow time is equal to the total time that an order is in the system.
- Minimum mean (or weighted) tardiness, where tardiness is the quality or habit of not adhering to a correct or usual or expected time.
- Minimum mean (or maximum) lateness, where lateness is the time by which the completion date of an order exceeds its due date.
- Maximum machine utilization, where machine utilization is the measure of machine hours recorded during production vs. the hours available or scheduled for a given period.
- Minimum average (or maximum or weighted) work in progress, where work in progress is work that has not been completed but has already incurred a capital investment from the company.
- Minimum makespan, sum of machine idle time; minimize the weighted sum of machine idle time; maximize the average utilization of machines over maximum completion time or maximize the average number of jobs processed per unit time.
- Minimum number of tardy jobs, where tardy jobs are jobs that are delayed.
- Maximum throughput, where throughput is the total amount of work done in a given period.
- Minimum average waiting time, where waiting time is the total time in system minus the total operation time of a part.

Buyurgan and Mendoza [46] add two criteria to the list, as follows:

- Mean due date deviation (DDD), where the due date deviation is equal to completion date minus due date.
- Utilization balance (UB), which is equal to the difference in utilization rates among machining centres.

Wu and Wysk [40] stated that a few equivalences in performance measures could be employed to reduce the number of distinct performance measures. This leads to a single

multi-criterion function that includes a greater number of performance measures than what was originally provided. Examples from Wu and Wysk [40] are:

1. Minimize makespan is equivalent to:
 - minimize the sum of machine idle time
 - minimize the weighted sum of machine idle time
 - maximize the average utilization of machines over maximum completion time or maximize the average number of jobs processed per unit time
2. Minimize the sum of completion times is equivalent to:
 - Minimize the sum of waiting times
 - Minimize the sum of flow times
 - Minimize the sum of lateness
3. Minimize average (or total) lateness will also minimize average (or total) tardiness; however the reverse is not necessarily true.

Wu and Wysk [40] suggest a list of the following primary measures:

1. Maximum completion time (makespan)
2. Mean flow time
3. Maximum flow time
4. Number of tardy jobs
5. Mean tardiness
6. Maximum lateness

This section gave a brief overview of the most popular performance criteria measures used as found in the literature. Next dispatching, or also called scheduling, rules are discussed.

3.5.2 Dispatching Rules

When a machine becomes free, it has to be decided which of the waiting jobs (if there are any in the queue waiting for the machine) is to be processed on the machine. For making this decision, a scheduling rule is used to assign a priority value to each of the waiting jobs. The job having the highest priority, which is defined by either the smallest or the largest numerical value, is selected for processing next. Different scheduling rules from different sources are listed below.

Shnits *et al.* [3] defined the following dispatching rules:

- FCFS/FIFO - first come first serve/ first in first out
- EDD - earliest due date
- SS - smallest slack (smallest difference between the due date and the current possible completion time)
- CR - critical ratio (ratio of the remaining process time and the time to due date)
- SRPT - slack/remaining process time (minimal ratio)
- WINQ - work with the least number of jobs in the queue (on the destination machine)
- COVERT - measure based on the minimal difference between the expected waiting time and the slack.

Montazeri and van Wassenhove [47] defined the following dispatching rules:

- SPT - shortest processing time
- LPT - longest processing time
- SIO - shortest imminent operation time
- LIO - longest imminent operation time
- SRPT - shortest remaining processing time
- LRPT - longest remaining processing time
- SDT - smallest ratio obtained by dividing the processing time of the imminent operation by the total processing time for the part (SIO/TP)
- SMT - smallest value obtained by multiplying the processing time of the imminent operation by the total processing time for the part ($SIO*TP$)
- LDT - largest ratio obtained by dividing the processing time of the imminent operation by the total processing time for the part (LIO/TP)
- LMT - largest value obtained by multiplying the processing time of the imminent operation by the total processing time for the part ($LIO*TP$)
- FRO - fewest number of remaining operations
- MRO - largest number of remaining operations
- SLACK/RO - job with the smallest ratio of slack time to the number of remaining operations (slack-per-operation)

- SSLACK/RO - Select the job with the smallest ratio of static slack time to the number of remaining operations
- SLACK/TP - Select the job with the smallest ratio of the job slack time to the total processing time
- SLACK/RP - Select the job with the smallest ratio of the job slack time to the remaining processing time
- SCR - smallest critical ratio, where critical ratio equals (Due date minus Current date) divided by Average remaining operation time.

Vinod and Sridharan [48] not only used existing rules, but defined five new rules. The existing rules are:

- EMDD - Earliest Modified Due Date
- SIMSET - Similar setup
- JCR - Job with similar setup and Critical Ratio
Select a job identical to the job that just finishes processing on the machine. When there is no identical job, select a job with the smallest critical ratio.

Their new dispatching rules are:

- SSPT - Shortest (Setup time + Processing time).
The job with the smallest value of the sum of setup time and processing time is selected.
- JSPT - Job with similar setup and Shortest Processing Time.
Select a job identical to the job that just finishes processing on the machine. When there is no identical job, select the job with the smallest processing time for the imminent operation.
- JEDD - Job with similar setup and Earliest Due Date.
Select a job identical to the job that just finishes processing on the machine. When there is no identical job, select the job with the earliest due date.
- JEMDD - Job with similar setup and Earliest Modified Due Date.
Select a job identical to the job that just finishes processing on the machine. When there is no identical job, select the job with the earliest modified due date with setup time.
- JSSPT - Job with similar setup and Shortest (Setup time + Processing time).

Select a job identical to the job that just finishes processing on the machine. When there is no identical job, select the job with the smallest value of the sum of setup time and processing time for imminent operation.

The most general dispatching, or scheduling, rules from the literature were presented in this section. The implementation techniques and their performance, together with the implementation techniques of the performance criteria measures previously discussed follows in the next section.

3.5.3 Implementation Techniques of the Performance Criteria and Dispatching Rules

There exist several methods to implement and use performance criteria. Shnits *et al.* [3] reviewed the literature and listed some of these methods. The first method which is found in many studies, defines a single fixed objective criterion that drives the system. This objective criterion serves as performance evaluation and does not change with changing conditions.

The second method is a selectable performance criteria method. This method allows the user to define an objective criterion that is in use for the next scheduling period and to change this criterion between different control periods. Such an approach requires frequent user intervention to evaluate system conditions and to change the objective criterion if necessary. If this method is implemented with a long scheduling period it would not be able to respond to system changes in a timely manner, whereas a short scheduling period is used, there will be an unreasonable load put on the user. This will happen because the user needs to analyse data and make decisions in frequent intervals in a short time period.

The third method is called the multi-criteria method. It is a combination of different performance criteria to which the system is constrained. Examples of this are primary and secondary objectives. The primary objective could for example be maximizing the number of jobs completed and secondary minimizing the number of late jobs.

The fourth method uses a set of objective criteria to derive a scheduling policy. The objectives are compiled by the decision maker and the scheduling policy must satisfy them. The decision maker may change the objectives as system conditions change, which may make this approach unpractical due to the load placed on the user.

A variation of the multi-criteria method assigns weights and rankings to system objectives according to their importance. This is to prioritize the system objectives and to set up a measure for scheduling accordingly to meet these weighted objectives. The drawback of this method is that the weights given to the objectives in most cases are determined once for a given scheduling period. Thus the system cannot respond dynamically to changing conditions.

To weight the objective criteria in a dynamic mode, a fuzzy method was developed based on changing shop floor conditions. A drawback of this method is that the scheduling rules for a certain objective criteria are predetermined by experience. This does not leave any room for evaluating their effectiveness for a given shop floor status.

The techniques of implementing dispatching rules and the performance of these rules under certain circumstances are discussed next.

The performance of scheduling rules depends on the configuration of the production system it is implemented on. It is also influenced by the chosen performance criteria. In the literature it is commonly found that researchers contradict each other, which is the result of testing rules on different systems with different configurations and operational policies. It is not always clear what the configuration of the system is, thus it is difficult to generalize results. This section will mention the findings of some researchers most relevant to the current study.

Conway [49] tested 16 priority rules (e.g. SIO, LIO, FIFO, FIRO) on a shop with nine machines. His comparison criteria were measures of WIP, inventory and job lateness. He concluded that the SIO rule dominates all other rules when WIP is the performance criteria, and that SIO also performs well with respect to average job lateness. Hershauer and Ebert [50] stated that the SIO priority rule minimizes mean flow time and that all due date type rules outperform SIO with respect to the cost per order.

Blackstone *et al.* [51] established that SIO is the best priority rule for a due date performance criteria in the following three scenarios:

- the shop has no control over due dates
- the shop has control over due dates and due dates are tight

- the shop has control over due dates, due dates are loose and there is a great congestion in the shop (machine utilization approaches 95%)

Conway [49], Hershauer and Ebert [50] have found that SLACK/RO consistently outperforms other due date based rules. McCartney and Hinds [52] on the other hand, found that when due dates are tight; the SIO rule gives better results than SLACK/RO.

For average tardiness performance criteria, McCartney and Hinds [52] tested three priority rules (FIFO, SIO, and SLACK/RO). They found SLACK/RO to perform better than the two other rules when due-dates are loose. Dar-El and Wysk [53] also concluded that the SIO priority rule performs best for average tardiness. Their ranking for the root mean square (RMS) tardiness performance is WINQ (work in next queue), FIFO, SLACK/RO and SIO.

Montazeri and van Wassenhove [47] found that for average machine utilization SDT (SIO/TP) performs best. It produces a rather high value for variance of machine utilization, which can cause a bottleneck. They found that LPT based rules maximize machine utilization. They also found that the SDT rule delivers the lowest makespan, therefore performs best for a high production rate performance criteria.

Having perspective of the different aspects of scheduling a manufacturing shop, an investigation into the type of scheduling technique that will be used in this study subsequently follows, starting with a description of discrete event simulation as scheduling will be done using simulation.

3.6 DESCRIPTION OF DISCRETE EVENT SIMULATION

A perspective of discrete event simulation (DES) is illustrated in Figure 12. The top down structure was developed from the work of Davis in Banks [5]. The rest of this section will follow the top down structure to define discrete event simulation.

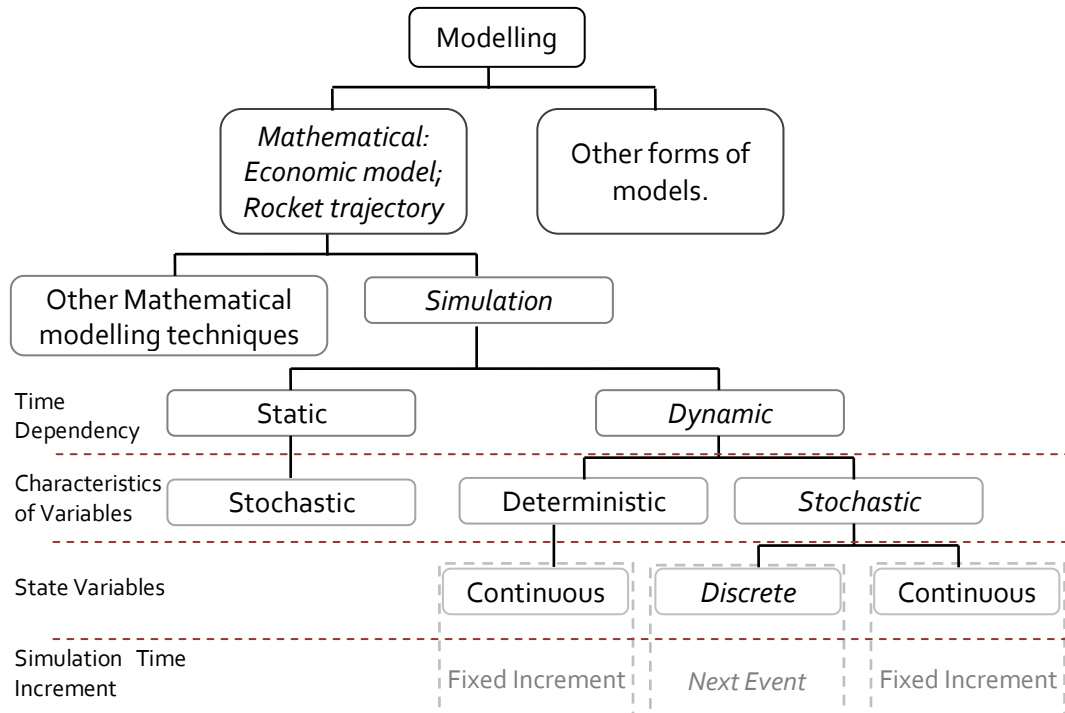


Figure 12 Simulation in perspective

Modelling is defined as the representation of an object in a form other than the object itself (Bekker [54]). Modelling can be used as an aid to the process of thought, communication, training, experimenting, etc. There are several forms of modelling that exists, e.g. physical, mathematical, ecological. A physical model will for example be a scale model of a vehicle, while the trajectory of a rocket can be described by a mathematical model.

Simulation, along with the likes of linear programming and queuing theory is classified as a mathematical modelling technique. Banks [5] defines simulation as follows: "It is the imitation of the operation of a real-world process or system over time. Simulation involves the generation of an artificial history of the system and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system that is represented." Simulation can be used to test the behaviour of a process or system under certain conditions without interrupting the real process or system. Simulation is often used to answer what-if questions (Bekker [54]).

Figure 12 also represents the three traditional simulation dimensions, and the classification of DES can be stated from it. DES is *dynamic* since it is time dependent, *stochastic* as variation is inevitable and *discrete* with regards to time increments. In DES a change of state in a system is an event, which can occur at any instance of time.

DES can thus be described as the imitation of the operation of a real-world process or system in which the system state changes at discrete, and possibly random, points in time. This characteristic enables the use of simulation as an online scheduling tool, which is discussed next.

3.7 USING SIMULATION AS SCHEDULING TOOL

During operation of a make-to-order manufacturing process, the system state changes frequently. A new order, new product or a new variant of a product could be introduced. These manufacturing changes and disturbances like machine breakdowns could force the rescheduling of the future. The impact of these changes and disturbances must be assessed, and if necessary, a new control policy must be developed and proved correct before changes are incorporated into the real system (Drake and Smith [55]).

Simulation is suggested as it provides for a model that corresponds to operation of the real world system under defined conditions, and it makes it possible to evaluate new control policies without implementing them into the real system.

Davis in Banks [5] states that the projected performance of a manufacturing process is usually over-estimated, because a statistical estimate of the steady-state performance of a manufacturing process has been used, and if the system is operated in a flexible manner, a steady-state will never exist. Davis describes on-line planning and control using real-time simulation (see Figure 13). He starts off by stating that on-line planning and control begins with the assumption that a simulation model for the real world exists and withstood the validation process. It is stated that the real-time simulation is necessarily data driven, because it is affected by the same inputs as the real system.

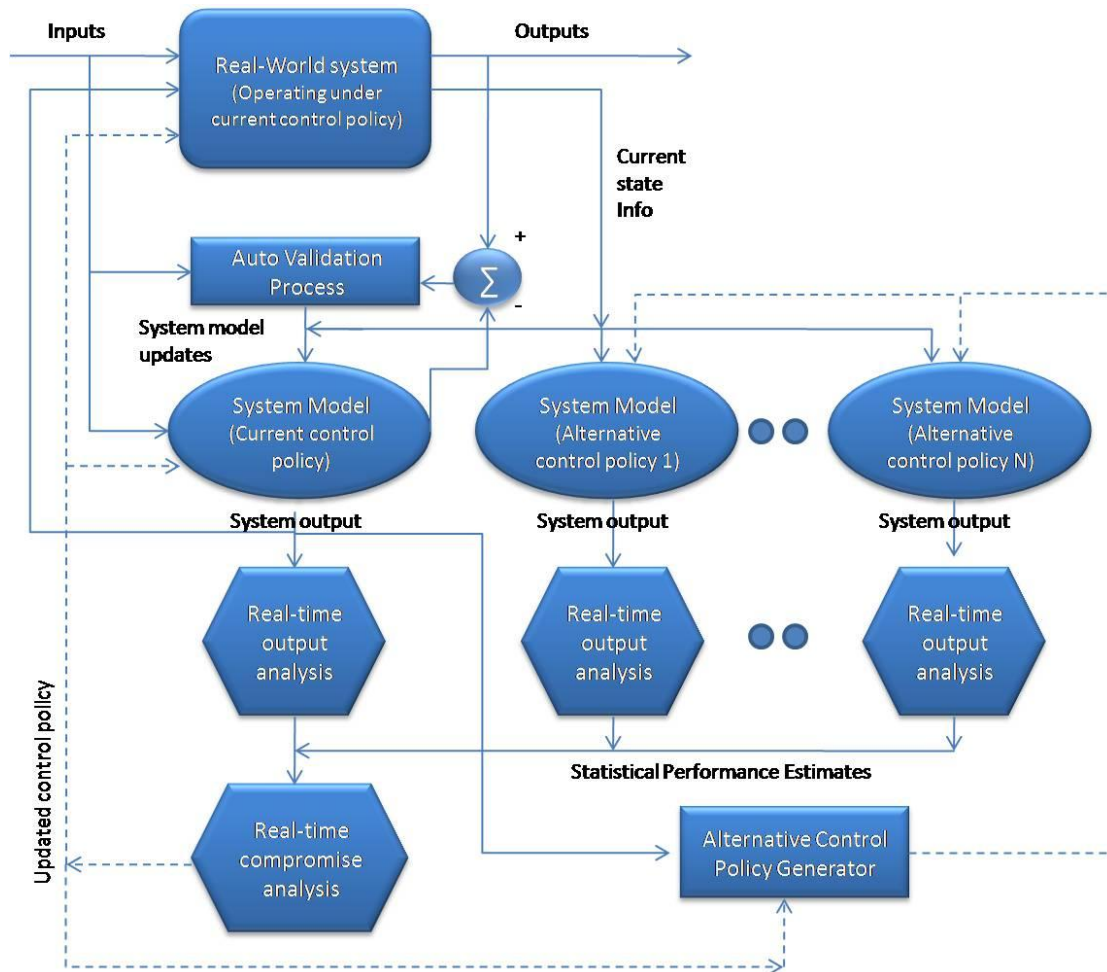


Figure 13 Schematic for the on-line planning/control process using real-time simulation (Banks [5])

From Figure 13 it can be seen that there exist three types of input components. There is the exogenous input component which the system has no control over. The second component is the endogenous control input which depends on the selected control policy. The last component is the current system state.

Davis states that the validation process must continue under the on-line planning and control scenario, even whilst the model has been validated. It is necessary because the system continually changes and the model must reflect these changes. Davis included an auto validation process that compares the output projected by the model against the measured output of the system.

Figure 13 suggests N instances of the system model. Each of these instances considers an alternative control law for possible implementation and the system is simulated with the alternative control law. These simulations have to execute faster than real time to

characterize the future response of the system. The result from each instance is passed on to an on-line output analysis process. If an alternative control policy provides better performance over the currently implemented policy, it replaces the current policy and is implemented immediately.

Kim [7] developed a scheduling mechanism, containing a simulation mechanism and a real-time control system, based on the scheduling/rescheduling approach. The simulation mechanism evaluates various dispatching rules and selects the best one for a given performance criterion. The rules that were selected are the input to the control system. The real-time control system periodically monitors the shop floor and checks the system performance value. A new simulation is performed when the performance of the system significantly differs from the predicted behaviour, or when there is a major disturbance in the system. If a machine breaks down and has to be repaired, a new simulation is run to determine a new schedule without the machine until it is fixed, when a new simulation will be run.

Drake and Smith [56] identified five basic concepts for simulation systems in on-line planning, scheduling and control:

1. *The system should be multifaceted, meaning that the system can be approached from different points of view by different users.*
2. *The logically distinct activities of on-line simulation-based planning, scheduling and control should be separated by the system. Activities includes modelling physical structure, modelling control logic, simulation input, experimentation, output analysis, and task dispatching.*
3. *The system should incorporate not only good human interfaces, but also explicit software interfaces to the activities of model development, simulation input, experimentation, output analysis, and task dispatching.*
4. *The system should allow model modification with minimal effort.*
5. *The system should be flexible yet easy to use for modellers and end-users.*

The framework that Drake and Smith [56] developed after identifying the concepts for simulation systems in on-line planning, scheduling and control is illustrated in Figure 14.

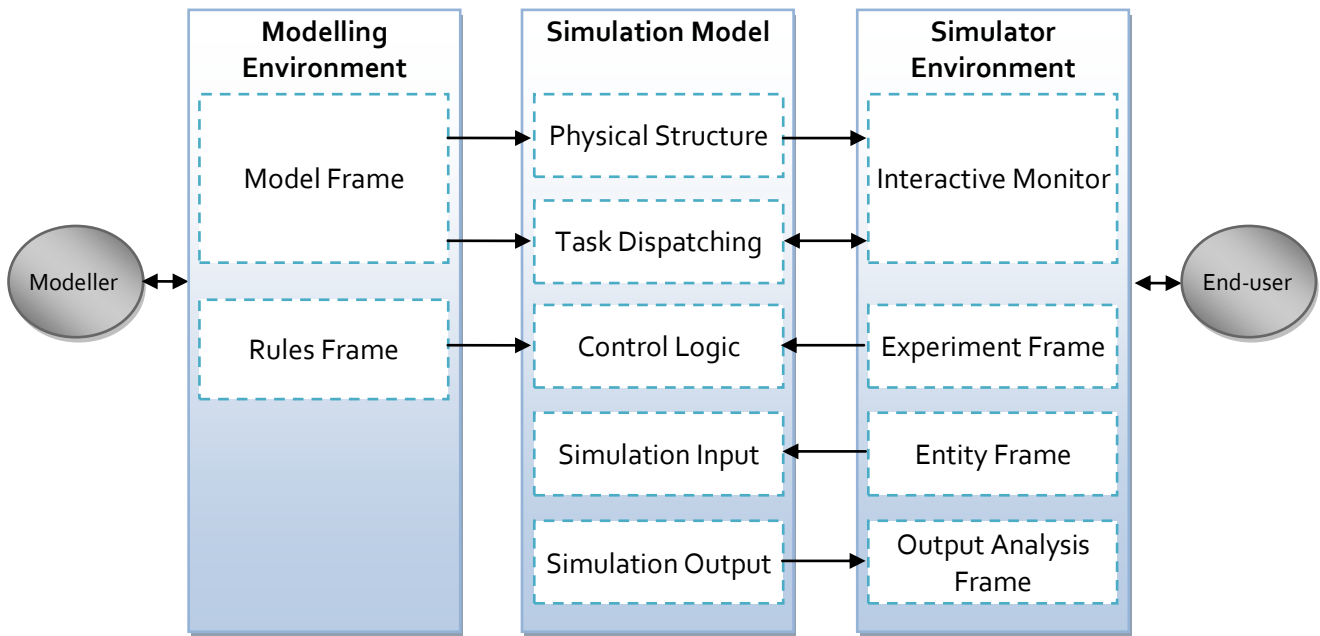


Figure 14 General framework and functional elements of an on-line simulation model (Drake and Smith [56])

The user interface of the framework is divided into two environments, namely modelling and simulator. The modelling environment is for the simulation analysts who define the simulation construct and logic. The physical structure and task dispatching of the simulator is defined in the model frame, whilst the control logic is defined in the rules frame. The simulator environment is for the end-user and includes the activities that define the process. The entity, experiment and output analysis frames are the three main frames of the simulator environment. Their respective functions are simulation input (e.g. order information), experimentation and execution (e.g. choosing scheduling rule), and manipulating and presentation of data (e.g. creating a schedule). The interactive monitor acts as a real-time interface to the simulator.

These two environments act as inputs for the simulation model which has five functional elements, namely physical structure, task dispatching, control logic, simulation input, and simulation output. Figure 14 shows the interaction of each of these functions with the respective frames of the interface environments.

This section discussed how simulation can be used as a scheduling tool, it also presented simulation-based scheduling frameworks that were used in other research projects.

3.8 SUMMARY OF THE LITERATURE REVIEW CHAPTER

In this chapter a discussion of some of the literature on manufacturing scheduling was presented. The different components that reside under the topic and work of several researchers were stated.

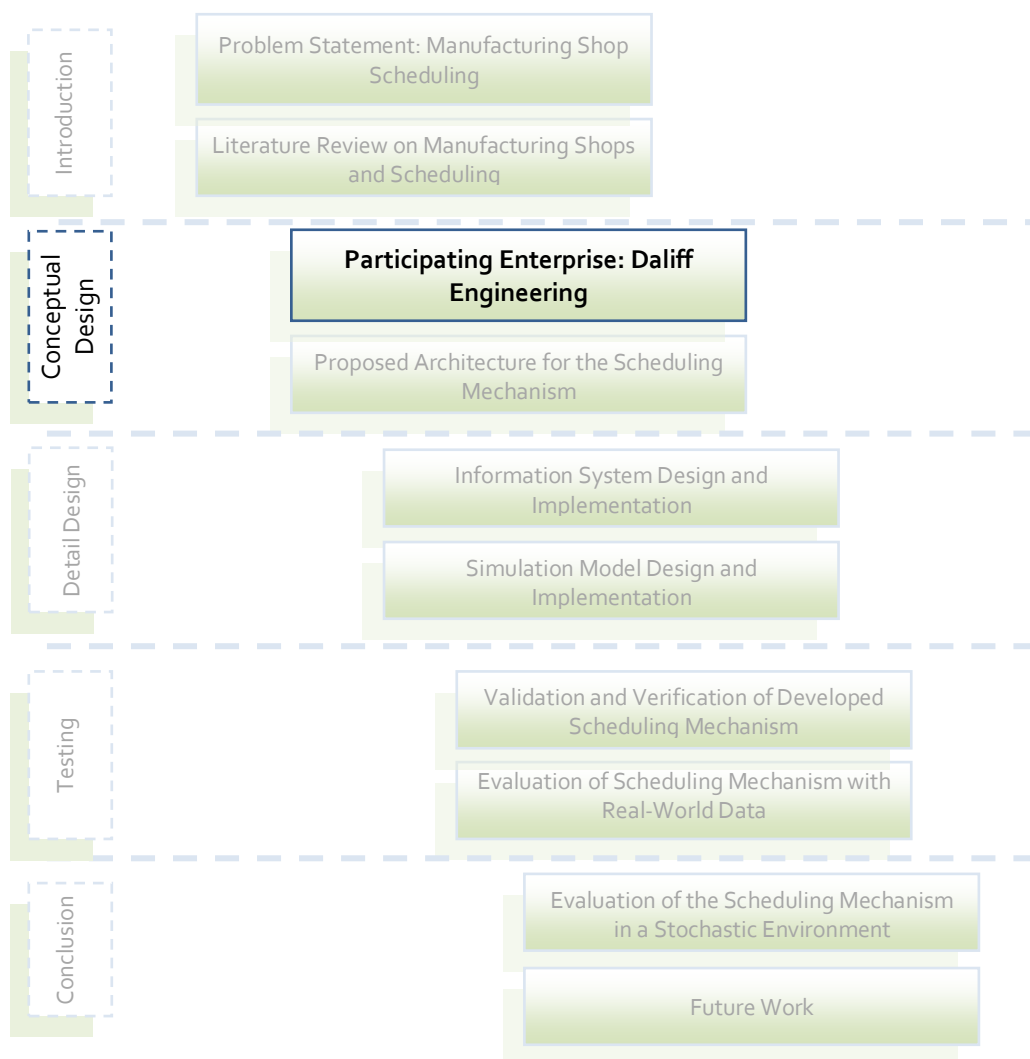
The main areas of discussion included flexible manufacturing systems; job-shops; make-to-order job-shops; the job-shop scheduling problem; planning, scheduling and control of manufacturing processes; performance criteria and dispatching rules; the performance of certain scheduling rules; discrete event simulation; and simulation as scheduling tool.

Previous work on using simulation as a scheduling tool, planning, scheduling and control systems developed, and different performance criteria's and scheduling rules used were illustrated and discussed.

This chapter strives to give the reader a good understanding of the topic at hand so that the rest of this thesis can be placed in context. The next chapter starts the conceptual design phase of this study, stating the manufacturing shop and manufacturing process of the enterprise under study.

4. TECHNICAL DESCRIPTION OF THE PARTICIPATING ENTERPRISE: DALIFF ENGINEERING

An overview of the local enterprise discussed in the problem statement is included in this chapter. The enterprise was previously introduced in the problem statement by giving a general background of the enterprise and its operations. An in-depth investigation about the configuration, characteristics, order-handling and production processes is discussed to develop a scheduling mechanism. This chapter, as shown below, is the start of the conceptual design phase of the study.



Daliff Engineering is a make-to-order job-shop and is situated in Airport Industria, Cape Town. Daliff specialises in the manufacturing of aerospace parts, but also manufacture

precision parts. Order sizes are usually small and each part unique. A detail description of the order processing at Daliff starts the description of the enterprise.

4.1 ORDER PROCESSING AT DALIFF ENGINEERING

The order handling process at Daliff Engineering can be described as follows. A customer requests a quote for a specific part, and one or more drawings of the part accompany the request. The manufacturing engineer determines what materials are needed and what operations must be performed to manufacture the part. The engineer also predicts the number of manufacturing hours and a quote is compiled and sent to the customer.

When the customer accepts the quote, an order is generated. The sheet that the manufacturing engineer used to generate a quote becomes the job sheet for manufacturing. All these are currently done on paper. Machines are set according to specifications and the part is manufactured.

A time sheet for every machine is kept as manufacturing continues. At the end of each working day, manufacturing hours are recorded into a database. Daliff uses the database only as an accounting tool. The processing of orders is illustrated in Figure 15.

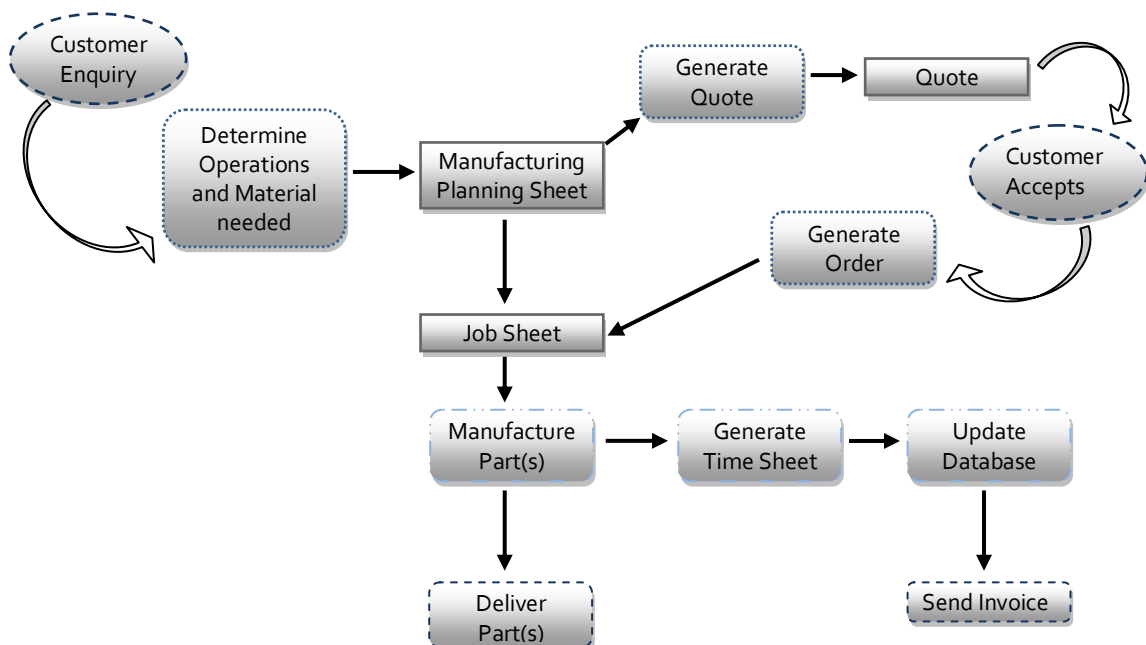


Figure 15 Order processing at Daliff Engineering

Daliff currently uses a responsive scheduling approach. There is no fixed schedule compiled which the operators follow, currently orders are sent to their particular machines and the operator attends to it as soon as possible. The machine configuration of Daliff is subsequently discussed.

4.2 MACHINE CONFIGURATION AT DALIFF ENGINEERING

The machine configuration at Daliff can be divided into two sections: the CNC Milling and Turning and the General Work Shop. Most of the work is done in the CNC section and this study will only focus on this section of the plant. The configuration of the CNC section is shown in Table 1 and Table 2.

Table 1 Milling Machines

Type	Reference	Quantity
General	Old V40	2
	New V40	4
Small	V30	1
	Mini Mill	1
Big	V80	1
5-Axis	DMU	1

Table 2 Turning machines

Type	Reference	Quantity
Small	AP 20	1
Medium	T7	1
Big	Megaturn	1

The new V40 machines perform tasks quicker and can handle more complex procedures than the old V40 machines. The small milling machines type contains two different machines, but because they can process similar tasks they are seen as one station that has a capacity of two. The same qualifies to the old V40 station with a two machine capacity and the new V40 station with a four machine capacity. The general milling machines thus have a capacity of six machines and Daliff has a total of thirteen machining resources.

The machine types classify the size of the parts that each machine can handle. It is possible for a small job to be milled on the V80, although it is actually meant to mill large parts.

The machining processes include a quality control station as well. When a setup for a part is finished, the machining operation is performed on the first unit of the part set. The unit is then sent to the quality control station for inspection. After the inspection is completed, and the quality is acceptable, the rest of the units of the part set can be processed.

The setting of an operation can only be done by personnel called setters, whilst the processing can be done by any of the qualified machine operators. There are only a few setters and they can thus be seen as limited resources that processes compete for.

Transfer times between stations are insignificantly small and are ignored. Daliff and its suppliers have a sound relationship, and material supply will not usually cause an order to be delayed. It can be assumed that there is never a material shortage.

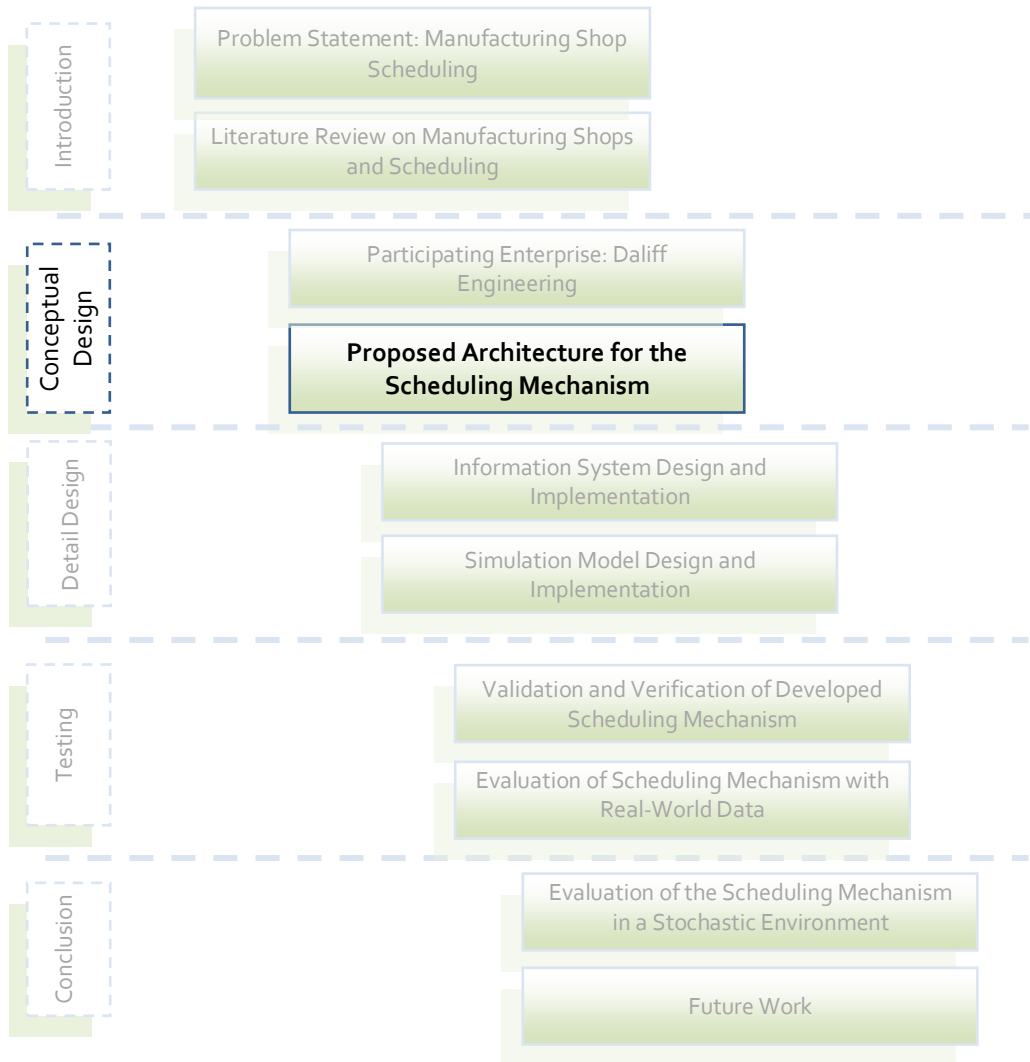
4.3 CHAPTER SUMMARY

In this chapter Daliff Engineering was discussed to illustrate what their manufacturing shop configuration is like. Knowing this, a scheduling mechanism that is appropriate to them can be developed.

In the next chapter the second part of the conceptual design phase of this study is discussed, namely the *architecture* that was developed for the scheduling mechanism.

5. ARCHITECTURE OF THE PROPOSED SCHEDULING MECHANISM

In this chapter the architecture that was developed for the proposed scheduling mechanism is described. It forms part of the conceptual design phase of this study, as illustrated by the thesis road map below. The top level architecture of the complete system and the low level architecture of the simulation model will be discussed.



5.1 TOP LEVEL ARCHITECTURE OF THE SYSTEM

The top level architecture of the scheduler displayed in Figure 16 describes the functionality of the scheduler. It can be broken down into four parts: input, simulation model, results analysis and output.

The input of the scheduler has two components, the enterprise information system and the shop floor. The information system provides information on the orders, while the shop floor component indicates the current state of the shop floor. These inputs drive the second component of the architecture, the simulation model.

The simulation model is configured according to the information system inputs. When the configuration is completed, the simulation model estimates the performance of the scheduling rules under the current shop floor status and latest order configuration.

The estimated performance of each schedule is recorded for analysis. The scenarios are compared to determine which scheduling rule must be implemented based on the performance criteria of the scenarios selected by the user. If the user would like to decrease the makespan of the orders, the scheduling rule that results in the shortest processing time of the current list of orders will be chosen as the best. The scheduler produces an updated schedule, and the schedule can be implemented as the user chooses.

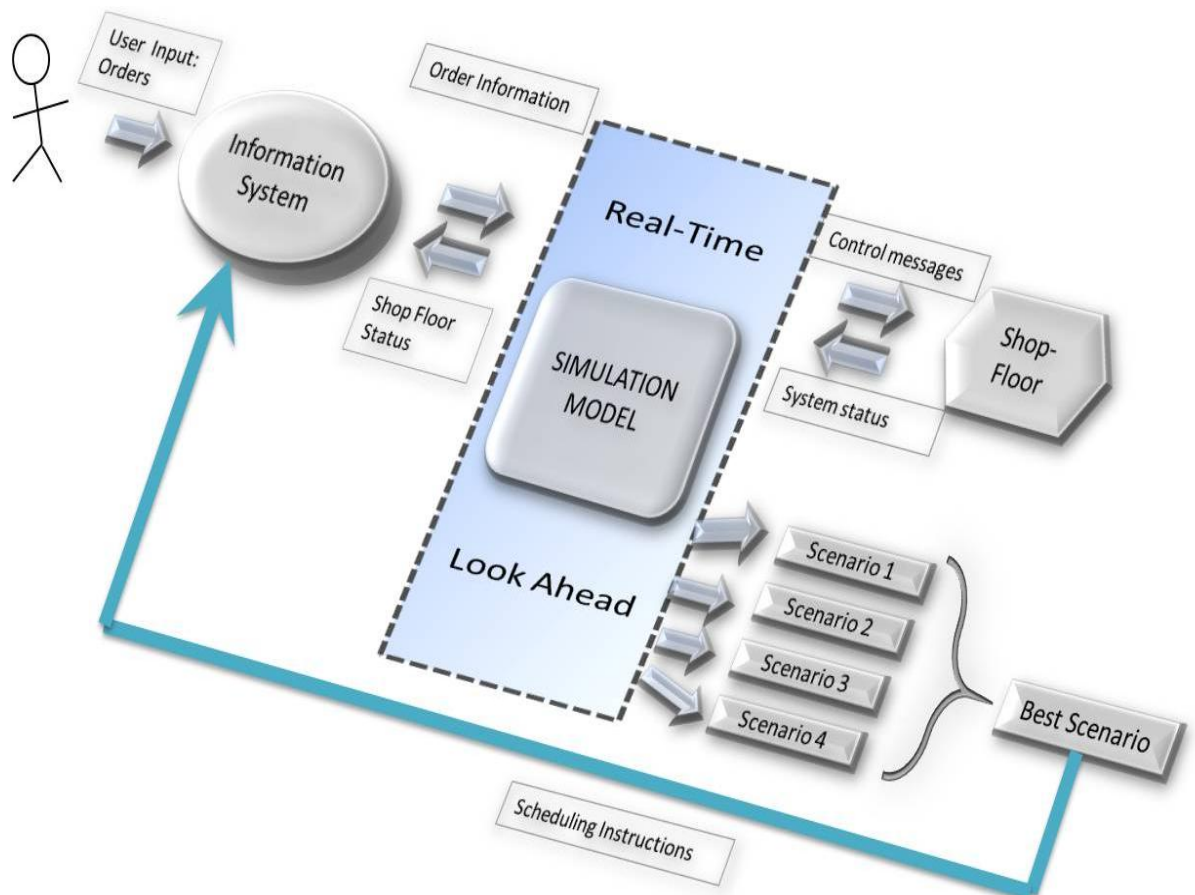


Figure 16 Top level architecture of the scheduler

5.2 LOW LEVEL ARCHITECTURE OF THE SIMULATION MODEL

The simulation model architecture is presented in Figure 17. The enterprise database feeds the simulation model with information about the system status and arriving orders. The database is concurrently updated from the real-world shop floor as production continues. The simulation model executes a query that filters the information in the information system into the format that suits the model, see Table 3. From the result of this query the simulation model is run.

Table 3 Query result

Job No	Approved Date	Prom Date	Ops No	Mach ID	Mach name	Setup timeA	Setup timeB	Setup timeC	Insp time	Proc TimeA	Proc TimeB	Proc TimeC	Status Id
1	2008/06/01	2008/08/02	1	1	V80	2	3	0	1	2	3	4	1
1	2008/06/01	2008/08/02	2	8	V40_New	2	3	0	1	20	30	0	0
1	2008/06/01	2008/08/02	3	2	V40_Old	2	3	0	1	1	2	0	0
2	2008/06/01	2008/07/25	4	1	V80	2	3	0	1	1	3	0	1
2	2008/06/01	2008/07/25	5	4	T7	2	3	0	1	7	8	13	0
2	2008/06/01	2008/07/25	6	3	Megaturn	2	3	0	1	1	7	8	0
3	2008/06/01	2008/08/01	7	1	V80	2	3	0	1	3	8	9	1
3	2008/06/01	2008/08/01	8	8	V40_New	2	3	0	1	10	12	13	0
3	2008/06/01	2008/08/01	9	3	Megaturn	2	3	0	1	2	4	0	0
3	2008/06/01	2008/08/01	10	4	T7	2	3	0	1	1	3	4	0
3	2008/06/01	2008/08/01	11	5	5Axil	2	3	0	1	4	7	0	0
4	2008/06/01	2008/06/30	12	2	V40_Old	2	3	0	1	1	4	5	1
4	2008/06/01	2008/06/30	13	1	V80	2	3	0	1	5	17	0	0

The simulation run is repeated for several independent replications, which allows one to estimate average values over all the replications. A schedule is compiled that may be implemented on the shop floor.

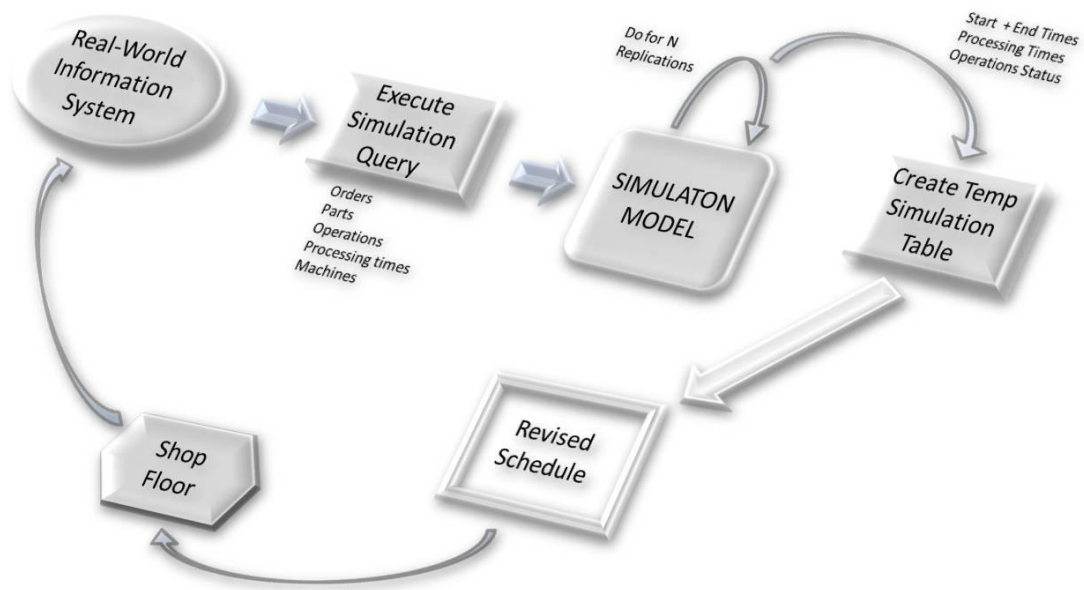


Figure 17 Simulation model architecture

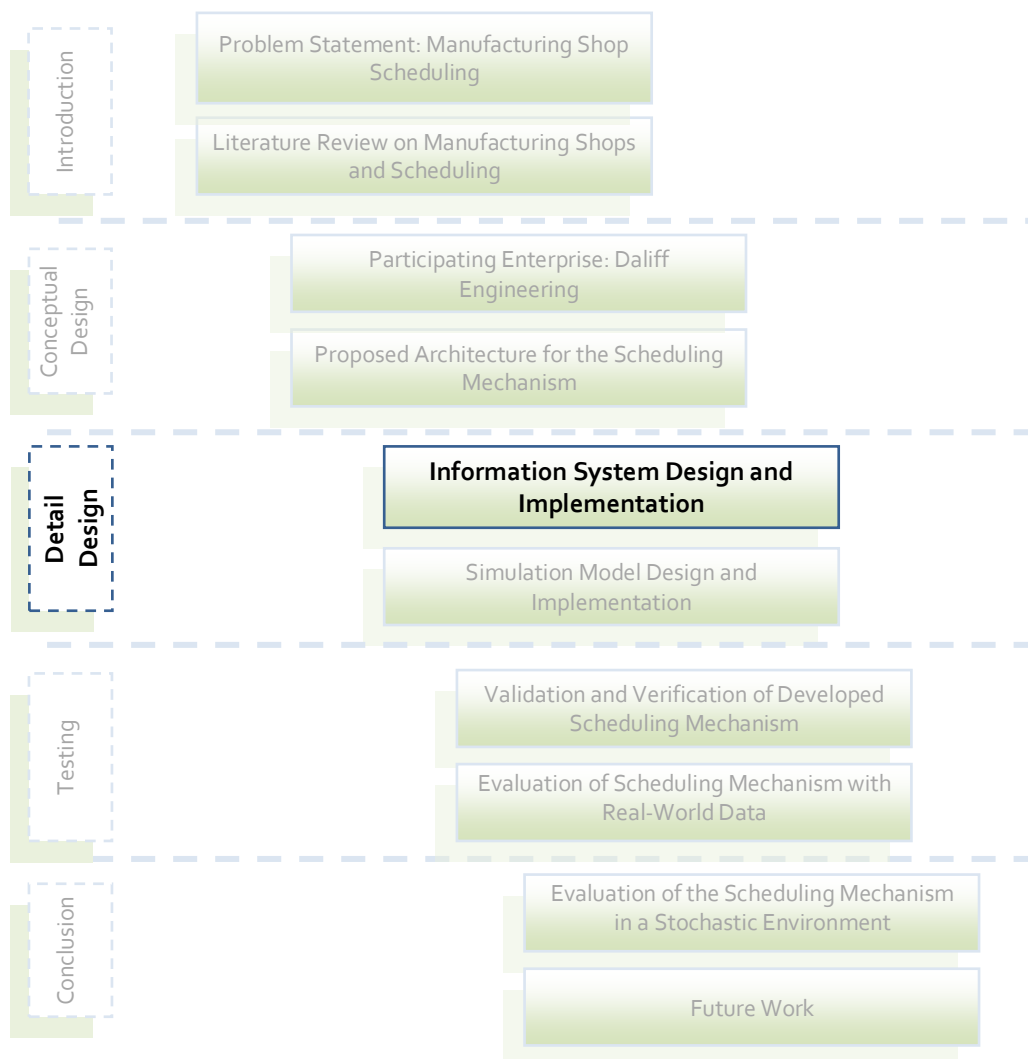
5.3 CHAPTER SUMMARY

In this chapter the architecture that was developed for the proposed scheduling mechanism is described. The high-level architecture was first described to state the concept of the scheduler, followed by a brief overview of the simulation model's architecture.

The design and implementation of the elements of the proposed architecture will now be discussed in subsequent sections, starting with the information system design and implementation.

6. INFORMATION SYSTEM DESIGN AND IMPLEMENTATION

The first part of the detail design phase of this study contains the development and implementation of an information system, see the thesis road map included on this page. The information system has been developed for several reasons, but most importantly to act as an input platform for the simulation model. It is a web-based information system developed in MS FrontPage using ASP coding for dynamic operations, while the data structure was implemented in MS Access. In this chapter the design and the implementation of the information system is discussed.



The information system enables the production planner to electronically generate a quote, which is also automatically stored in the database. When the customer accepts the quote, the production planner changes the quote to an order using the information system. The

information system will then implement this change in quote status and configure the order information to become input to the simulation model.

The information system also enables the user to add new customers and materials. It also has the capability to give the user summary reports of the current quotes, orders and operations, and completed orders.

A detailed description of the information system can be divided into two sections: 1) the data and the information system configuration and 2) processes. Both will be subsequently discussed, starting with the data configuration using entity relationship diagrams.

6.1 INFORMATION SYSTEM ENTITY RELATIONSHIP DIAGRAM AND DATA STRUCTURE

An entity relationship diagram (ERD) shows the relationships among entities. Entities represent any object or event that data has been collected about and a relationship describes the association among these entities. The ERD of the information system developed for this study is shown in Figure 18, followed by a description.

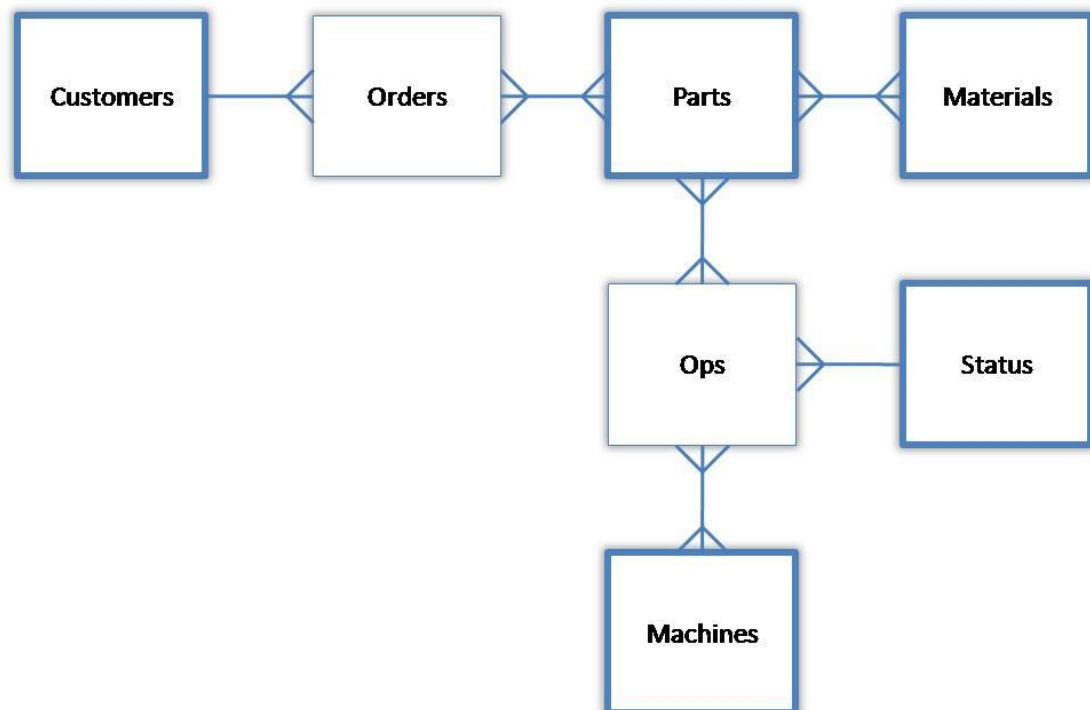


Figure 18 ERD of the information system

The enterprise has several *Customers*. One customer can place many *Orders*, but one order can only have one customer, therefore the one-to-many relationship between customers and orders. The optionalities confirm that customers can exist without having an order, but an order needs a customer to exist. The only other one-to-many relationship is between *Status* and *Ops*. An operation can only have one status, where many operations can have the same status. An operation will always have a status, where it is possible that a status could not be assigned to any operation yet.

Orders can consist of many *Parts*, whilst *Parts* can belong to many *Orders*. Each order consists of at least one part and a part must be assigned to an order. *Parts* consist of many *Materials* where a material can be assigned to many parts. There exist materials that have not yet been assigned to a part, but a part must be made of materials. Each part can have many *Operations* and an operation can be repeated on many *Parts*, both need to be assigned to one another to exist. A *Machine* can manage many operations and it is possible that an operation can be processed on different machines. It is possible that a machine has no operations assigned to it, but an operation needs a machine to be processed.

6.2 RELATIONAL DATA STRUCTURE FOR THE INFORMATION SYSTEM

The entities are represented as tables in the database. These tables are also referred to as relations. The relational data structure further describes these relations by defining the contents of each – these are known as attributes. This structure is illustrated in Figure 19. Note that a specific instance of an entity, i.e. a set of attribute values, is a record in the entity.

```

Customers (Cust ID, Cust_Name)

Orders (Job No, Order No, Cust ID FK, RecDate, ApprovedDate,
PromDate, OrderStatus)

Parts (Part ID, Part_Name, Qty, Drawing_No, NumOps)

Materials (Mat ID, Mat_Name, Stock_Level)

Ops (Ops No, Ops_Name, Setup_Time, Insp_Time, Prod_Time,
Start_DateTime, End_DateTime, CSetup_Time, CInsp_Time,
CProd_Time, Status ID FK )

Machines (Mach ID, Mach_Name)

Status (Status ID, OpsStatus)

```

Figure 19 Relation Data Structure of the Information System

The Primary key of the entity record is underlined with a solid line. A primary key is defined as an attribute that uniquely identifies a record. The attributes with the 'FK' abbreviation are known as foreign keys. A foreign key is any attribute that is a nonkey in one relation but a primary key in another, and they establish the entity relationships.

The following relations of the relational data structure, displayed in Figure 20, are also referred to as associative entities. An associative entity, sometimes called a junction or intersection entity, allows for implementation of many-to-many relationships. The combination of foreign keys forms the primary key of each intersection entity.

```

Orders_Parts (Job No FK, Part ID FK)

Parts_Materials (Part ID FK, Mat ID FK, Mat_Qty)

Parts_Ops (Part ID FK, Ops No FK)

Ops_Machines (Ops No FK, Mach ID FK, Enforce)

```

Figure 20 Relational Data Structure containing associative entities

6.3 DATA DICTIONARY

The data dictionary contains data about the data elements (attributes) in an entity. The data about the data is known as metadata and describe the data type, allowable range and default value, among others. The data dictionary for this design is included in Appendix I.

6.4 INFORMATION SYSTEM CONFIGURATION AND PROCESSES

The configuration of the information system is described by data flow diagrams of the processes of the information system. The implementation of this configuration is shown by several screenshots of the information system interface.

6.4.1 DATA FLOW DIAGRAMS OF THE MAIN PROCESSES OF THE INFORMATION SYSTEM

A data flow diagram (DFD) is a graphical representation of data flow. The information system that was developed has seven main processes. The DFD for each process is subsequently illustrated and described separately.

Figure 21 shows the DFD for the quote generation process. When a customer requests a quote, the request gets added to the order master record. A quote is generated using data records from the customer, material and machine master records. The quote is sent to the customer and the quote also gets updated in the order master record.

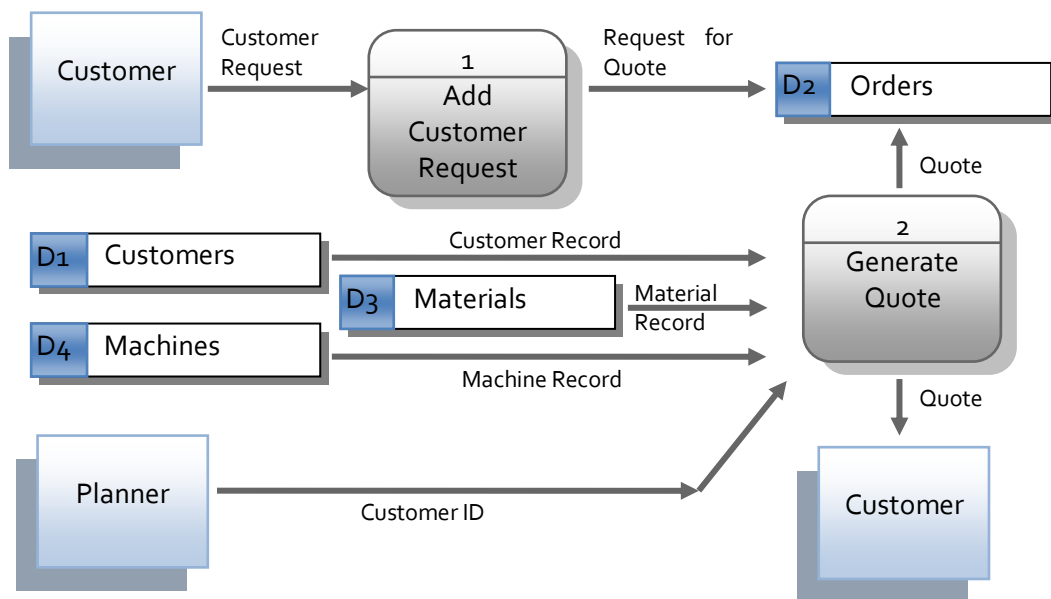


Figure 21 DFD for quote generation process

Figure 22 shows the DFD for the process of adding a new customer. The customer information is used to generate a customer record that is saved in the customer master record.

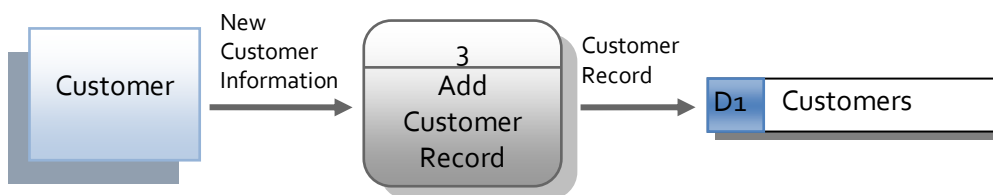


Figure 22 DFD for add new customer process

The customer places an order by accepting the quote. The status of the applicable record in the order master data store gets changed from a quote to an order by the user. An order is then generated by using information that was stored with the quote, and the order is sent to the production planner. The DFD describing the status change process is shown in Figure 23. The user chooses the particular quote that he/she wants to change into an order. The particular order is selected from the orders master data store. Its status gets changed from a quote to an order and the change is stored in the orders master data store.

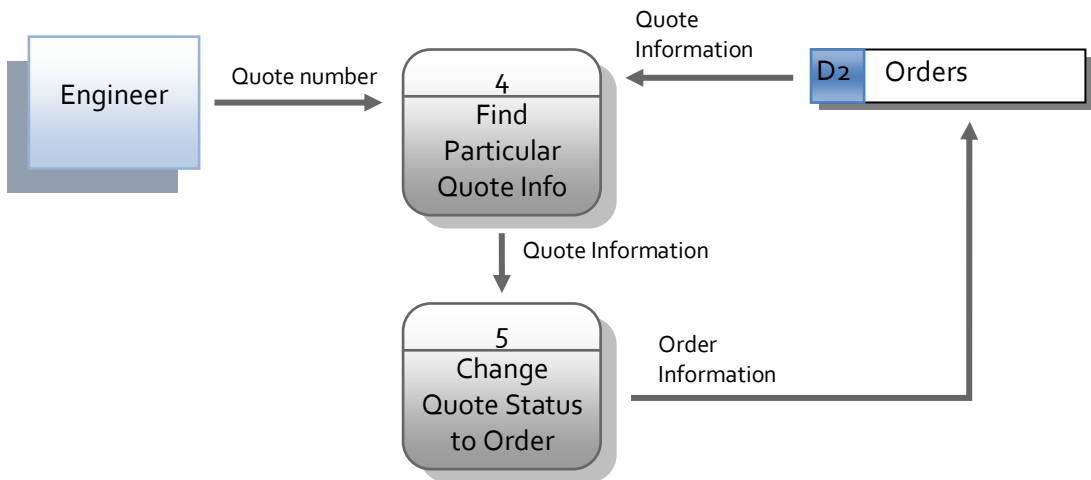


Figure 23 DFD for quote status change process

The user can request a report on the existing orders, quotes or operations in the system. The DFD, see Figure 24, illustrates the process of report generation. When the user sends a request for a report, the relevant data is collected from the orders master data store. A report is then generated and presented to the user.

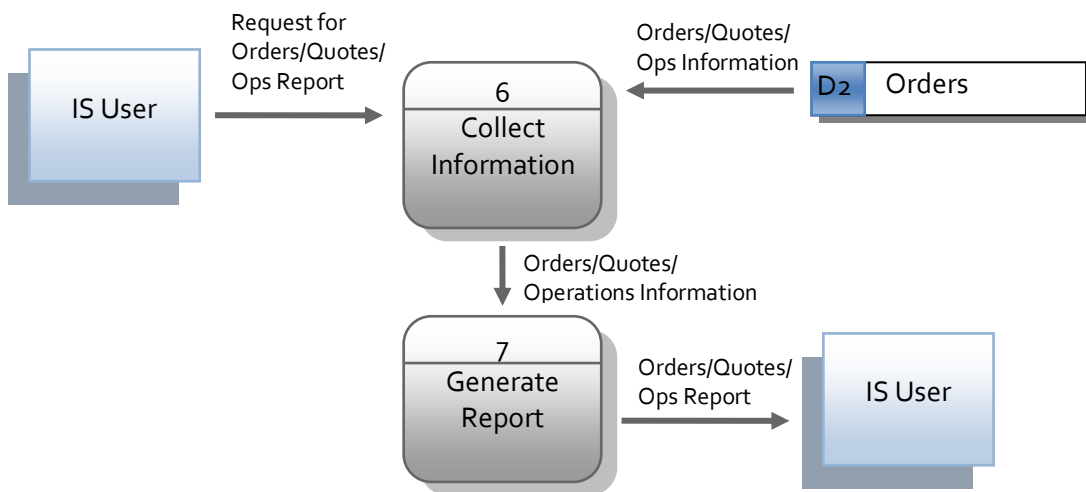


Figure 24 DFD for report process

Figure 25 shows the DFD for the process of adding a new material type. The new material information is used to generate a material record that is stored in the material master data store.

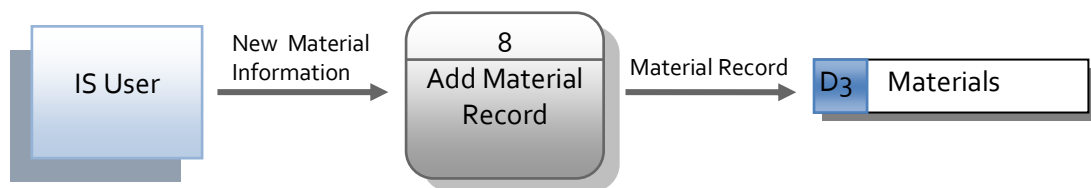


Figure 25 DFD for add material process

A child diagram is an exploded version of a process. The process that is exploded is called a parent process, and the inputs and outputs for the child diagram must be the same as those of the parent process. The only process that is significant enough to explode is Process 2, the generation of a quote. The child diagram of this process is shown in Figure 26.

The information of the customer that requested the quote must first be found from the customer master data store. This information is added to the quote information. The particular material that will be needed to produce the order is selected from the material master data store. The material information is added to the quote information. The operations that need to be performed are created and stored in the quote information and in the operation master data store. The machine on which each operation must be performed is selected from the machine master data store. This selection information is added to the quote information. The machining time of each operation on the respective machines is calculated and also saved to the quote information. The quote information is then sent as an output of the quote generation process. The quote record is now completed and also acts as an output.

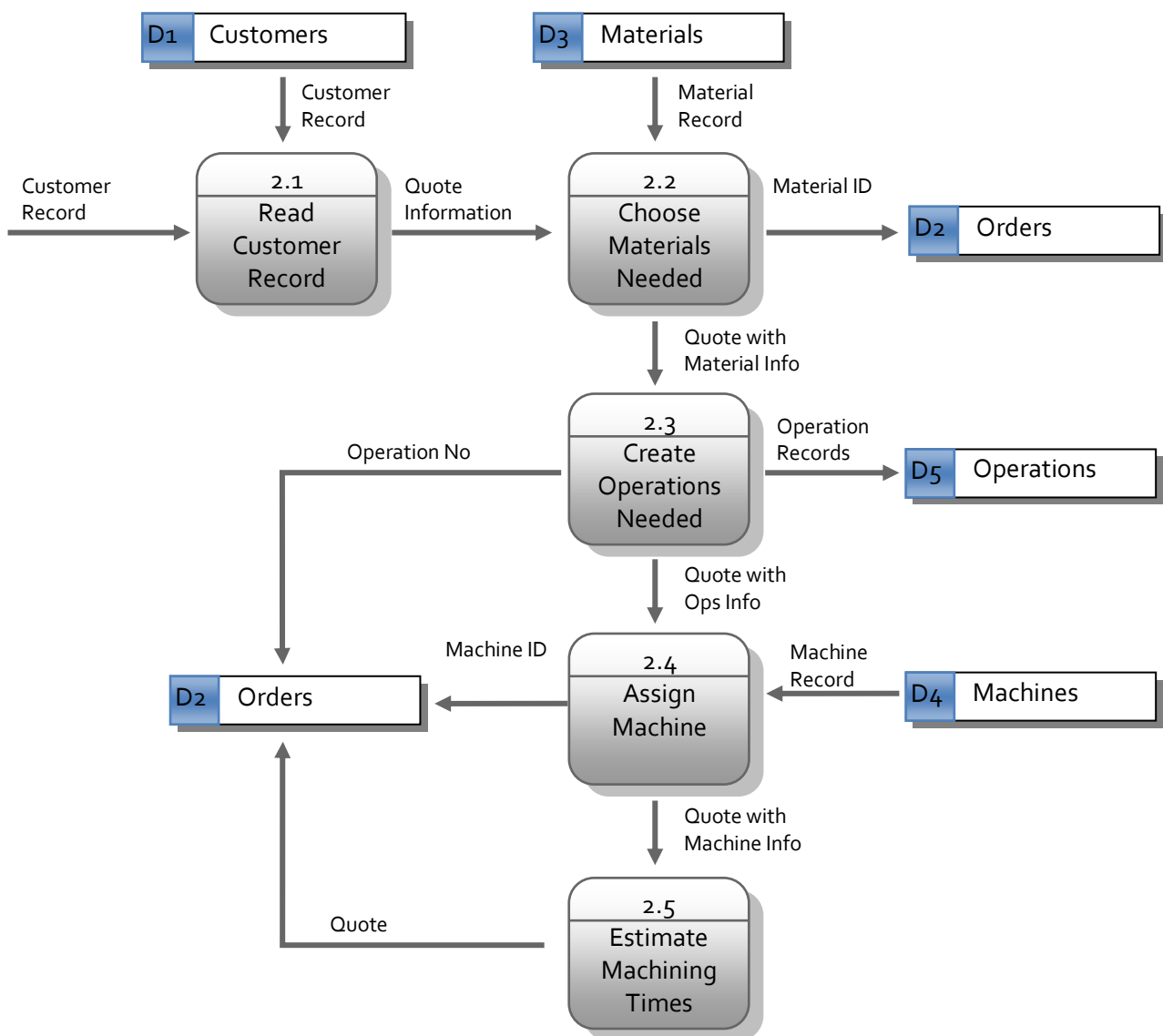


Figure 26 Child diagram for the quote generation process

6.4.2 INFORMATION SYSTEM INTERFACE

The interface of the information system is best described using screenshots. In the following few pages, screenshots of the main processes are shown. The homepage with the user options is shown in Figure 27.

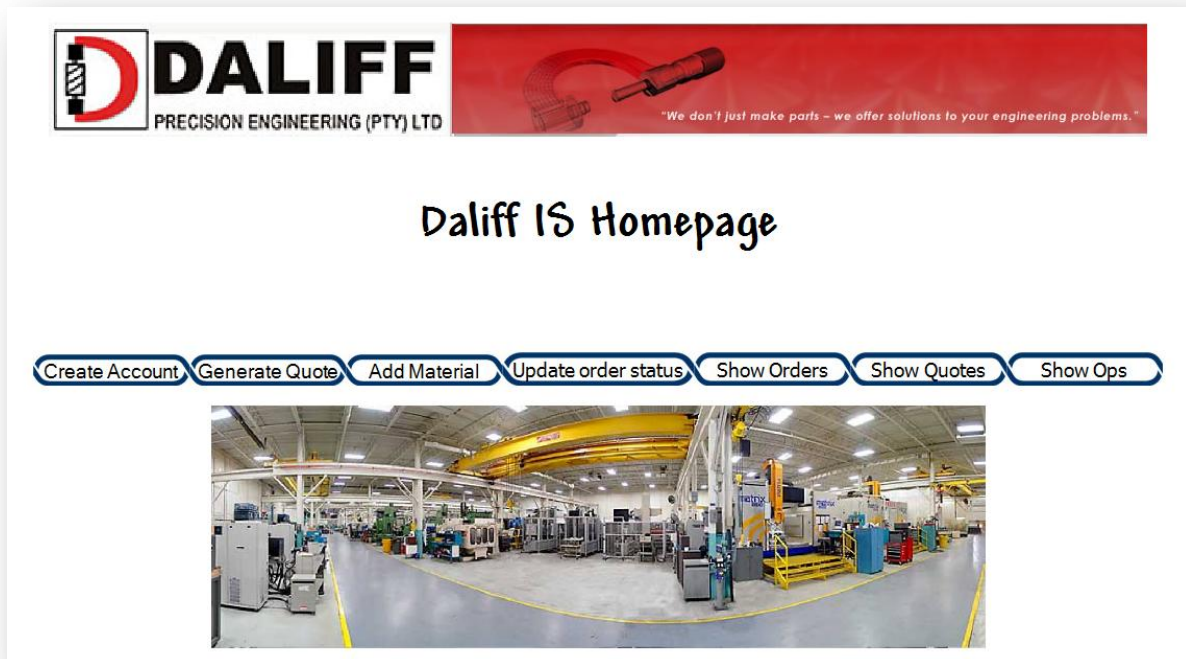


Figure 27 Homepage screenshot

The page used to add material is shown in Figure 28. It contains input fields for the material name and quantity.

Add Material

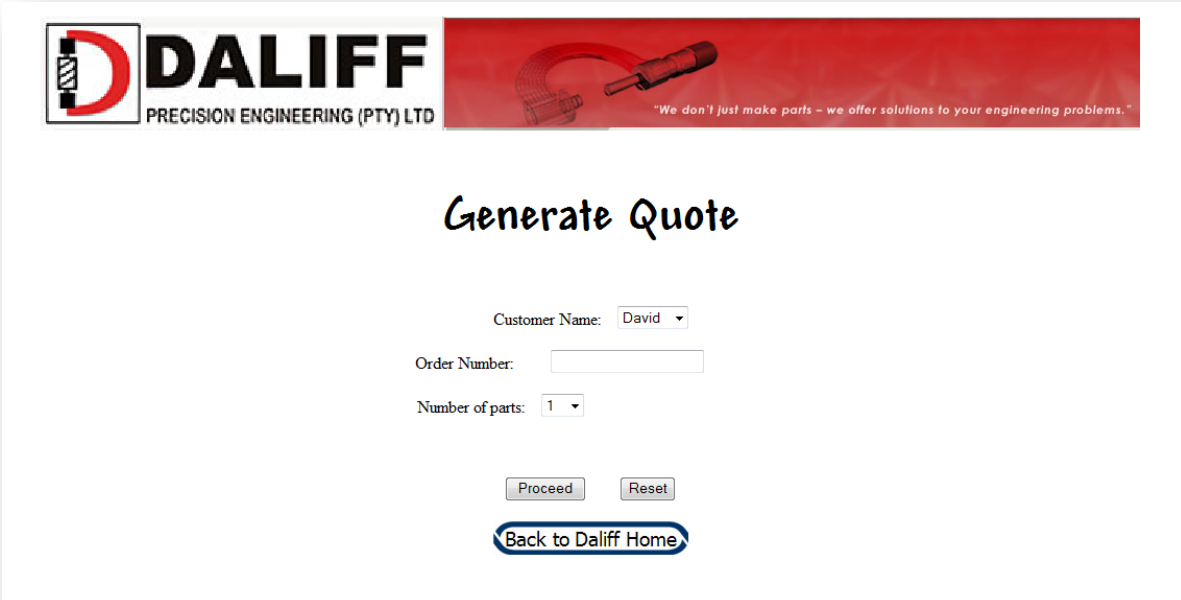
Number of material types wanted to add: 2

	Material Name:	Material Quantity:
1		1
2		1

[Back to Daliff Home](#)

Figure 28 Add material screenshot

The page used to generate a quote is shown in Figure 29. The name of the customer that requested the order is selected with the drop-down box, whilst the order number is written to the appropriate input box and the number of parts the order consist of is selected with the second drop-down box.



The screenshot shows the 'Generate Quote' page for Daliff Precision Engineering (PTY) LTD. The page features a header with the company logo and tagline: "We don't just make parts - we offer solutions to your engineering problems." Below the header, the main heading is "Generate Quote". The form includes three input fields: "Customer Name" (a dropdown menu with "David" selected), "Order Number" (a text input field), and "Number of parts" (a dropdown menu with "1" selected). At the bottom of the form, there are three buttons: "Proceed", "Reset", and "Back to Daliff Home" (which is highlighted with a blue border).

Figure 29 Generate quote screenshot

The page that follows the "Generate quote" page is the "Assign part information" page, shown in Figure 30. The name of the part, the quantity, its drawing number, the number of operations that needs to be executed to produce the part and what type of material is needed must be chosen and act as inputs.

DALIFF
PRECISION ENGINEERING (PTY) LTD

"We don't just make parts – we offer solutions to your engineering problems."

Assign Part Information

[Add New Material to materials list](#)

Name of Part 1: (Enter part name here)

Quantity: (Enter Quantity here)

Drawing: (Enter Drawing No here)

Number of Ops:

Select one or more items

- Al
- Mg
- Cu
- Fe
- Wood
- Fibre
- Plastic

Figure 30 Assign part information screenshot

The next page in the quote generation process is the “Add operations” page, shown in Figure 31. The number of operations related to the order has already been determined, the information of each operation now needs to be added. The operation name, processing time and setup time must be entered into the text boxes. The machine on which the operation must be performed is chosen from the drop-down box.

DALIFF
PRECISION ENGINEERING (PTY) LTD

"We don't just make parts – we offer solutions to your engineering problems."

Add Ops

Account Number: D1

Account Name: David

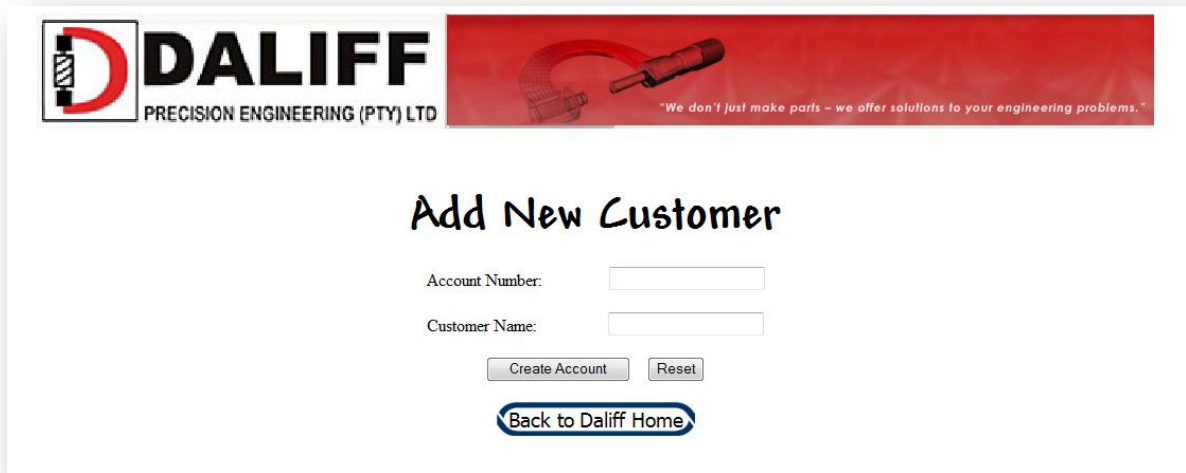
Order Number: 123

Number of parts: 1

bas		QTY of Parts: 1	
<u>Ops Description</u>	<u>Enforce Machine</u>	<u>Processing Time</u>	<u>Setup Time</u>
Ops 1: <input type="text"/>	V80 <input type="checkbox"/> (check to enforce)	<input type="text"/>	<input type="text"/>

Figure 31 Adding operations to an order screenshot

The page used to add a new customer is shown in Figure 32, the account number and customer name need to be entered into the input boxes.



The screenshot shows the top header of the Daliff website with the logo and tagline. Below the header is a form titled "Add New Customer". The form contains two input fields: "Account Number:" and "Customer Name:". Below these fields are two buttons: "Create Account" and "Reset". At the bottom of the form is a blue button labeled "Back to Daliff Home".

Figure 32 Add new customer screenshot

For changing a quote to an order, the user visits the page shown in Figure 33. The particular quote is selected from the drop-down box and the accepted date and promised date are entered; the user can enter the date either manually or by the use of a pop-up calendar.



The screenshot shows the top header of the Daliff website with the logo and tagline. Below the header is a form titled "Update Quote to an Order". The form contains a drop-down menu labeled "Choose from the following Quotes:" with "123 David" selected. Below the drop-down menu are two date input fields: "Accepted Date:" and "Promised Date:", both showing "2008/09/04". Below these fields is a button labeled "Change Status". At the bottom of the form is a blue button labeled "Back to Daliff Home".

Figure 33 Update quote to an order screenshot

The pages displayed when reports of the current quotes, orders, etc. are requested, contains simply a table that lists the particular records. This concludes the description of the information system interface and also the chapter on the information system, a brief overview of the chapter follows.

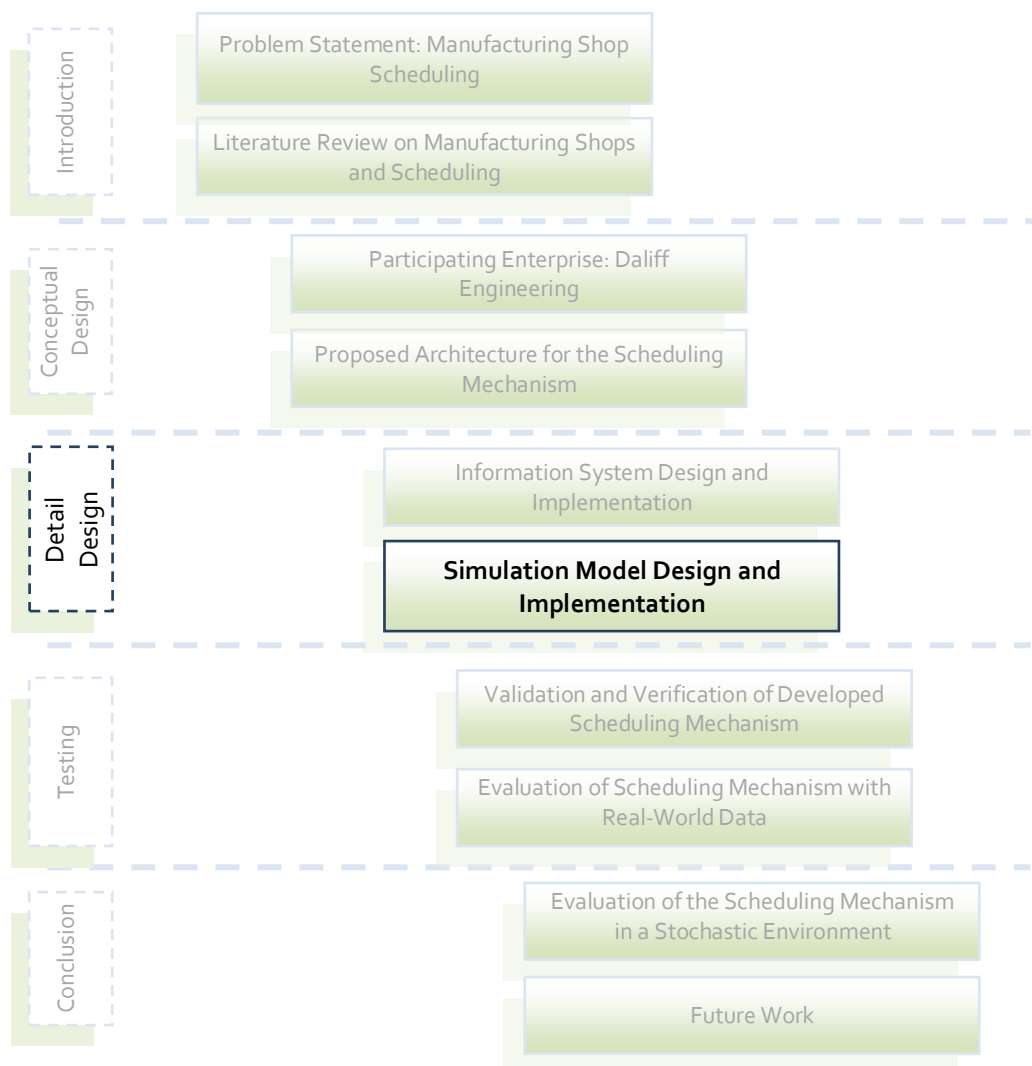
6.5 BRIEF OVERVIEW OF THE INFORMATION SYSTEM DESIGN AND IMPLEMENTATION

In this chapter the design and implementation of the information system that was developed to support the simulation scheduling model was described. The chapter started by stating what the function of the information system is, followed by a description of the relationships among entities through the use of an entity relationship diagram and relational data structure. The configuration was described through data flow diagrams, while the interface was shown using screenshots of the information system. The ASP code of the web based information system is included in Appendix II.

As previously stated, the information system acts as an input for the simulation model, which will be described in the next chapter.

7. DESCRIPTION OF THE SIMULATION MODEL

A simulation model has been developed as the next stage in the detail design phase of this study, shown on the thesis road map on this page, and to represent the process of manufacturing at Daliff Engineering. The process an order follows from order placement to delivery needs to be simulated as close as possible to the actual process of the real-world system. A schedule needs to be generated while the process is simulated. In this chapter the design and implementation of the simulation model is described.



As introduction to the detail description of the simulation model, the high-level concept of the simulation model will be explained in the next section.

7.1 HIGH-LEVEL CONCEPT OF THE SIMULATION MODEL

The information system populates the enterprise database, which serves as the input for the simulation model. The simulation model searches the data for orders that need to be processed, and creates a table with the information of these orders.

An order can have one or more than one part that must be manufactured, while the quantity of each part that must be manufactured can also differ. Each part has to go through several different operations during manufacturing. These operations are performed on specific machines, depending on the type of operation. The operations of a part follow a certain sequence and no two operations can be processed at the same time.

Figure 34 shows a typical order structure. The order has two different types of parts, requiring two units of Part 1 that must be manufactured, and one unit of Part 2. Part 1 requires four operations and Part 2 only three.

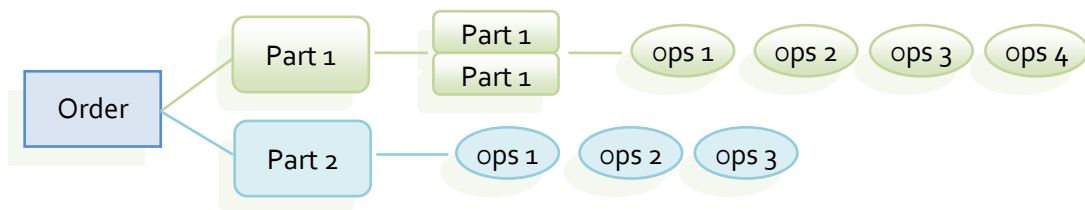


Figure 34 Order structure

The input for the simulation model is thus a table with a list of operations. Each operation is a record and has an order number and part ID. The other fields of the records are shown in Table 4.

Table 4 Fields of the Input Record

	Name:	Description:
1	Job_No	Order number
2	Status	Order status (Quote/Order/Finished)
3	ApprovedDate	Order accepted date
4	PromDate	Promised date of delivery (Due Date)
5	Part_ID	Part
6	Qty	Quantity of parts
7	Ops_No	Operation ID
8	Ops_Name	Operation Name
9	Mach_ID	Machine ID on which operation is performed
10	Mach_name	Machine Name

	Name:	Description:
11	Setup_time	Time to complete setup
12	Insp_time	Time to inspect part
13	Prod_Time	Time of processing operation
14	Status_ID	Status of Ops (Inactive/Active/Busy/Complete)
15	CSetup_time	Completed setup time
16	CInsp_time	Completed inspection time
17	CProd_time	Completed production time
18	Start_DateTime	Date and time Operation started
19	End_DateTime	Date and time Operation ended

The data of the first thirteen fields are generated by the information system and are static, except for the status field. The next six fields are dynamic fields, which are generated and updated by the simulation model and the shop floor database.

Before the scheduling mechanism activates the simulation model, the user must choose the scheduling rule which the simulation model must enforce. The rules in this study are the FIFO, LPT, SPT, EDD, SS, and CR rules, these rules were discussed in the literature study section, see page 28.

When scheduling is commenced, the simulation model generates entities that represent the operations in the resulting query table. An array is also constructed, called *EntityRecord* array, and all the information is also saved to this array. The array enables the simulation model to access entity (operations) information when entity attribute values are not available, this will be discussed later in this chapter.

The data in the fields of the table are assigned to the simulation entity as attributes. These attributes determine the flow sequence and processing time of the entity. It is important to understand that an entity is the representation of an operation on all the physical units of an ordered part. Figure 35 illustrates the entity structure. Pre-emption is not allowed, meaning that when an operation starts on a certain part, it must be completed before it is discharged from a machine.

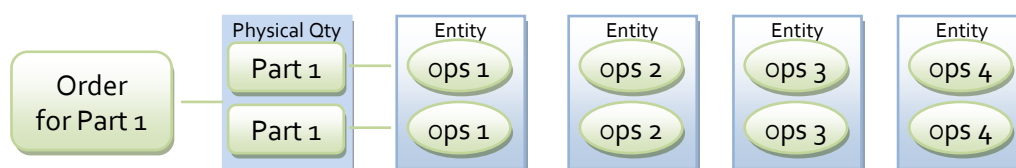


Figure 35 Entity structure

An operation must be completed on the complete set of physical parts before the next operation on this set of parts can start. The processing time of an entity at a machine is equal to the time it takes to complete the operation on the total number to be manufactured of the part, i.e. if ten of the same part have to undergo an operation, the processing time of the entity will be equal to ten times the processing time of the operation on one part. The processing time includes the setup time.

The attributes each entity has is stated in Table 5, the appropriate record field that is assigned to the attribute is included in the column next to the attribute column. The attributes that computed values are assigned to during simulation model execution, are indicated with (*Simulation model*) in the record field.

Table 5 Description of Entity Attributes

Attribute Name:	Record Field:
attrPartsize	(Simulation model)
attrEnt_Name	Ops_No
attrStartTime	Start_DateTime (Simulation model)
attrSlack	(Simulation model)
attrLocation	(Simulation model)
attrCInsp_Time	CInsp_Time (Simulation model)
attrTotMakespan	(Simulation model)
attrPart_ID	Part_ID
attrTime	Prod_Time
attrCProd_Process	(Simulation model)
attrProgress	Status_ID (Simulation model)
attrEnt_Successor	(Simulation model)
attrCSetup_Time	CSetup_Time (Simulation model)
attrCInsp_Process	(Simulation model)
attrProd_Start	(Simulation model)
attrCompletionTime	End_DateTime (Simulation model)
attrCR	(Simulation model)
attrTotTime	(Simulation model)
attrDD	PromDate (Simulation model)
attrSetup	Setup_Time
attrCProd_Time	(Simulation model)
attrSetup_Start	(Simulation model)
attrInspection	Insp_Time
attrInsp_Start	(Simulation model)
attrCSetup_Process	(Simulation model)
EntityStationAttribute	Mach_ID

An entity achieves an *Active* status when the operation before it in the sequence is completed or if it is the first operation of the sequence. The entities with an *Active* status can be processed. The entities whose predecessors have not been completed, have *Inactive* statuses. The reason for this is to enforce that the operations follow in the appropriate sequence. The fifth operation on a part for example can thus not start until the fourth operation ended and changed the status of the fifth operation to *Active*. The simulation model updates the status of the entities as the situation changes.

If an operation on a part is being processed when the simulation is started, a *Busy* status is assigned to the entity that represents the operation and is sent directly to the particular machine. The other entities are sent to the appropriate queues. The entities are queued according to the certain dispatching/scheduling rule chosen by the user at the beginning of a scheduling event.

The chosen scheduling rule discussed previously applies to these entities and not the parts itself. Each of the rules has a certain entity attribute that determines the rank of an entity. The EDD rule ranks entities according to the *attrDD* attribute of the entities. The *attrDD* attribute is calculated by the simulation model using the due date of the part the operation belongs to. The attribute is set to the due date of the part minus the processing hours of the remaining operations after the particular operation, i.e. $d_{i,j} = D_j - \sum_{r=i+1}^N p_{r,j,k}$, where N is the number of operations job j has.

The SS rule ranks entities according to the *attrSlack* attribute, which is calculated by subtracting the remaining processing time of the particular operation from the hours till its due date. The CR rule uses the *attrCR* attribute, which is calculated by dividing the remaining processing time of the part by the hours to the due date of the part.

The LPT and SPT rank entities according to the *attrTotTime* attribute of the entities, be it the longest for LPT or the shortest for SPT. The FIFO rule has no attribute for ranking entities, the entities are ranked according to the order in which they join the queues.

To ensure that active entities are not ranked behind inactive entities in the queues, dummy values for the attributes (which determines the queue rankings) are given to inactive entities. For example, if shortest processing time is the dispatch rule, an unpractical big value is given to the processing time attributes of the inactive entities, while the respective

actual processing time attributes are assigned to active entities. This ensures that the inactive entities are ranked behind the active ones, because the queues rank entities according to shortest processing times first. If the inactive entities were allowed to be ranked before active entities, the queue will be stationary until the inactive entity that is ranked first becomes active. This will imply that the resource will be idle whilst jobs that can be processed are waiting in the queue.

As stated previously it is possible to machine small parts on a machine that is meant to machine big parts. In other words, some parts that are assigned to a specific machine can also be processed on another machine, making alternative routing a possibility. This feature will not be included in the scheduler, because it is assumed that the cost of machining a small part with a big machine will be too high. However, the manufacturing engineer will be able to assign small parts to large machines if necessary. This could typically be the case when workload balancing is skewed towards the smaller machines.

When all the entities are placed either in queues or on machines, the simulation starts. It processes the entities that are on the machines and when applicable, takes an entity from the queue and start processing it. Entities can only be taken from the queue and put on a machine if the machine is idle, the entity has an *Active* status and is ranked first in the queue. When processing of an entity starts, the status of the entity gets changed from *Active* to *Busy*. The start time of the operation is saved in the entity start time attribute.

Each time an entity is completed its status is changed from *Busy* to *Complete*, its successor's status changes to *Active*. The end time of the operation is saved in the entity end time attribute. The queue in which the successor is, is reordered according to the dispatching/scheduling rule. This is needed as the status change of the successor (to *Active*), implies that the successor operation can now also be processed, thus it must receive a higher ranking.

The simulation model runs until all the entities are processed. The appropriate attribute values are recorded into a temporary table. The values are divided by the number of replications used in the simulation, which ensures that average values of the attributes for all the replications are stored. The next replication starts, which again uses the table constructed by the query to configure the system for the next replication run. After all the replications have been run, the average values of the replications from the temporary table

are used to construct a schedule. The proposed schedule resulting from the simulation run is presented in Gantt chart format in MS-Excel. The performance of the schedule is also recorded and can be compared to other schedules constructed under other dispatching/scheduling rules.

The simulation model must be implemented according to the high level concept that was described in this section. The implementation is described in the next section.

7.2 THE SIMULATION MODEL IMPLEMENTATION

The simulation model is implemented in the simulation software Arena (Rockwell Software [57]). The simulation software is chosen because it is available for education and research, while support was readily available. It also accommodates the discrete, stochastic nature of the system under study, and allows for customization through Visual Basic for Applications (VBA) on the Microsoft-platform.

The simulation model was implemented as two components, namely the Arena Model and the VBA code. The Arena model represents the configuration of the enterprise, while the VBA code is the customization of the configuration according to the current state and orders that need to be processed. In the next few sections a detail description of each will be given. These two components act together to compile an output file, which will also be discussed in this chapter.

7.3 ARENA COMPONENT OF THE SIMULATION MODEL IMPLEMENTATION

The number of resources, the flow of orders and the seizing of operators are some of the configurations that are implemented by the Arena model.

The entity flow is briefly explained here, followed by a detail discussion in the subsequent sections. Entities are created and attributes get assigned according to the enterprise database. A picture is then assigned to the entity before it is held in a *Hold* module until all the entities have been created. The entities are then sent to their particular machines, where they either join the queue or if their status is busy, put on the machine. The entities are dispatched from the queue as the dispatching criteria are met. The entities are processed and when finished sent to the part assembly station, where entities are batched according to the part they belong to. As processing of the entities finishes, their successors'

states change and the successors can now compete for machine time. When all the entities (which represent operations of a part) are processed, the part is completed and the entity disposed.

7.4 PHYSICAL CONFIGURATION OF THE SIMULATION MODEL

The model can be divided into the following sections: the entity creation and attribute assignment, the machine station and queuing, the part assembly and the statistic recording sections. These are now discussed.

7.4.1 Entity creation and attribute assignment

In this section of the simulation model, see Figure 36, entities are created using the *Create* module. The VBA code modifies the creation module where entities will be created for all the operations from the query table. The procedure the code follows will be explained in the section about the VBA code. Attributes are assigned to each entity by an *Assign* module. The Attribute values are set to zero as a *VBA block* will assign them the appropriate values.

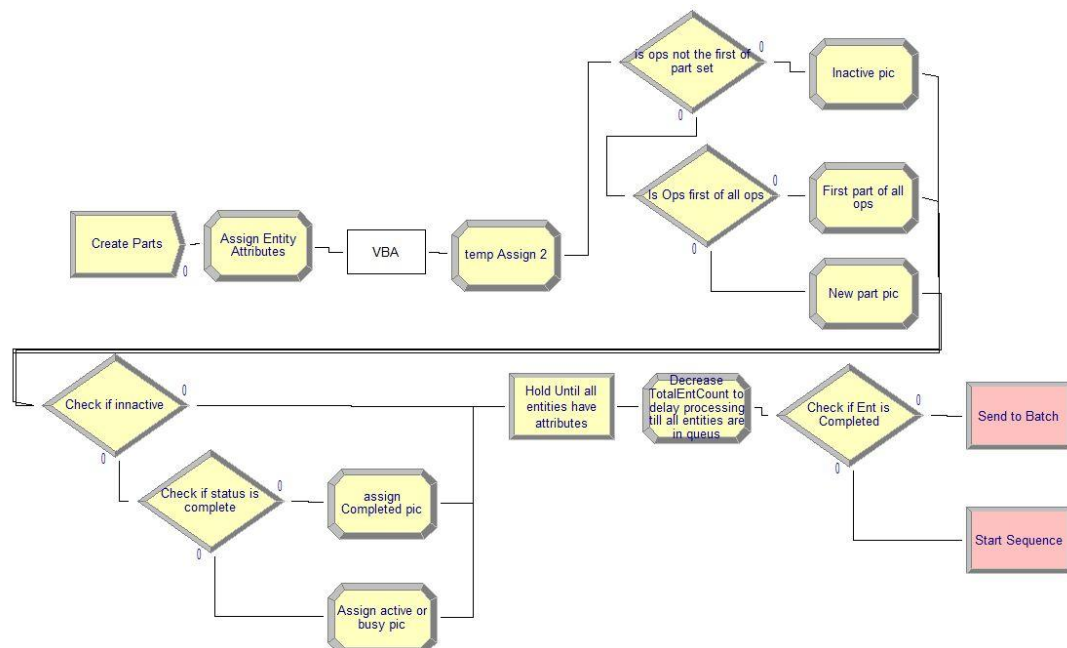


Figure 36 Diagram of entity creation and attribute assignment component of the simulation model

For animation, the entity pictures are assigned according to their appropriate part set. Each part is assigned with a certain shape. All the entities of the part have the same shape for its entity picture. Their colour represents their status: red for *Inactive*, yellow for *Active* and

Busy and green for *Complete*. The first of two *Decide* module combinations determine the picture shape and the second its colour.

7.4.2 Machine station and queuing

There is a machine station for every resource of the enterprise in the model, which can be divided into two components, namely queuing and processing, see Figure 37. In the queuing part, the scheduling rules chosen by the user are implemented and entities are queued accordingly, whilst the processing part processes the entities on the machines.

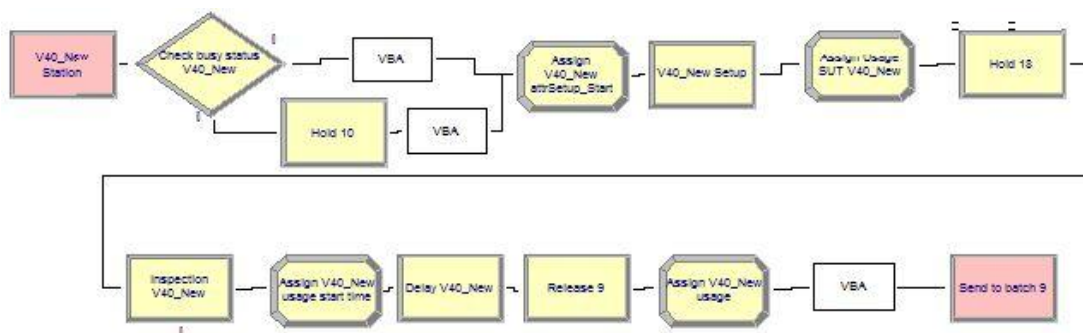


Figure 37 Diagram of machine station and queuing component of the simulation model

When an entity arrives at this section the status of the entity is firstly checked. If its status is *Busy*, it is directly sent to the processing part of the section. It calls VBA code that assigns the appropriate processing times to the entity attributes. If it is *Inactive* or *Active* it will go to the queuing part of the section. As previously discussed, dummy values are assigned to the attribute (that determines the entity rank in the queue) of the inactive entities and actual values to those of the active entities. The queuing part queues the entities and holds the entities until the machine becomes free and their entity status are *Active*. The queue discipline is set by the VBA code at the beginning of the simulation run according to user specification. As the entity is released from the queue it calls VBA code that changes the entity status form *Active* to *Busy* and records the start time of the processing.

In the processing part of the machine station and queuing section the setup, inspection and manufacturing processes are conducted. The start and processing times of each are recorded as entity attributes. The usage of each machine is also recorded in a variable array. The setup start time attribute is recorded through an *Assign* module after which the setup process seizes the particular machine and delays the entity for the setup duration. The actual setup time is recorded by the next *Assign* module. This *Assign* module also records

the usage of the machine and indicates that the setup process is completed by assigning a True value to the appropriate attribute. It also records the start time of the inspection process. The entity is kept in a *Hold* module until inspection can be done. The inspection resource is seized, delayed for the inspection process time and released. The actual inspection time is recorded by the next *Assign* module, which also records the start time of the production process and sets the appropriate attribute to True. The entity is delayed for the production time and the machine is released afterwards. The final *Assign* module records the actual production time, machine usage and sets the *attrCprod_Process* attribute to True.

The entity then calls VBA code that changes the status of the entity to *Complete* and the status of its successor to *Active*. The picture of the successor is changed and the queue that contains the successor is rearranged by the VBA code. The makespan of the entity is also recorded. If the work has been done on the entity before the simulation started, it is added to the simulation time. The actual processing times are also recorded into the temporary simulation table. The entity is then sent to the part assembly section.

7.4.3 Part assembly

The part assembly section consists of a *Batch* module, see Figure 38. The entities are batched according to their part ID. The last entity of a part, that resembles the last operation on the part, has an attribute *attrPartsize* that determines the number of entities that the particular part consists of. The batched entity adopts the attributes of the last entity of the part.

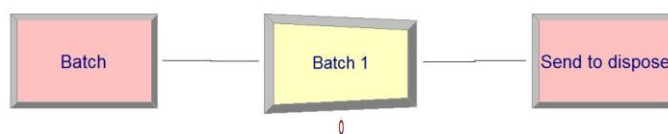


Figure 38 Diagram of the part assembly section of the simulation model

7.4.4 Statistic recording

The entity calls VBA code through a *VBA block* module, see Figure 39. The code determines if the part is early or late and quantifies the outcome. The part statistics like completion time, lateness and earliness are written to the Excel results file. The total makespan is recorded for further statistics. The part, that resembles a few entities, is now disposed.

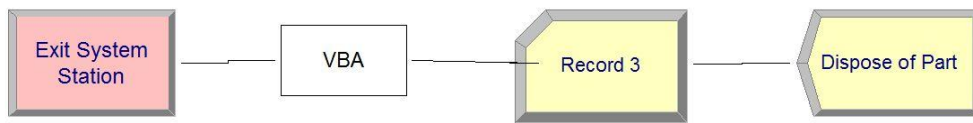


Figure 39 Diagram of the statistic recording section of the simulation model

Next the customization of the simulation model according to the current system state as implemented in Visual Basic for Applications is described.

7.5 VBA CODE COMPONENT OF THE SIMULATION MODEL IMPLEMENTATION

The VBA code customizes the simulation model according to the current system state when the simulation is started. The code can be activated through the simulation model run logic and VBA blocks in the Arena model. The functions of the procedures that are called by the model run logic are discussed in this section. The custom codes called by the model run logic and VBA blocks are discussed in Appendix III. All the VBA code that the simulation uses is included in Appendix IV.

It is required to discuss how the procedures are called and what information is available when doing so, before the procedures itself can be discussed. The description of how the model run logic functions follows, the structure of the model run logic is shown in Figure 40.

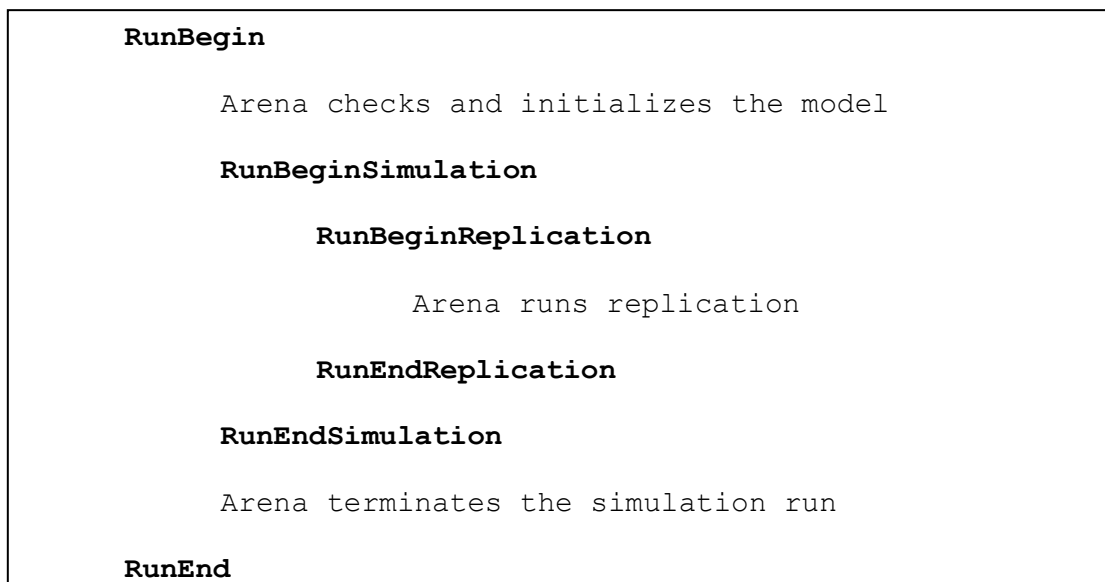


Figure 40 Model logic structure

Module data is the information that was defined in the Arena model and is available when the simulation is not running. Thus, the module operands values are available when the simulation is not running. Simulation run data is only available when the simulation run has started. When the simulation run is started, Arena checks and initializes the model, translating the information provided in the modules in the Arena model into the format required to perform the simulation run. During the run, the values of variables, attributes, resource states, etc., can be examined and changed through Arena's run controller and VBA code.

The simulation model logic statements, referred to as events from here on, is subsequently discussed.

7.5.1 Model logic Events procedures

RunBegin

A form is displayed that prompts the user to choose a certain scheduling rule. After the selection, the *startModel* subroutine is called.

RunBeginSimulation

The Excel results workbook is opened and cleared due to this event.

RunBeginReplication

The *determineCR* subroutine is called to determine the critical ratios of all the entities. The *TotalEntCount* Arena variable is assigned by the *AssignTotalEntCount* subroutine. The event then runs through the records of the simulation query record set. For each record the appropriate values are assigned to the entity records array through the *AssignEntArray* subroutine. The successors are also assigned by calling the *AssignEntSuccessor* subroutine.

RunEndReplication

The state of the system as the replication ends, is recorded by calling the *UpdateSnapShotAtEndRep* subroutine. This is included when the replication length is not set to the length to complete all the operations, which in theory will not happen often as a simulation is run to schedule all the operations.

RunEndSimulation

The simulation query recordset is updated by calling the *UpdateSimTableFromTempTable* subroutine. The simulation run output statistics are written to the Excel worksheet.

The start and end time attributes of each entity is written to the Excel file. Eight loops, each representing a machine, scan through the temporary simulation table to find entities that were processed on the machine currently under consideration. The part ID the entity belongs to, the entity name, the entity start time, the entity end time and the machine ID are written to the Excel worksheet.

Before the start and end times are written, their values are checked. If any of the values are less than zero, it means that the process started, or started and ended before the start of the simulation run. In the case that the start and end time are less than zero, both values are set to zero. When only the start time is less than zero, its value is set to zero and the end time is set to the value of the *EntityRecord* array value. If in this case the end time in the *EntityRecord* has no value, processing of the entity has not finished. The end time must be set to the end time of the replication run, so that the schedule will display the entities that are still being processed at the replication end.

The chart in the Excel results worksheet (see Figure 41 on page 76) that represents the schedule is also modified by this event. The colours of the bars are changed so that all the operations of a part have the same colour and no one part has the same colour as another. This enables the user to clearly see when a new operation or the next operation on the same part starts. The operation name is also written in the applicable bar. These modifications were implemented using VBA code that adjusts a previously constructed chart that was constructed according to certain source data.

RunEnd

The entities that were created for the simulation model is deleted by calling the *CleanEntities* subroutine. The event further closes all the applications that were opened by the simulation model.

In this section, the customization of the simulation model was discussed briefly. References were made to appendices included in this thesis for further explanation of the customization codes. Next, the output created by the simulation model is discussed.

7.6 SIMULATION OUTPUT FILES

The output files created by the simulation model are used for analysis. Arena generates its own statistics output file, but it is not sufficient to set up a schedule. The data in the file is used in the customized output file that is generated by the simulation model. The model output file is an Excel workbook, which has a worksheet for each type of scheduling rule and a worksheet for result comparison. The attribute values for each entity are written to the appropriate worksheet as the entities are processed by the simulation model. The structure of the entity attributes can be seen in Table 6. Each entity, that represents an operation, has a Part ID, Ops No, Start time, End time and a Machine ID which are written to the Excel file.

Table 6 Structure of recorded entity information

Part_ID	Ops_ID	Start Time	End Time	Machine
1	1	0	6.5	8
2	2	0	2.5	7
3	3	0	3.75	8
4	4	0	19.5	2
5	5	0	14.25	3
5	6	14.25	17.75	3
6	7	0	58.5	1
7	8	0	15.75	8
7	9	19.25	27	8
8	10	0	5	7
9	11	3.75	4.5	8
9	12	9.5	15	8
10	13	4.5	6	8
10	14	13	14	8
10	15	20	21	8
11	16	6	13	8
12	17	6.5	8	8
12	18	14	20	8
12	19	28.5	30.5	5

The duration of the operations is calculated by the Excel file and a bar chart is constructed from this data. The VBA code adjusts the chart as described in section 7.5.1 under the *RunEndSimulation* event. This bar chart represents the schedule and an example of it is shown in Figure 41.

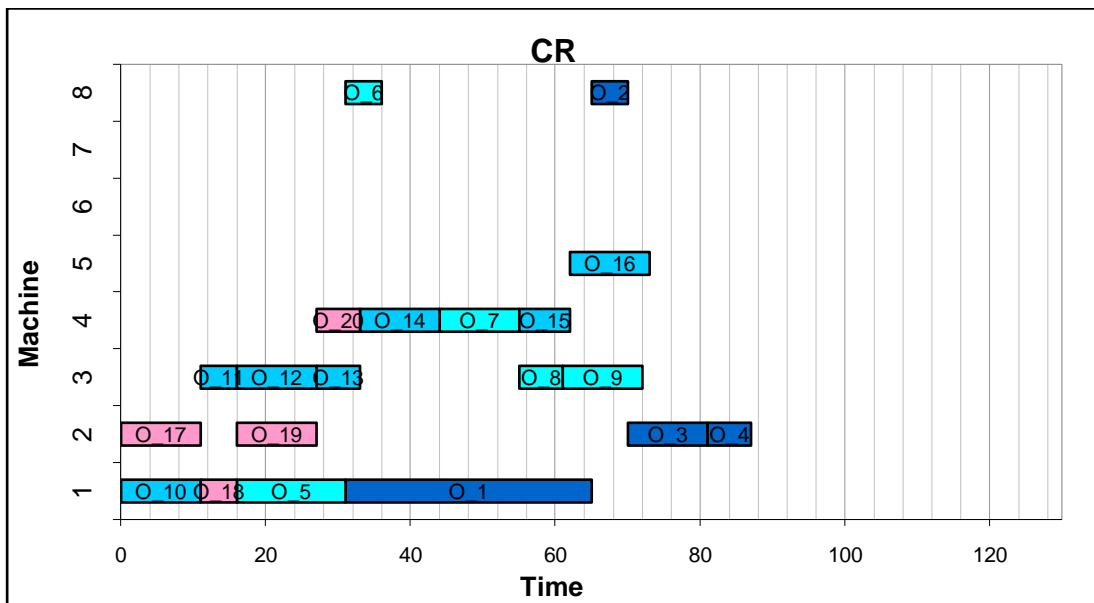


Figure 41 Example of a schedule that is developed in the output file

The information about each part is also written to the worksheet as parts (represented by batched entities) are disposed from the model. The structure is shown in Table 7. The part ID and its particular Due date, End time, Hours late and Hours early are written to the worksheet. This information is written when processing of all the entities of a part is complete. The entities are batched and sent through a VBA *block* module.

Table 7 Structure of recorded part information

Part_ID	DueDate	End Time	Hours Late	Hours Early
1	5	6.5	1.5	
2	15.5	2.5		13
3	10	86.25	78.25	
4	139.5	19.5		120
5	139.5	17		122.5
6	58.5	58.5		0
7	58.5	75.5	17	
8	13.5	17.25	3.75	
9	13.5	108	94.5	
10	13.5	99.75	86.25	
11	13.5	82.5	69	
12	13.5	97.25	83.75	

The output statistics of Arena are also written in the Excel worksheet representing the relevant scheduling rule. The makespan, total earliness, total lateness and average flow time of the schedule and the average usage of each machine are recorded.

The comparison worksheet contains the output statistics of the schedule under each scheduling rule. Bar charts are compiled to compare the different statistics with each other. It is then possible to visually compare scheduling rule performances per measuring criteria. Figure 42 to Figure 46 illustrate the comparison bar charts of the performance measures, the y-axis state which performance criterion is applicable, the x-axis is a measure in hours or percentage and each bar has a name which shows the scheduling rule it represents.

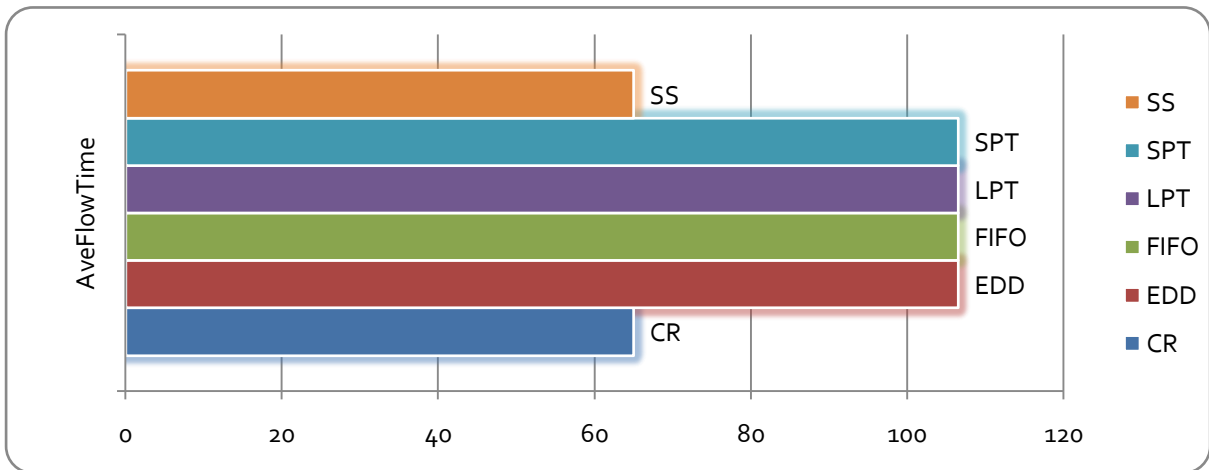


Figure 42 Typical average flow time comparison bar chart

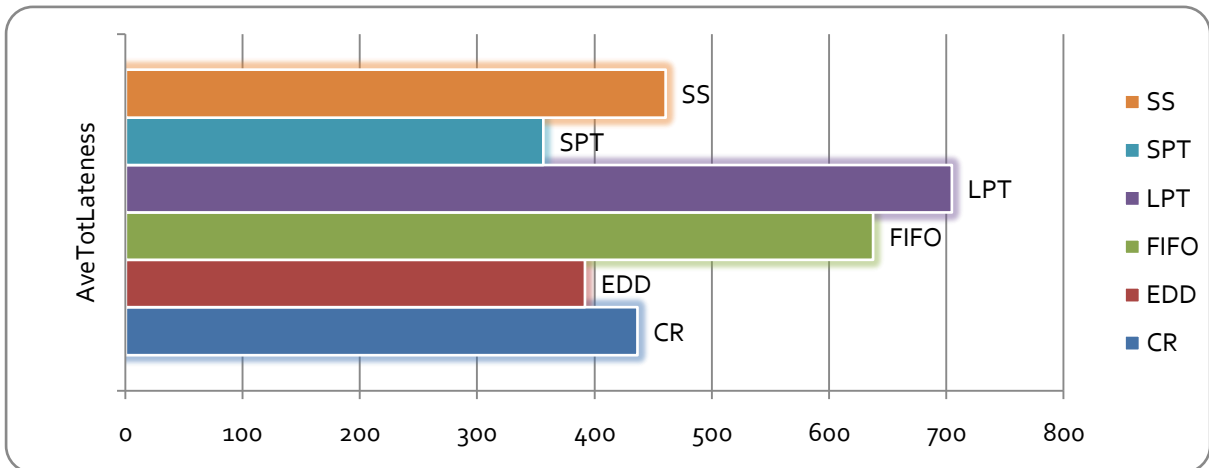


Figure 43 Typical average total lateness comparison bar chart

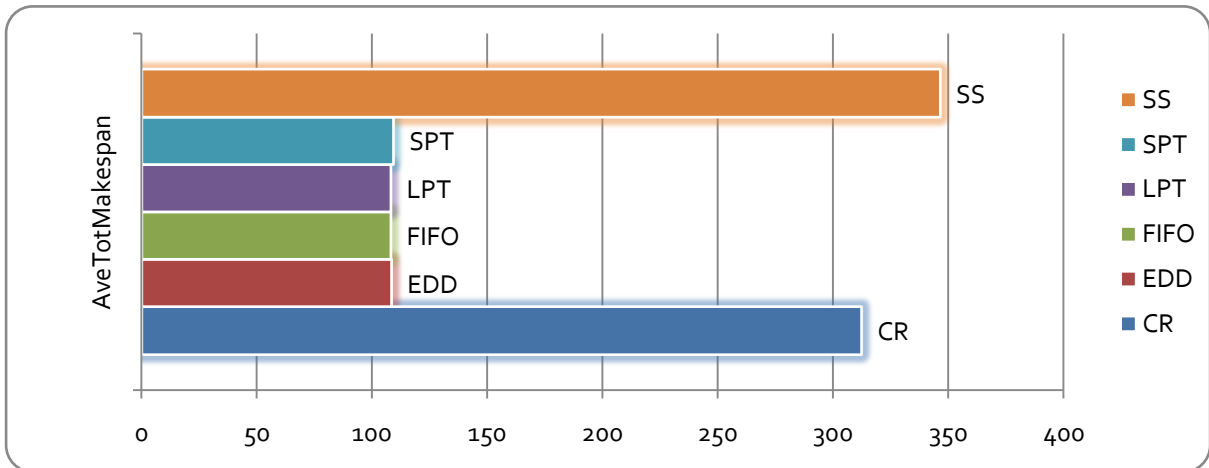


Figure 44 Typical average total makespan comparison bar chart

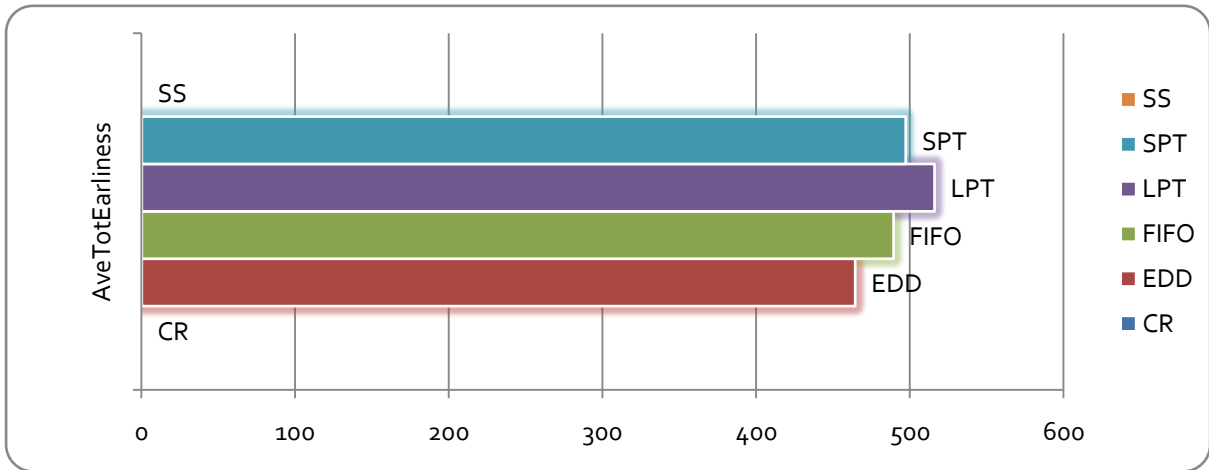


Figure 45 Typical average total earliness comparison bar chart

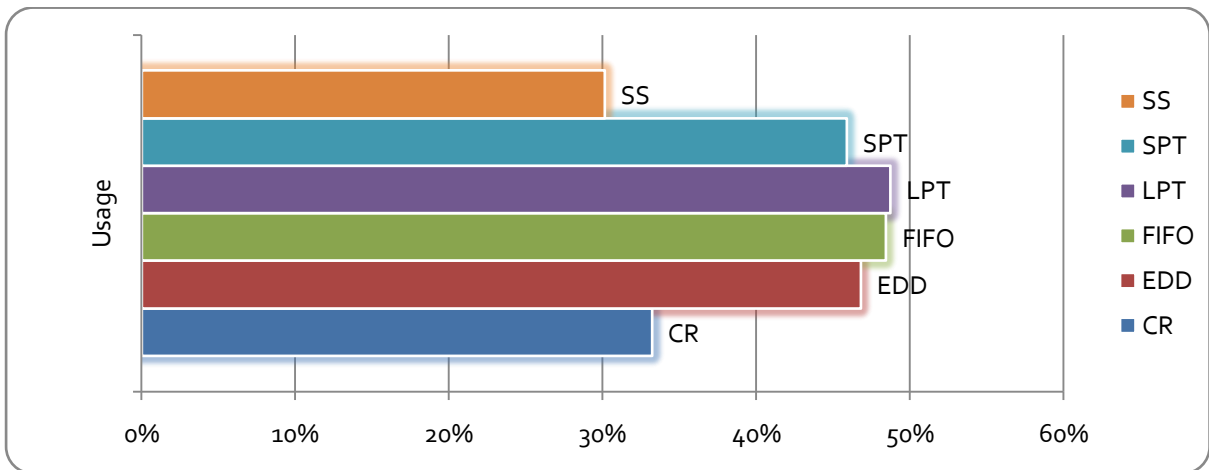


Figure 46 Typical usage comparison bar chart

Depending on the preference of the user, the appropriate schedule is chosen. Take for example the figures above, the SPT rule tends to have the best combined result. Compared to the performance of the other rules it has a short makespan, the total late hours is the least, the total early hours is relatively large and its usage is relatively high .

7.7 CHAPTER OVERVIEW

This chapter described the simulation model that was developed in the study. The description started with an overview of the concept of the simulation model, which was described by referring to the composition of orders, simulation input and entity description. The configuration of the simulation model was then described by explaining what the functions of the two components of the simulation model are. The two components are the Arena model and the Visual Basic for Application code. Both were described on a technical level. The output file that is created by the simulation model is also described in this chapter.

The next step is to verify and validate the simulation model supporting the scheduler, and the scheduler itself. The next chapter describes the validation and verification process that was followed.

**8. VALIDATION AND VERIFICATION OF THE SCHEDULING MECHANISM
FUNCTIONALITY**

To be able to justify the results of this study, the components of the scheduler that determine the proposed schedule need to be verified and validated. In this chapter the verification and validation process is discussed, which is cardinal to the testing phase of this study as shown in the thesis road map on this page. The aim of this chapter is not to prove that the scheduler functions perfectly as it is not feasible to investigate the scheduler's response in every possible system configuration. This chapter aims to build confidence in the function of the scheduler and to create credibility so that it can be implemented.



Verification can be defined as asking the question: "Was the model built right?", i.e. does the model function as intended? Whereas validation is asking the question: "Was the right model built?", i.e. is the model an adequate representation of the real-world system?





Verification focuses on the correctness of the model and includes actions such as inspecting logic, correcting syntax errors, and correcting run-time errors. Throughout the development of the model, verification was done as syntax and run-time errors were fixed as they appeared. The logic of the model was examined by simulating several scenarios and evaluating the result of each of them. Validating the model can also be seen as testing the model under extreme conditions, to see if the model still operates correctly.

Different scenarios were developed to test the operation of the model under normal and extreme conditions. The scenarios that were simulated have some similar attributes, namely:

- Four Orders consisting of one part each: $j = \{1 \dots 4\}$
- Total of 20 operations: $O_{i,j}$, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$
- The same release dates for all the orders: $r_j = 2008/06/25$, where $j = \{1 \dots 4\}$

The composition of the orders is shown in Table 8, the orders are listed each with its part and the operations of each part. The colour that represents a part in the schedule is also included. To be able to display which operation is which on the graph an adjusted version of the operation notation was used in the schedules. Instead of using $O_{i,j}$, O_n was used to represent operation n , Table 8 shows how the new format relates to the old format.

Table 8 Order Composition for Validation

Order	Part	Colour	Operations
1	1		$O_{1,1}; O_{2,1}; O_{3,1}; O_{4,1}$ $O_1; O_2; O_3; O_4$
2	2		$O_{1,2}; O_{2,2}; O_{3,2}; O_{4,2}; O_{5,2}$ $O_5; O_6; O_7; O_8; O_9$
3	3		$O_{1,3}; O_{2,3}; O_{3,3}; O_{4,3}; O_{5,3}; O_{6,3}; O_{7,3}$ $O_{10}; O_{11}; O_{12}; O_{13}; O_{14}; O_{15}; O_{16}$
4	4		$O_{1,4}; O_{2,4}; O_{3,4}; O_{4,4}$ $O_{17}; O_{18}; O_{19}; O_{20}$

The difference in the scenarios is in terms of processing times, due dates, and number of machines. These differences and the discussion of the resulting schedule of each scenario follows:

Scenario 1A:

- One machine: $k = 1$
- Same processing times: $p_{i,j,k} = 11 \text{ hours}$, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$

$p_{i,j,k}$	i							Total
	1	2	3	4	5	6	7	
1	11	11	11	11				44
2	11	11	11	11	11			55
3	11	11	11	11	11	11	11	77
4	11	11	11	11				44

- Different due dates: $d_1 = 2008/08/02$
 $d_2 = 2008/07/25$
 $d_3 = 2008/08/01$
 $d_4 = 2008/07/23$

Scenario 1A illustrates that the functionality of the simulation model of enforcing the FIFO, EDD, SS, and CR scheduling rules are correct. Figure 47 shows the resulting schedules for *Scenario 1A* under the different scheduling rules.

The proposed schedule of the FIFO rule correctly schedules the operations one after another as the operations become active. Looking at the colour schemes of the operation, one can see that the operations follow the sequence of the parts. The first operation of each part is processed, in order from *Part 1* to *Part 4*, the second operation of each part then follows (also in order from *Part 1* to *Part 4*), the schedule continues to be generated in this manner. The three operations ($O_{14}: p_{5,3,1} = 11$, $O_{15}: p_{6,3,1} = 11$, and $O_{16}: p_{7,3,1} = 11$) that *Part 3* has extra are added at the end of the schedule as they only become active after the last operation of *Part 4* became active.

The proposed schedules of the LPT and SPT rules are not discussed, as they are the same as the schedule of the FIFO rule, because the processing times of all the operations are the same and the rules have no effect.

The schedule developed under the EDD rule shows that the processing of parts is finished in the following order: *Part 4*, *Part 2*, *Part 3*, and *Part 1*. This is the same order in which the due dates follow on each other from the nearest date to the furthest. As the model started, operations $O_{17}: p_{1,4,1} = 11$, $O_{18}: p_{2,4,1} = 11$, $O_{5}: p_{1,2,1} = 11$ and $O_{10}: p_{1,3,1} = 11$ were all active. $O_{17}: p_{1,4,1} = 11$ was processed first as its due date is the earliest, after its processing was finished $O_{18}: p_{2,4,1} = 11$ also became active with the other active operations. $O_{5}: p_{1,2,1} = 11$ was started next as its due date was earlier than $O_{18}: p_{2,4,1} = 11$, although the due date of the part $O_{18}: p_{2,4,1} = 11$ belonged to was earlier than the part $O_{5}: p_{1,2,1} = 11$ belonged to. This is a result of the due dates of the operations that are not the same as the due date of the part they belong to (see Section 7.1, page 67). The rest of the operations are scheduled similarly.

The proposed schedule of the SS rule shows that the processing of parts is finished in the same order as the EDD rule. The difference though, is that the parts are finished as a whole before an operation on the following part starts. This happens because the slack is the same for all the operations of a particular part (see Section 7.1, page 67). In this scenario *Part 4* has the smallest slack as the difference in due dates is bigger than the difference in total processing time of the parts, i.e. the hours that d_4 is before d_3 is greater than the hours p_3 is longer as p_4 , making the slack of *Part 4* smaller than the slack of *Part 3*. Processing of $O_{17}: p_{1,4,1} = 11$, the first operation of *Part 4*, starts first followed by $O_{18}: p_{2,4,1} = 11$, the second operation of *Part 4*. All the first operations of all the parts are also active, but because $O_{18}: p_{2,4,1} = 11$ is an operation of *Part 4*, which has the smallest slack, it is processed before the other operations that were active before them.

Finally, the schedule developed under the CR rule shows that the rule is correctly implemented. The schedule starts with $O_{10}: p_{1,3,1} = 11$, the first operation of *Part 3* which has the longest total processing time ($7 \times 11h$). As operations of *Part 3* are finished, the remaining processing time and consequently the critical ratio of *Part 3* decreases, until the first three operations are finished, making the critical ratio of *Part 3* smaller than the ratio of *Part 2*. Processing on the first operation of *Part 2* is then started.

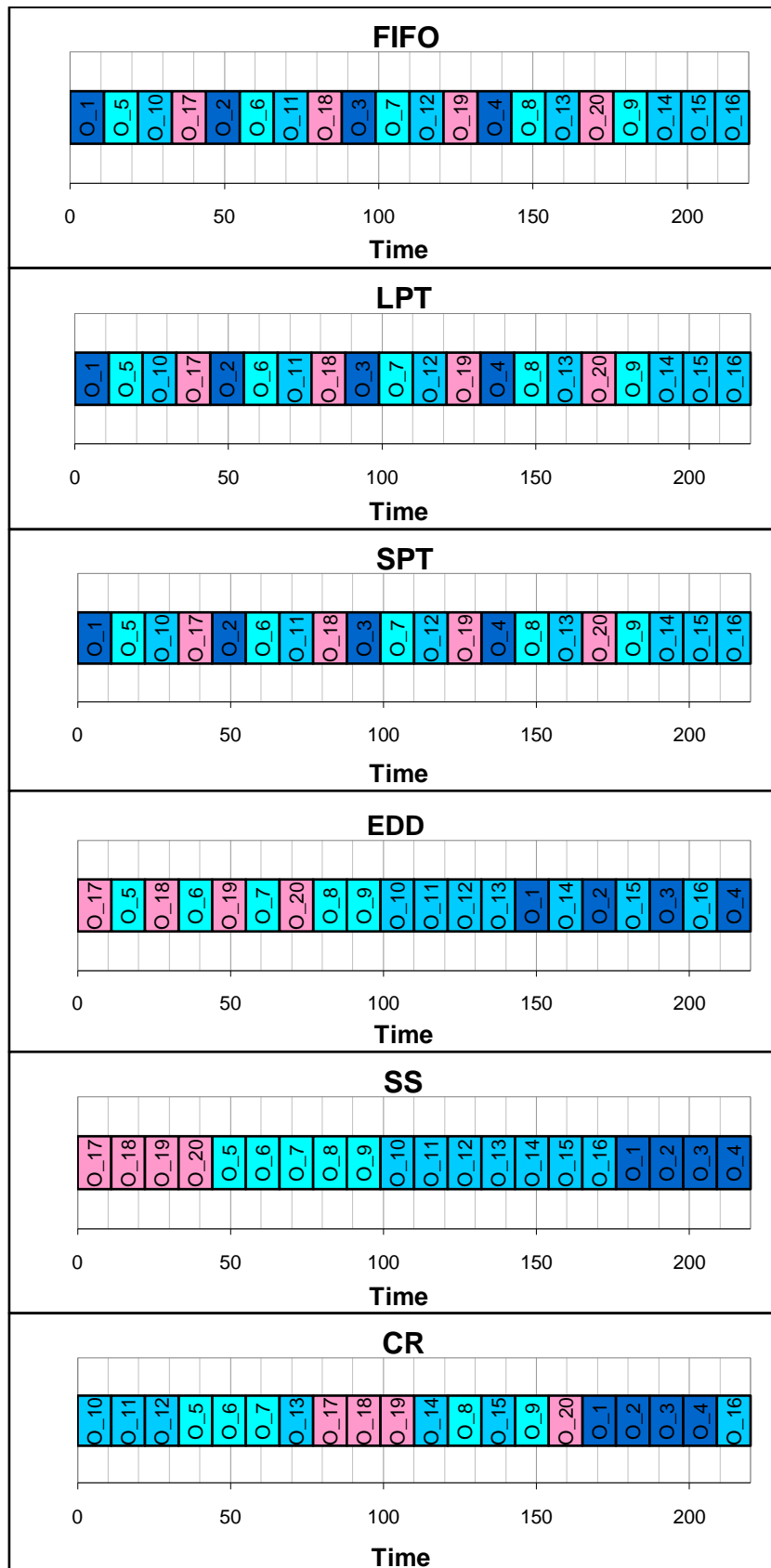


Figure 47 Scenario 1A: Resulting Schedules

Scenario 1B:

- Two machines: $k = 1, 2$
- Same processing times: $p_{i,j,k} = 11$ hours, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$

$p_{i,j,k}$	i							Total
	1	2	3	4	5	6	7	
1	11	11	11	11				44
	$k = 2$	$k = 1$	$k = 2$	$k = 1$				
2	11	11	11	11	11			55
j	$k = 2$	$k = 1$	$k = 1$	$k = 1$	$k = 2$			
3	11	11	11	11	11	11	11	77
	$k = 1$	$k = 1$	$k = 2$	$k = 2$	$k = 2$	$k = 1$	$k = 1$	
4	11	11	11	11				44
	$k = 2$	$k = 2$	$k = 1$	$k = 2$				

- Different due dates: $d_1 = 2008/08/02$
 $d_2 = 2008/07/25$
 $d_3 = 2008/08/01$
 $d_4 = 2008/07/23$

Scenario 1B is similar to Scenario 1A, the difference is that Scenario 1B has two machines. The proposed schedules are shown in Figure 48, they confirm that the scheduling mechanism correctly enforces the scheduling rules. The proposed schedules from the FIFO, LPT and SPT rules are the same again, this is correct as the processing times of the operations are the same. Note that the makespan is now half of that of Scenario 1A. The rest of the schedules differ because the due dates are not the same.

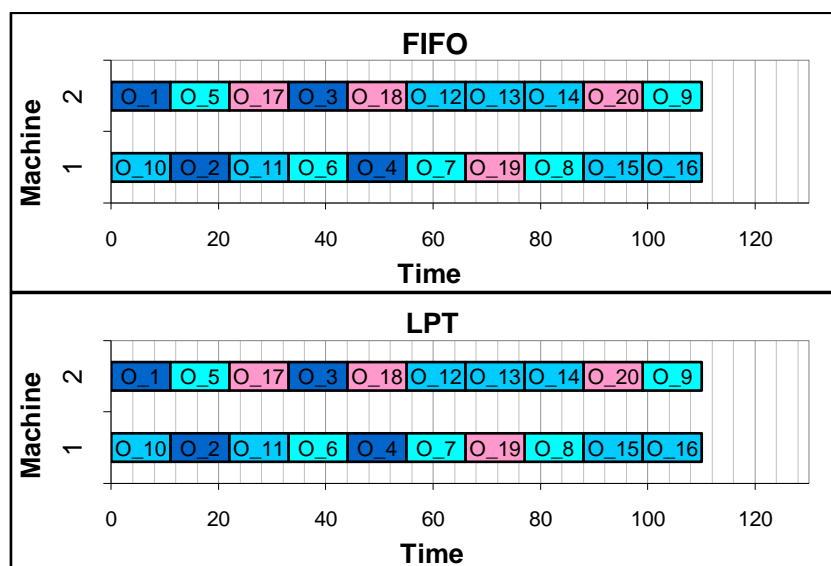


Figure 48 Scenario 1B: Resulting Schedules (Continued on next page)

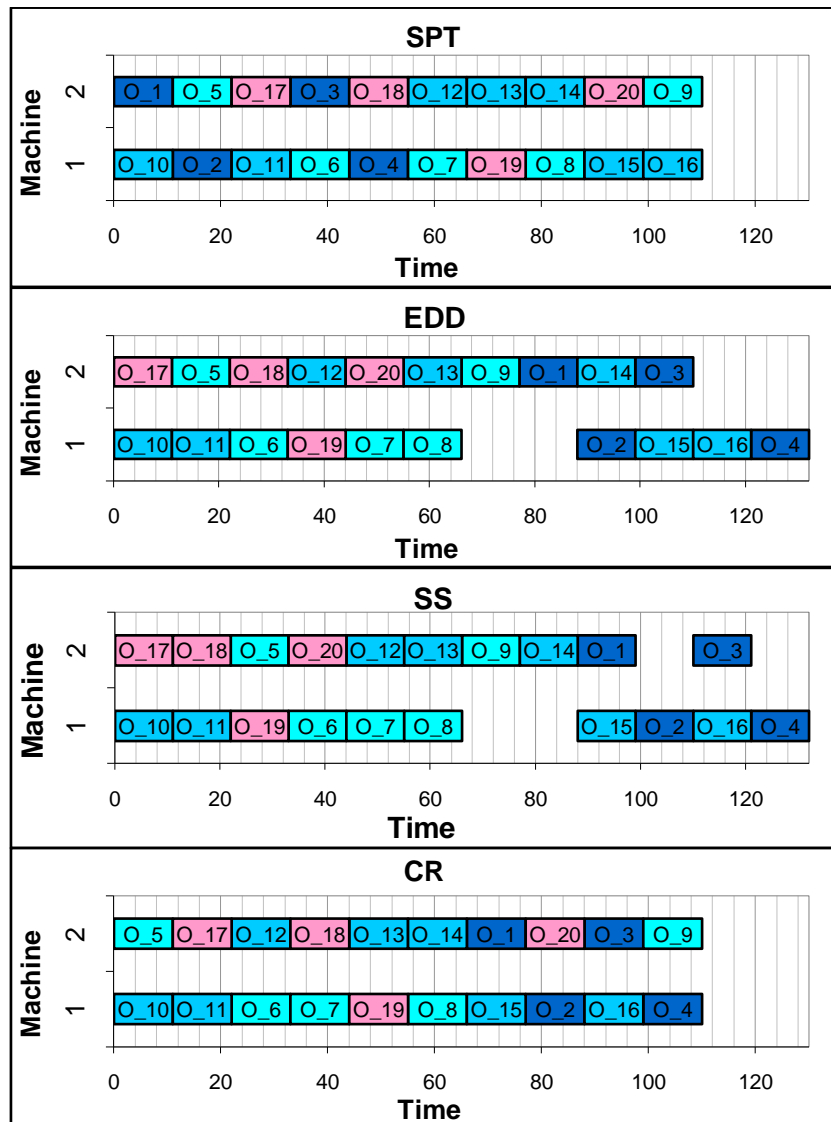


Figure 4.8 Scenario 1B: Resulting Schedules

Scenario 2A:

- One machine: $k = 1$
- Different processing times $p_{i,j,k} > 0$, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$

$p_{i,j,k}$	i							Total
	1	2	3	4	5	6	7	
j	1	11	5	11	6			33
	2	11	5	11	6	11		44
	3	11	5	11	6	11	7	62
	4	11	5	11	6			33

- Different due dates: $d_1 = 2008/08/02$
 $d_2 = 2008/07/25$
 $d_3 = 2008/08/01$
 $d_4 = 2008/07/23$

Scenario 2A confirms that all the scheduling rules are correctly implemented, see Figure 49. This scenario is similar to *Scenario 1A* as it has the same due dates and only one machine, while the difference is in the processing times. This implies that the functionality of the rules concerned with the processing times (LPT and SPT) now also gets tested, not like in the first two scenarios. The order in which the parts finish under the FIFO and EDD rules is the same as the order it finished in the first two scenarios, as the arrival times of the parts and their due dates are the same.

The LPT rule had an impact from the start, after $O_{-1}: p_{1,1,1} = 11$ finished, $O_{-2}: p_{2,1,1} = 5$ also became active with the first operations of the other parts ($O_{-5}: p_{1,2,1} = 11$, $O_{-10}: p_{1,3,1} = 11$, and $O_{-17}: p_{1,4,1} = 11$), but seeing that these operations had longer processing times than $O_{-2}: p_{2,1,1} = 5$, they were processed first. When $O_{-2}: p_{2,1,1} = 5$ finished, $O_{-3}: p_{3,1,1} = 11$ also became active with the second set of operations of the other parts ($O_{-6}: p_{2,2,1} = 5$, $O_{-11}: p_{2,3,1} = 5$, and $O_{-18}: p_{2,4,1} = 5$), but seeing that $O_{-3}: p_{3,1,1} = 11$ had a longer processing time than the other active operations, it was processed first. The rest of the schedule was conducted in similar fashion.

The SPT rule processed $O_{-2}: p_{2,1,1} = 5$ directly after $O_{-1}: p_{1,1,1} = 11$, before the first operations of the other operations ($O_{-5}: p_{1,2,1} = 11$, $O_{-10}: p_{1,3,1} = 11$, and $O_{-17}: p_{1,4,1} = 11$), as its processing time was shorter than those of the other active operations. $O_{-5}: p_{1,2,1} = 11$ was processed next as its processing time was the same as the other active operations, including the newly added $O_{-3}: p_{3,1,1} = 11$, but entered the queue before all of them. The SPT rule affected the rest of schedule in similar fashion.

The order in which the parts are finished in the EDD and SS schedules, is the same as for these schedules in *Scenario 1A* and their discussion thus are very similar. The order in which the operations are processed in the EDD schedule differs from *Scenario 1A*, because the difference in processing times means that the operations do not have the same due dates. $O_{-18}: p_{2,4,1} = 5$ (*Part 4*) for example has an earlier due date than $O_{-5}: p_{1,2,1} = 5$ (*Part 2*), in contrast with *Scenario 1A* where it was the other way round. The reason for this is that

although the processing times of both parts are shorter, the processing time of *Part 2* decreased more relative to its processing time in *Scenario 1A* because it has making the margin the due date is later bigger than the margin of *Part 4*.

The SS schedule is the same as in *Scenario 1A*, this can be justified with the fact that the processing times do not differ enough to change the slacks in such a way that the schedule differs. The CR rule has different attribute values than in *Scenario 1A* because of the difference in processing times, resulting in a different schedule than in *Scenario 1A*.

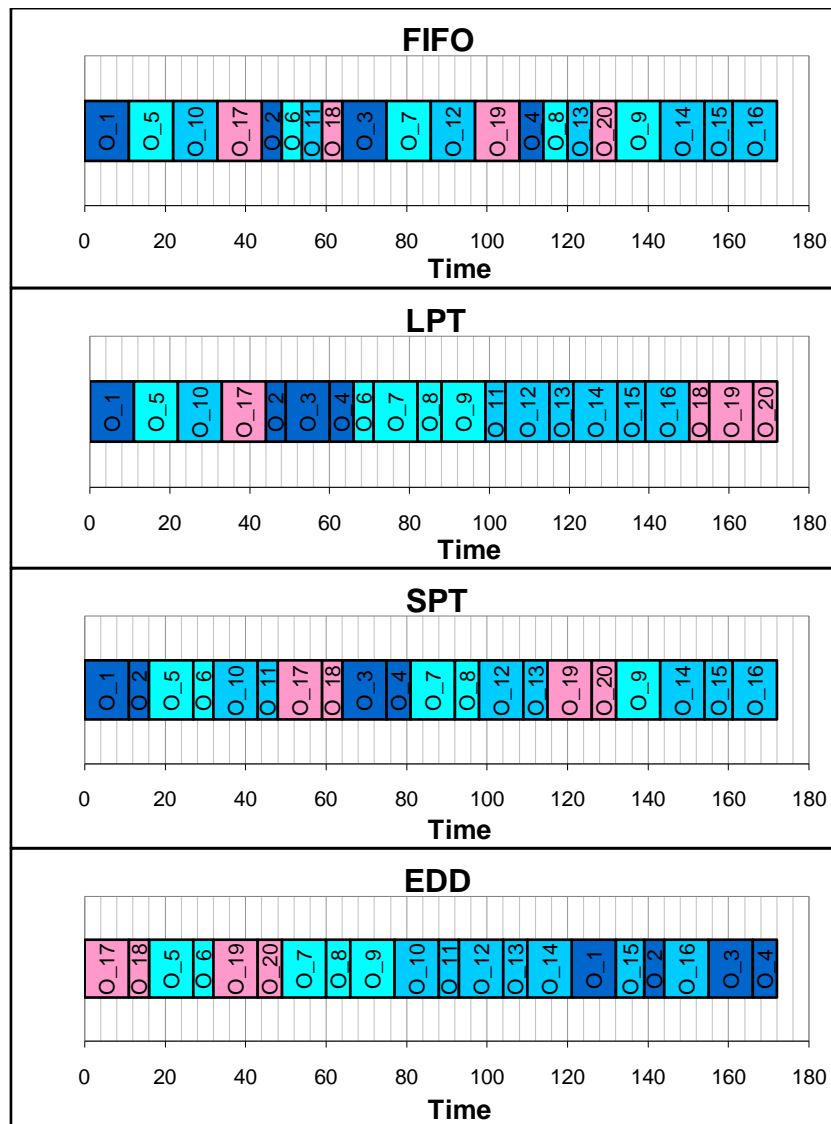


Figure 49 *Scenario 2A*: Resulting Schedules (Continued on next page)

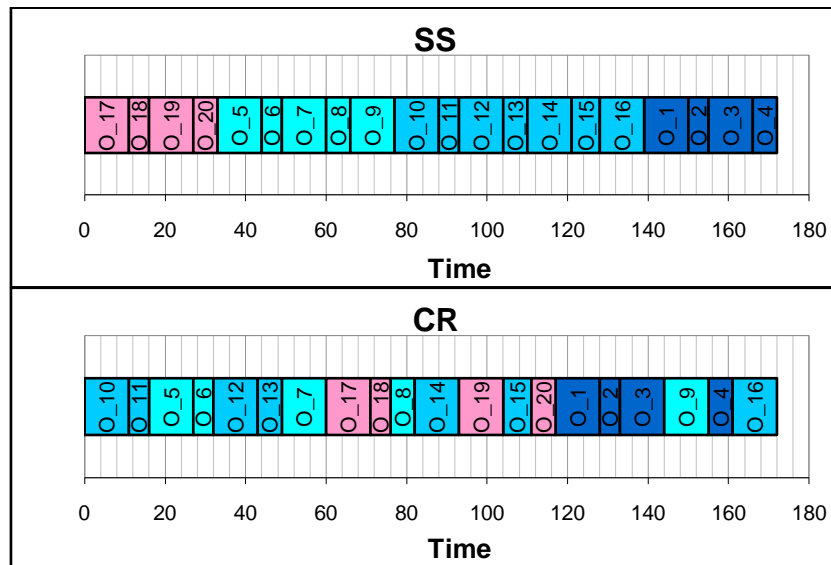


Figure 49 Scenario 2A: Resulting Schedules

Scenario 2B:

- Two machines: $k = 1, 2$
- Different processing times $p_{i,j,k} > 0$, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$

$p_{i,j,k}$	i							Total
	1	2	3	4	5	6	7	
1	11	5	11	6				33
	$k = 2$	$k = 1$	$k = 2$	$k = 1$				
2	11	5	11	6	11			44
	$k = 2$	$k = 1$	$k = 1$	$k = 1$	$k = 2$			
3	11	5	11	6	11	7	11	62
	$k = 1$	$k = 1$	$k = 2$	$k = 2$	$k = 2$	$k = 1$	$k = 1$	
4	11	5	11	6				33
	$k = 2$	$k = 2$	$k = 1$	$k = 2$				

- Different due dates: $d_1 = 2008/08/02$
 $d_2 = 2008/07/25$
 $d_3 = 2008/08/01$
 $d_4 = 2008/07/23$

The difference between the configurations of Scenario 2B and Scenario 2A is that Scenario 2B has two machines instead of one as in Scenario 2A. The proposed schedules for the different scheduling rules are shown in Figure 50.

Looking at the FIFO schedule, it is evident that the scheduling rule is correctly implemented as the operations are scheduled in the same order as they became active. The LPT schedule

process $O_9: p_{5,2,2} = 11$, $O_{13}: p_{4,3,2} = 6$ and $O_{14}: p_{5,3,2} = 11$ before $O_{18}: p_{2,4,2} = 5$ as their processing times are longer than $O_{18}: p_{2,4,2} = 5$, whereas $O_{18}: p_{2,4,2} = 5$ is processed before these operations and even $O_3: p_{3,1,2} = 11$ and $O_{12}: p_{3,3,2} = 11$ in the SPT schedule because of its shorter processing time.

The EDD and SS schedules process the operations correctly according to their operations, resulting in finishing the orders in the same order as their due dates. The CR schedule starts processing of *Part 3* first seeing that it has the longest processing time, making its critical ratio the biggest. As its operations are finished its critical ratio shrinks, because the part's remaining processing time shrinks. This allows the processing of operations of other parts to start seeing that their critical ratio got bigger, because their remaining processing time remained the same as processing was not done, whilst the hours to their due date got shorter, making the ratio bigger.

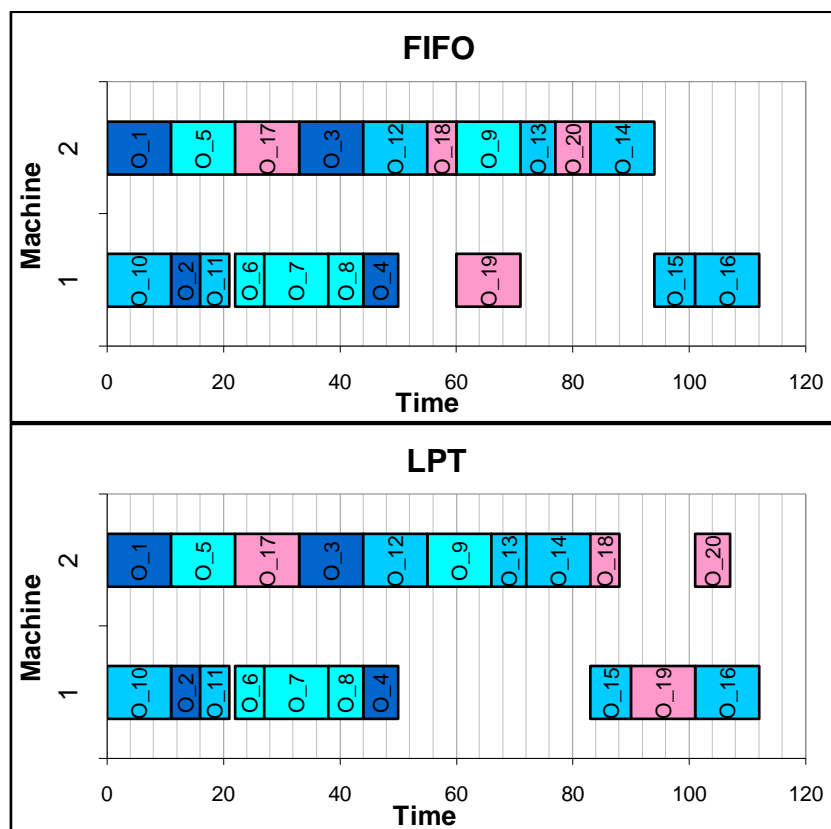


Figure 50 Scenario 2B: Resulting Schedules (Continued on next page)

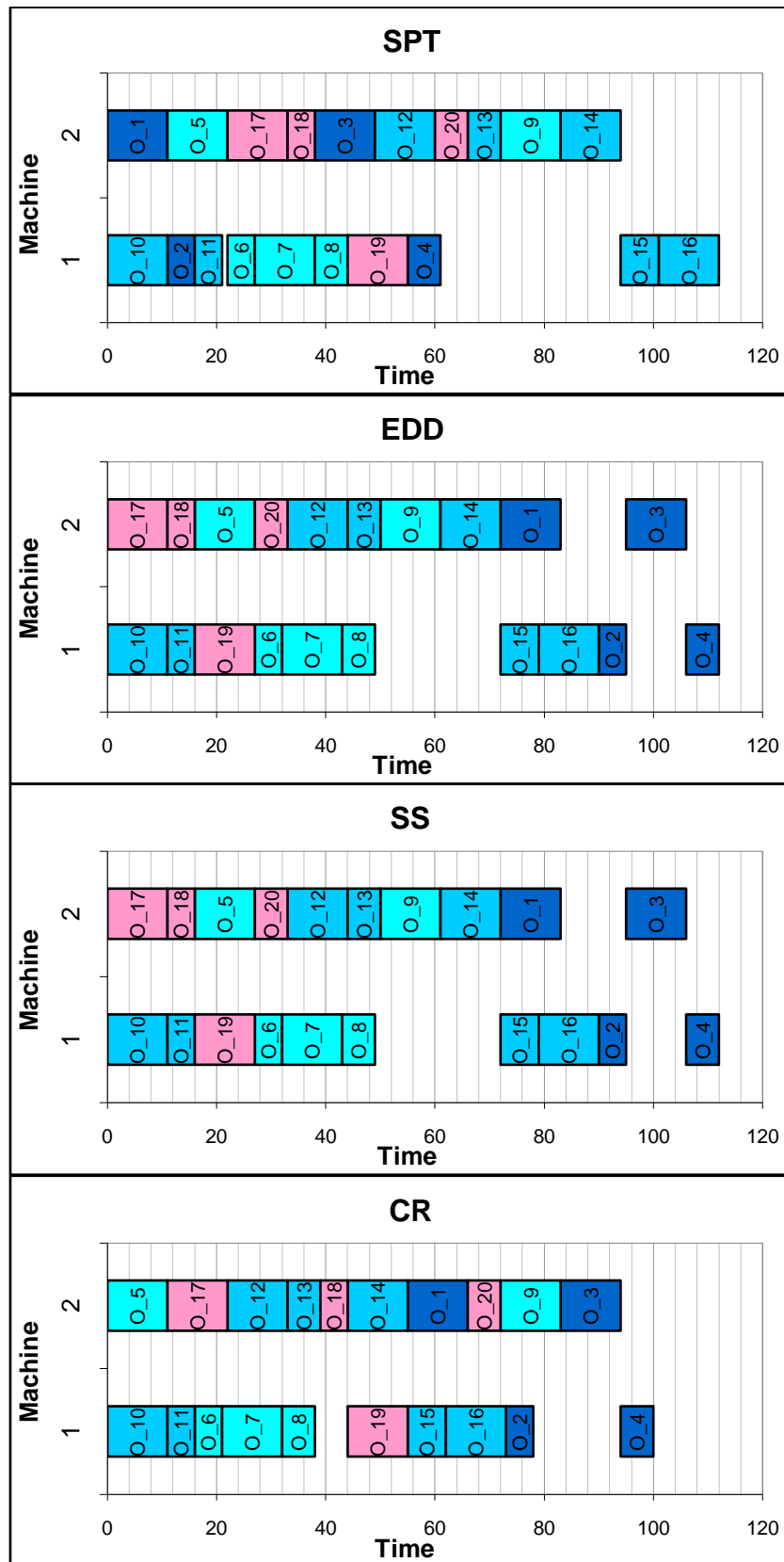


Figure 50 Scenario 2B: Resulting Schedules

Scenario 3A:

- Different machines: $k = 1, 2$
- Different processing times: $p_{i,j,k} > 0$, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$

Table 9 Processing times for Scenario 3A

$p_{i,j,k}$	i							Total	
	1	2	3	4	5	6	7		
j	1	34 $k = 1$	5 $k = 2$	11 $k = 2$	6 $k = 2$				56
	2	14 $k = 1$	5 $k = 2$	11 $k = 1$	6 $k = 1$	11 $k = 1$			47
	3	11 $k = 1$	5 $k = 2$	11 $k = 2$	6 $k = 1$	11 $k = 1$	7 $k = 1$	11 $k = 2$	62
	4	11 $k = 2$	5 $k = 1$	11 $k = 2$	6 $k = 1$				33

- Different due dates: $d_1 = 2008/08/02$
 $d_2 = 2008/07/25$
 $d_3 = 2008/08/01$
 $d_4 = 2008/06/30$

In Scenario 3A operations are performed on two machines, the due date of Part 4 is set very tight and the first operation of Part 1 has a very long processing time relative to the other operations. This scenario tested the response of the simulation model under fairly normal conditions.

Not one of the proposed schedules shown in Figure 51 is the same as another, the order in which operations are finished is also different for every schedule. The FIFO rule correctly processes the operations in the same order as they become active, $O_{-1}: p_{1,1,1} = 34$ with its long processing time therefore delays the other operations to adhere to the precedence constraints. $O_{-17}: p_{1,4,2} = 11$ starts on Machine 2 at t_0 because the first operations of the other parts must be processed on Machine 1 after $O_{-1}: p_{1,1,1} = 34$. The LPT schedule starts the same as the FIFO rule until $O_{-3}: p_{3,1,2} = 11$ is finished, when $O_{-4}: p_{4,1,2} = 6$ is processed instead of $O_{-6}: p_{2,2,2} = 5$ as in the FIFO schedule. This indicates that the scheduling rule is correctly implemented.

The SPT schedule clearly shows that the SPT rule delays the processing of $O_{-1}: p_{1,1,1} = 34$ because of its long processing time relative to the other operations. This enables other operations to start earlier as their precedence constraints are met, while the other

operations on *Part 1* ($O_{-2}: p_{2,1,2} = 5$, $O_{-3}: p_{3,1,2} = 11$, and $O_{-4}: p_{4,1,2} = 6$) are delayed to meet its precedence constraint.

The schedules of the due date based scheduling rules (EDD, SS and CR) all finish processing *Part 4* first, which is correct as its due date is very tight. The CR schedule differs from the EDD and SS schedule as it processes $O_{-1}: p_{1,1,1} = 34$ earlier, which states that the CR rule is implemented correctly. The critical ratio of *Part 1* got larger as time elapsed, until it was bigger than another part, resulting in the processing of $O_{-1}: p_{1,1,1} = 34$. The effect of processing $O_{-1}: p_{1,1,1} = 34$ and other operations earlier because of their bigger critical ratio can be seen on the schedule as the makespan is shorter.

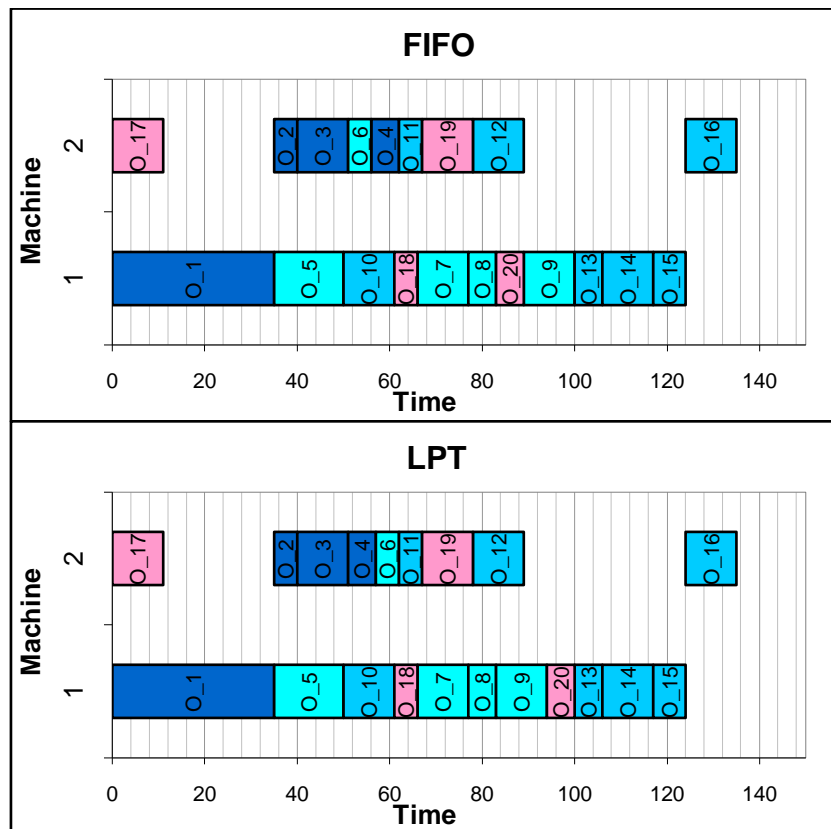


Figure 51 Scenario 3A: Resulting Schedules (Continued on next page)

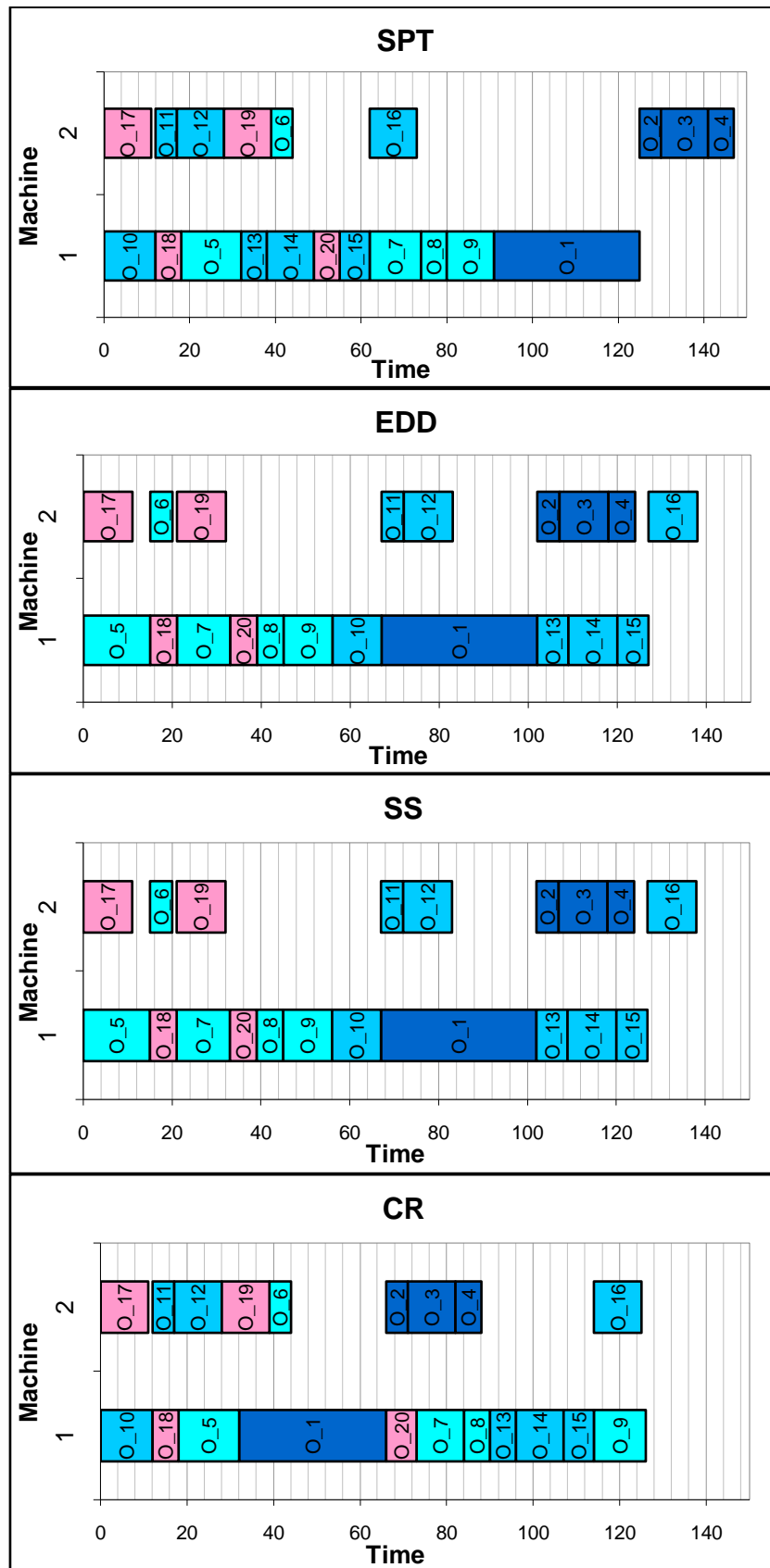


Figure 51 Scenario 3A: Resulting Schedules

Scenario 3B:

- Different machines: $k = \{1 \dots 8\}$ (No operations on machines six and seven)
- Different processing times: $p_{i,j,k} > 0$, where $i = \{1 \dots 7\}$ and $j = \{1 \dots 4\}$

$p_{i,j,k}$	i							Total	
	1	2	3	4	5	6	7		
j	1	34 $k = 1$	5 $k = 8$	11 $k = 2$	6 $k = 2$				56
	2	14 $k = 1$	5 $k = 2$	11 $k = 4$	6 $k = 3$	11 $k = 3$			47
	3	11 $k = 1$	5 $k = 3$	11 $k = 3$	6 $k = 3$	11 $k = 4$	7 $k = 4$	11 $k = 5$	62
	4	11 $k = 2$	5 $k = 1$	11 $k = 2$	6 $k = 4$				33

- Different due dates: $d_1 = 2008/08/02$
 $d_2 = 2008/07/25$
 $d_3 = 2008/08/01$
 $d_4 = 2008/06/30$

The configuration of Scenario 3B is very similar to the configuration of Scenario 3A, except that the operations were performed on several different machines. Each operation was assigned to a specific machine, machines six and seven have no operations assigned to them. This scenario tested the response of the simulation model under extreme conditions.

The proposed schedules of the FIFO and LPT rules are very similar, the order in which parts are finished are the same. The schedule is quite stretched out because of the precedence constraints and the long processing time of $O_{1,1,1} = 34$, resulting in operations waiting for other operations to finish.

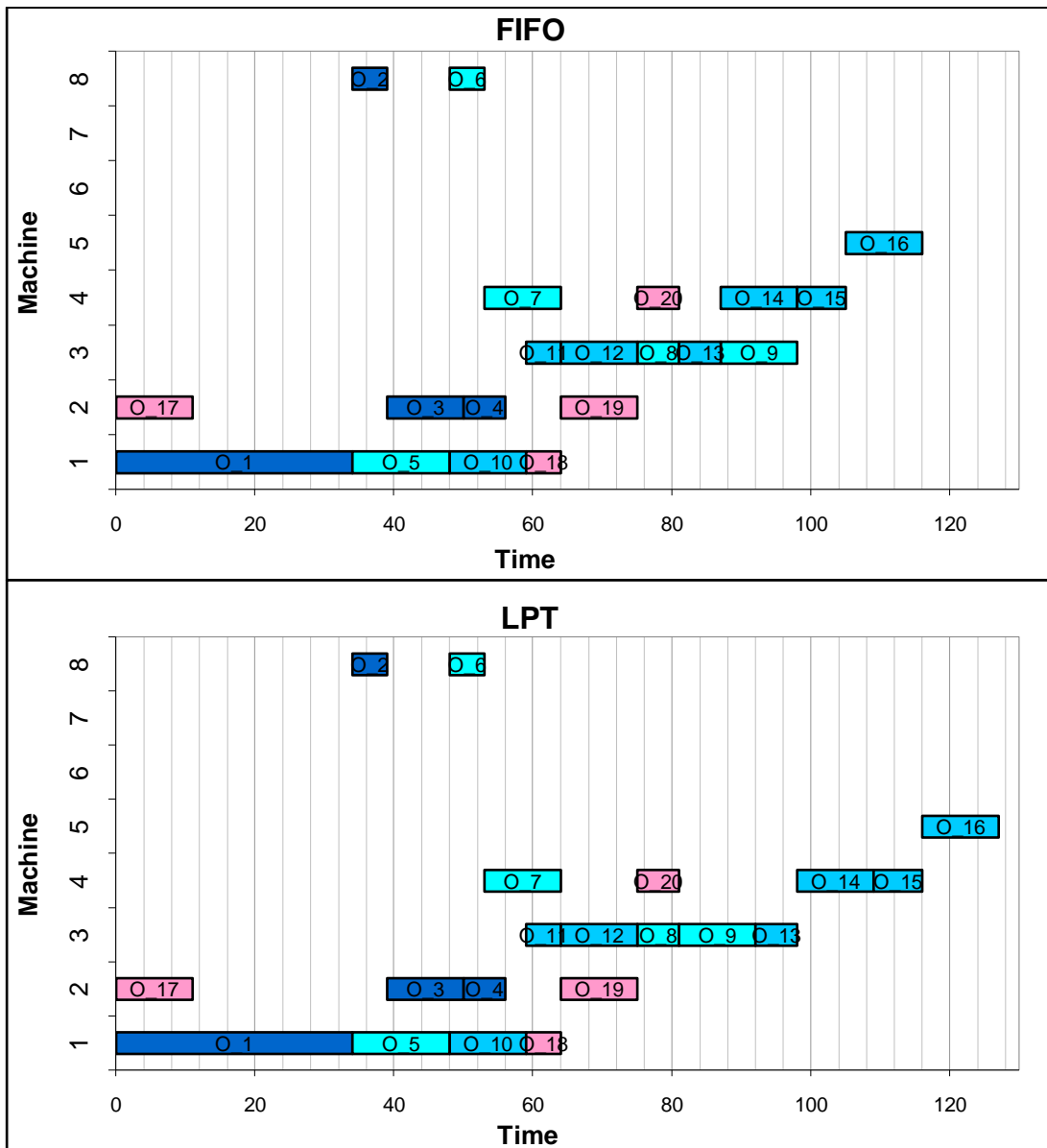


Figure 52 Scenario 3B: Resulting schedules (continued on next page)

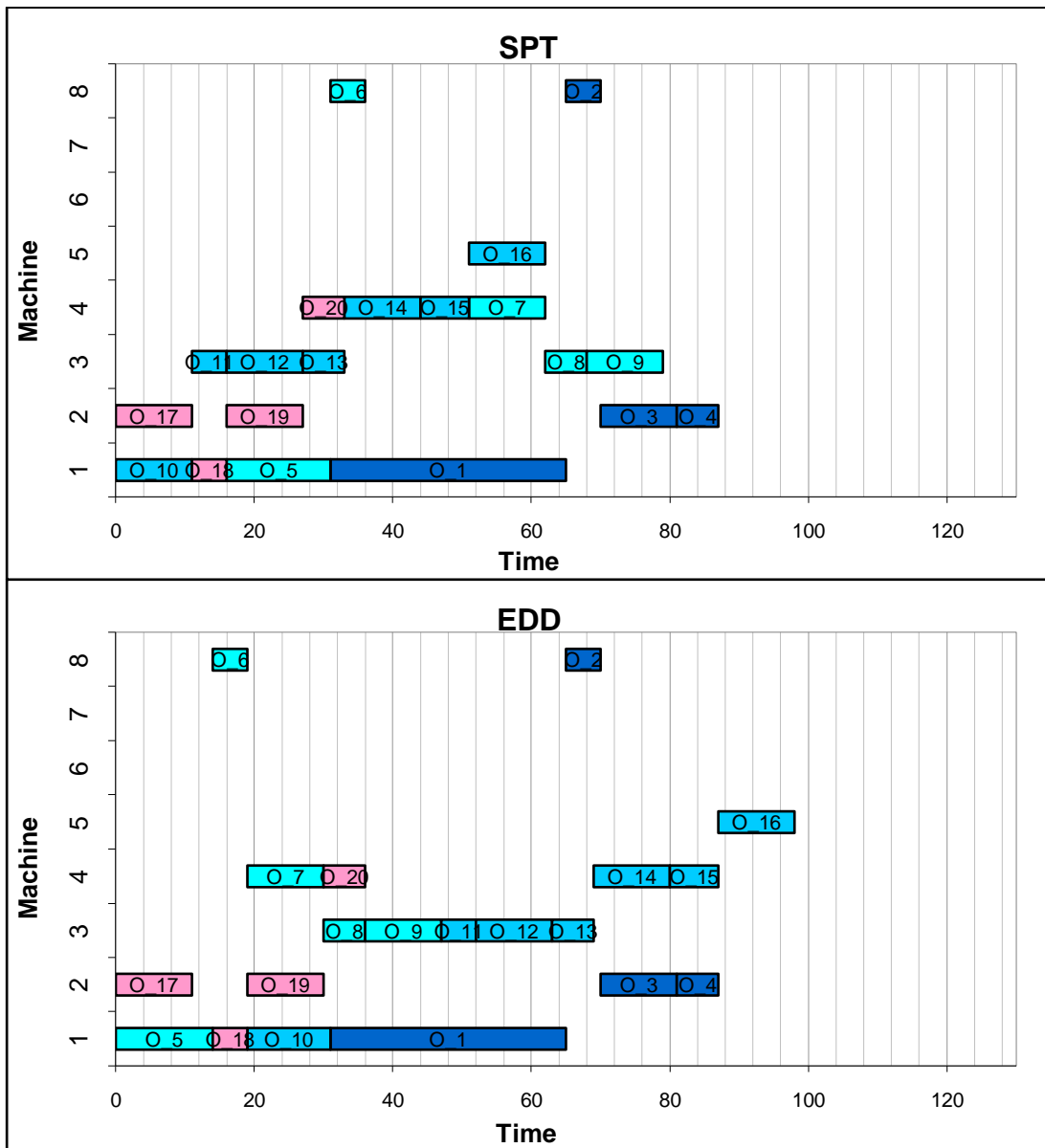


Figure 52 Scenario 3B: Resulting schedules (continued on next page)

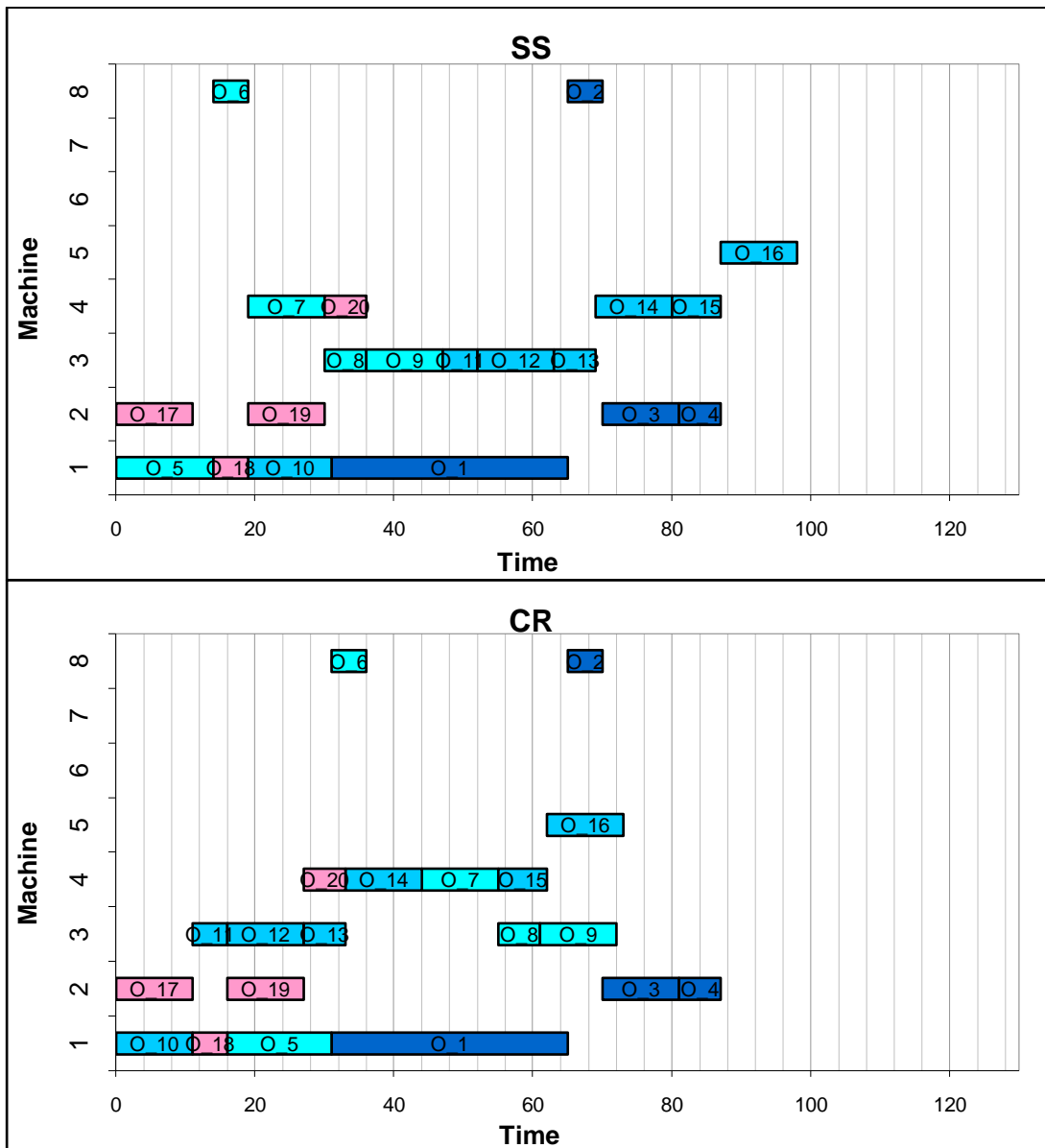


Figure 52 Scenario 3B: Resulting schedules

The SPT rule schedule is significantly shorter as several operations start before $O_{-1}: p_{1,1,1} = 34$, implying that their successors do not have to wait for $O_{-1}: p_{1,1,1} = 34$ to finish before their predecessors can be processed and they can become active.

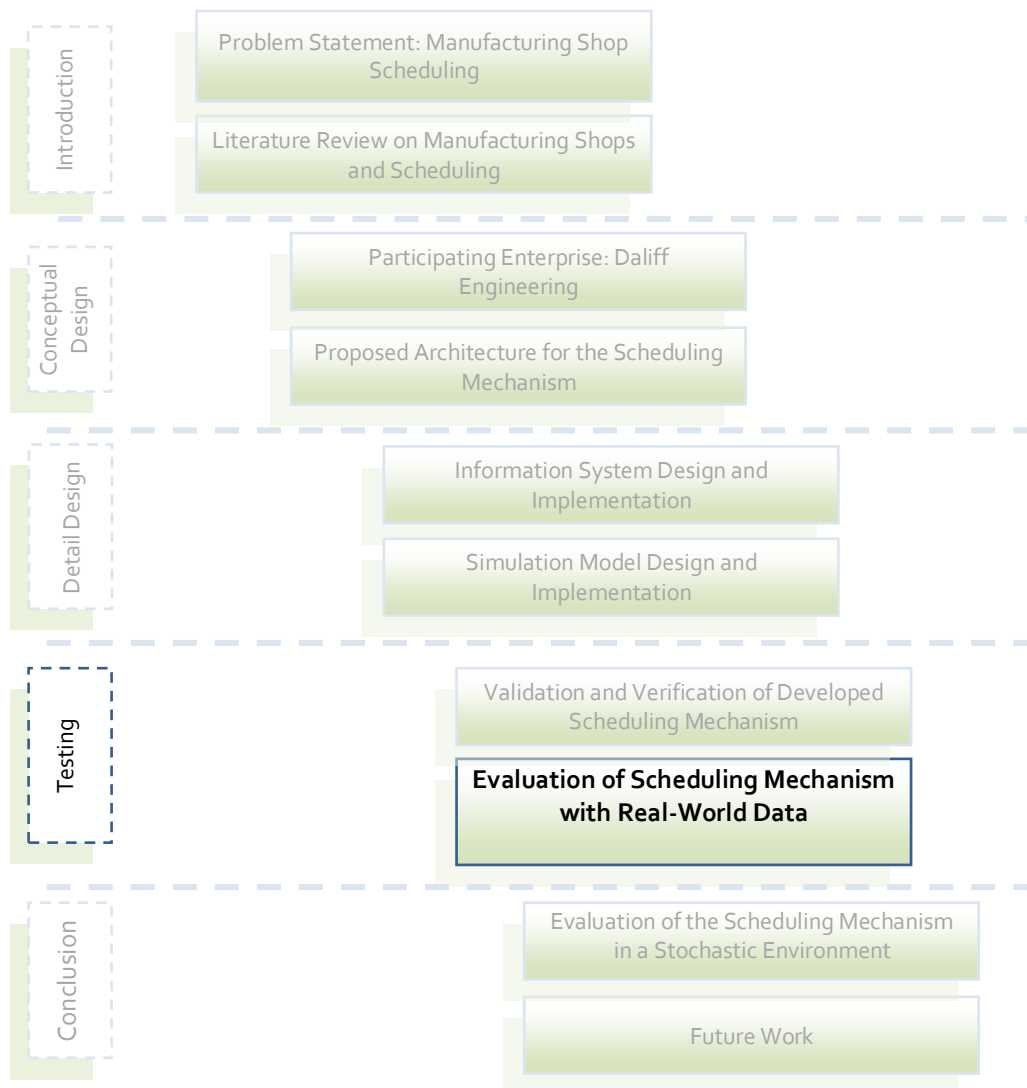
The schedules of EDD, SS and CR are also shorter than those of FIFO and LPT, as the due dates of the parts are so that the same happens as with SPT regarding $O_{-1}: p_{1,1,1} = 34$. All three schedules finish processing of Part 4 first as its due date is first and very tight. The order in which processing of the parts in EDD and SS finishes, is the same (Part 4, Part 2, Part 1, and Part 3). Part 3 is processed second in the CR schedule because its ratio of

remaining processing time to hours to due date is bigger than those of *Part 2* and *Part 1* at certain points in time, where scheduling decisions were made.

In this chapter the functionality of the scheduling mechanism was verified using test data. The validation of the scheduling mechanism on the specific manufacturing shop is done in the next chapter by evaluating the scheduler with real-world data.

9. EVALUATION OF THE SCHEDULING MECHANISM WITH REAL-WORLD DATA

In the previous chapter the testing phase (see roadmap) was completed, the next phase is drawing a conclusion, as shown on the roadmap below. The first step of drawing a conclusion is to test the performance of the scheduling system in a real-world scenario to determine the effectiveness of the scheduling mechanism. The schedule that is developed by the scheduling mechanism for the real-world scenario must be compared to the actual schedule that was used in the real-world system. The process that was followed to implement the scheduling mechanism with historic real-world data and evaluate its performance is discussed in this chapter.



9.1 EVALUATION PROCESS DESIGN AND IMPLEMENTATION

The evaluation methodology, shown in Figure 53, that was developed can be described as follows. Real-world data of the arriving orders during a set time frame had to be captured to use as input for the scheduling mechanism. This data had to contain order information like production plans, release dates, due dates and processing times to enable the scheduling mechanism to generate a schedule. Further data, such as operation start- and end times were needed to construct the actual schedule that was followed to be able to compare it to the schedule of the scheduling mechanism.

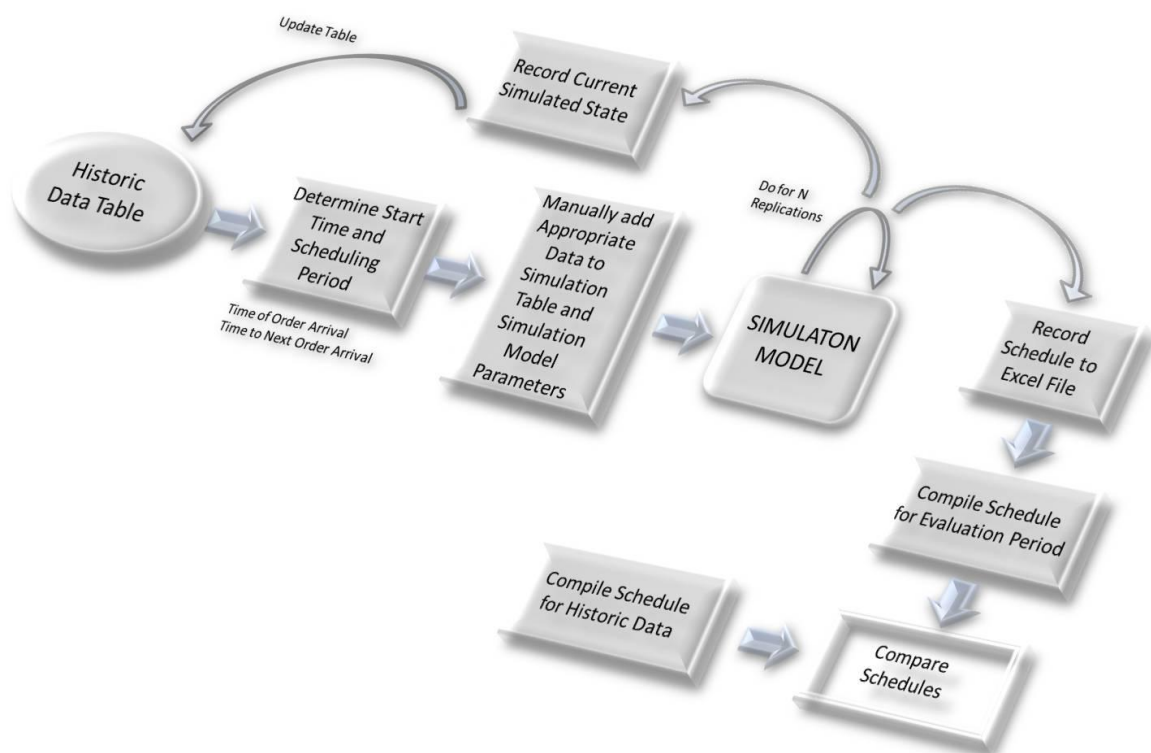


Figure 53 The Evaluation Methodology

Since the database Daliff currently uses is purely for costing purposes, the data in its current form was not sufficient, thus the operational data had to be generated from time sheets. The time sheet currently in use has five columns where the type of procedure can be represented; they are setup, inspection, production, rework and other. The sheet has nineteen rows that together represent all the hours of the day in increments of 30 minutes. Each machine has its own time sheet and a new sheet is started at the beginning of each day.

The current method of keeping record of the processing done on machines can be described as follows: when a job starts on a machine, the job and operation number are added to the time sheet at the appropriate time of day that represents the start time of the operation. A tick is made in the column that represents the type of operation that was completed to record the operation hours on the machine, i.e. to determine the setup time of an operation, the number of ticks that are categorized under the operation in the setup column is counted and multiplied with 30 minutes. The start time and duration of the setup, inspection and production processes of each operation can thus be read from the time sheet.

The operations that were already occupying a machine before the time frame, but continued doing so at the start of the time frame, could not be scheduled along with the other operations. These operations must start at the beginning of the schedule. Thus a busy status had to be given to these operations to enforce these operations to start at the beginning of the schedule. With this, they were also excluded from the scheduling process.

As this data is historic, it has characteristics that the data the scheduler was designed for, do not have. One example of this is, when the scheduler is activated on a certain date, it will have access to order information that will actually only be available after the date the scheduler is activated. The data could thus not be used as it would during normal operation of the scheduler. Alteration in the normal use of the scheduler was needed to test the performance of the scheduler against the performance of the current system. The adjusted use of the scheduler is described subsequently.

Usually the use of the scheduler has no time constraints as it works from the current state of the system. Using historic data, the use of the scheduler needs some time constraints. The start date and time window of the scheduler need to be adjusted according to the historic data. The first start date is set to the beginning of the time period chosen for the evaluation. The length of the simulation time window is set to the time from the start of the scheduling to the time the next order arrives in the system. Therefore the scheduler generates a schedule for the time period between order arrivals. The start time of the next scheduling run is set to the time of arrival of the new order. The time window is again set to the time till the next order arrives. In the historic data new order arrivals only occurred at the beginning of a day, thus the scheduler run window is set to a day and on each new day a run is started.

The historic data were saved in a temporary table in the database, but some data constraints had to apply as well during the evaluation. Only data of the orders that arrived in the scheduling period had to be included for the scheduling. The appropriate data are manually added to the simulation query table before each run. The scheduler will then develop a schedule for this data. After each scheduling run in the evaluation period the resulting schedule is written to the temporary Excel file. The table with the historic data is also updated so that the system state at the end of the scheduling run can be recorded. This is needed so that the simulated system state can be reflected at the start of the next scheduling run.

At the end of the evaluation period the schedules are merged to generate the schedule the scheduler would have implemented. The schedules can then be compared with each other.

9.2 EVALUATION PROCESS RESULTS

This section shows the proposed schedule for each rule that the scheduling mechanism generated and compares it to the actual schedule that was used in the time frame. A schedule that has been developed by a mixture of the scheduling rules is also included in this section, the schedule was built by choosing the best performing scheduling rule for every simulation period, i.e. implementing the schedule on a particular day that gave the best results in terms of the hours parts are produced before their due dates. A summary of the comparison between the proposed schedules and the actual schedule is shown in Table 10. The comparison is made in terms of the end times of the parts, and not the individual operations of the parts, because the delivery of parts is the outcome of any manufacturing process.

Table 10 Comparison of schedules

Scheduling rule	Total hours earlier	Hours per part earlier	Number of parts earlier	Number of parts later	Percentage of hours earlier
FIFO	173.167	2.935	28	31	17%
LPT	-102.083	-1.73	28	31	-10%
SPT	256.167	4.342	36	23	24%
EDD	202.667	3.435	30	29	19%
SS	245.667	4.164	36	23	23%
CR	3.167	0.054	27	32	0%
Mixed	189.667	3.215	28	31	18%

* The evaluation period consisted of 1105 machine hours

Table 10 has five columns that each represents a comparison criterion; the first column shows the total hours gained, and in the case of LPT hours lost, per scheduling rule in terms of parts being finished earlier than parts in the actual schedule. The second column shows the average hours gained or lost per part. The third column shows the total number of parts that were finished earlier than the parts in the actual schedule, whilst the fourth shows the number of parts that were finished later. The last column shows the cumulative percentage of the hours that parts are delivered earlier in comparison to the total machine hours of the time period.

From the table it can be seen that the proposed schedule developed under the SPT rule has the best result and would have, if implemented in stead of the actual schedule, produced the parts 4.342 hours earlier on average. The proposed schedule of the LPT rule has the worst result, seeing that on average parts are delivered later than with the actual schedule.

The scheduler is built to be used as it was in the mixed schedule, the results are thus a little bit concerning because using the scheduler as intended did not yield the best result. This shows the uncertainty involved with online scheduling. For example, if one knew what kind of production would arrive in the system at a later stage in time, one could adhere to only one specific scheduling rule and know that eventually it will yield better results. Resolving this concern will later be discussed in this thesis in the chapter about further developments that can be made to the scheduling mechanism.

What this table does not show, is that the makespan of all of these schedules are exactly the same. The reason for this is that there is an order that is only released on the last day of the time frame, which could not be rescheduled any earlier than its release date. The actual schedule and schedules that were generated by the different scheduling rules are shown and discussed on the next few pages.

It is important to note that in these schedules there are processing periods that overlap as some of the machines have a capacity of more than one. This enables the scheduler to schedule more than one operation on a machine at the same time, making the periods as they appear in the schedules not completely correct, for example machines two, seven and eight have a capacity of two, two and four respectively and the rest of the machines a capacity of only one. The accompanying Excel sheet with each operation start and end time is used to see how operations were scheduled on these machines (see section 7.6 on p.75).

The actual schedule that the production of the parts in the real-world followed is displayed in Figure 54. The other schedules are compared to this schedule and a short discussion of the comparison is included. The following comparisons refer to how the operations related to machines three, four, five and six are scheduled seeing that the other machines have capacities of more than one and machine one having only one operation. Looking at the operations of machine seven in the FIFO schedule in Figure 55, an example of the overlapping scheduling periods discussed previously can be given. The schedule shows that there are three operations scheduled one after another from time zero to ten hours, what actually happened is that there were four operations scheduled on machine seven in that time period. The fourth operation started on time zero and ended just before ten hours, the three operations displayed in the schedule were thus scheduled concurrently with the fourth operation seeing that the machine has a capacity of two. This concludes the explanation of the reason why the comparison of the schedules only refers to four machines.

A general observation that could be made from all the proposed schedules compared to the actual schedule, is that they are much less fragmented, i.e. having less stop start production on the machines. This is an indication that the time it took for decisions to be made regarding job allocation is eliminated in the proposed schedules.

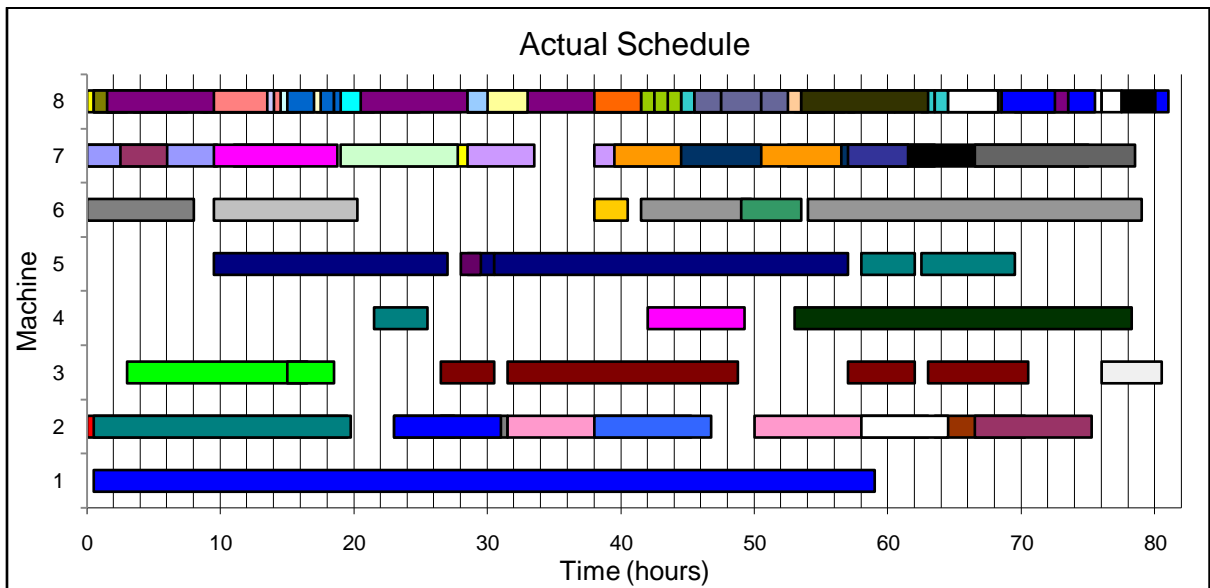


Figure 54 Actual schedule followed

The schedule that was developed with the FIFO rule is shown in Figure 55. From the figure it can be seen that the makespan of the production on machine two is shorter in the proposed FIFO schedule. The first operation on machine four (represented by the pink rectangle and indicated with an arrow) is scheduled right after its predecessor operation was finished on machine seven, not like in the actual schedule where it is delayed and resultantly delaying the part.

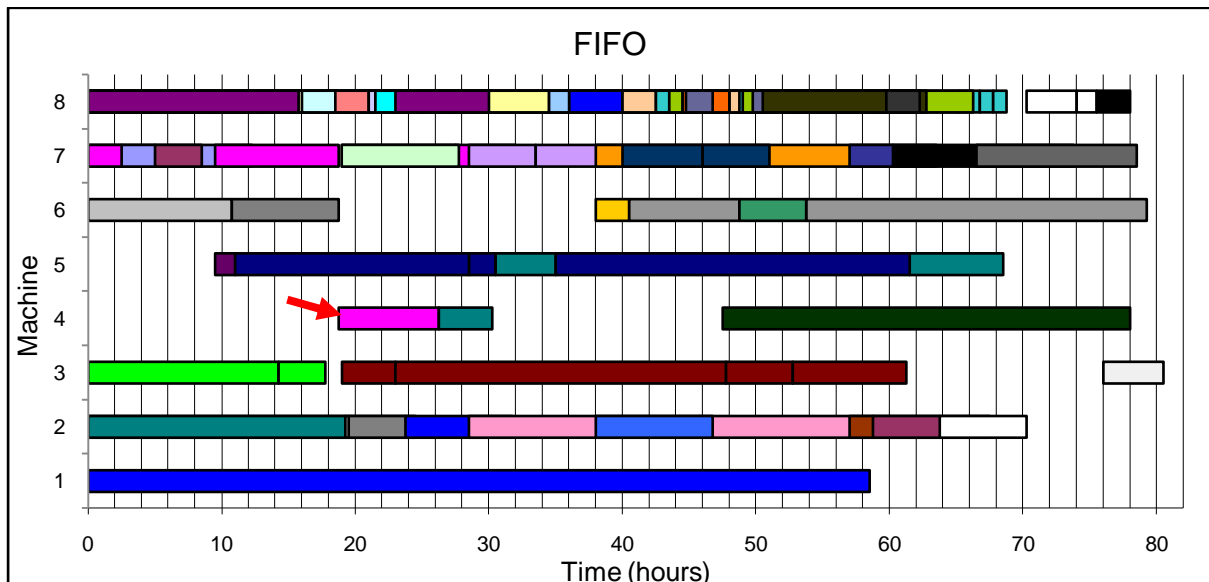


Figure 55 Proposed schedule for the FIFO rule

Figure 56 displays the schedule that was generated by using the SPT rule. The scheduled operations on machine five are a good indication that the SPT rule was implemented

correctly. The operations with the shortest processing time were scheduled first whilst the precedence constraints were still met.

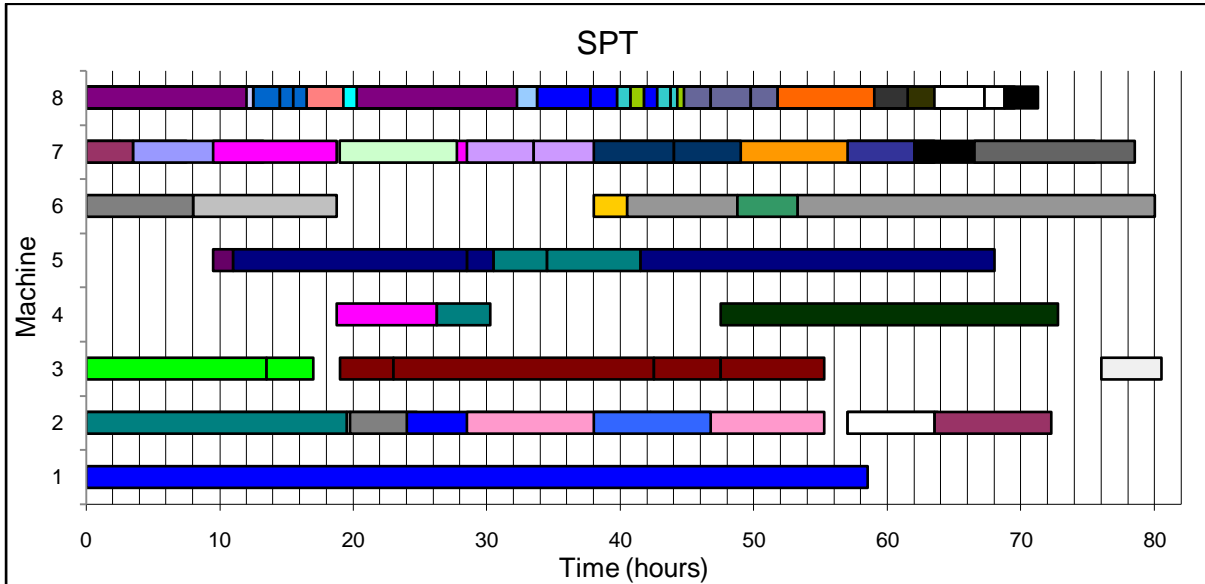


Figure 56 Proposed schedule for the SPT rule

Looking at machine five in Figure 57, which represents the proposed schedule developed under the LPT rule, it is evident that the LPT rule is correctly implemented as the operations are scheduled according to the longest processing times first. The operation that is represented by the white rectangle (indicated by an arrow) on machine two is delayed as its processing time is shorter than the processing times of other operations on the particular machine. This delays the further operation of the part on machine eight, which results in the longer makespan of production on machine eight.

The longer makespan on machine eight which has the highest capacity and at the same time the most operations to process, results in the schedule having the worst performance of all the schedules.

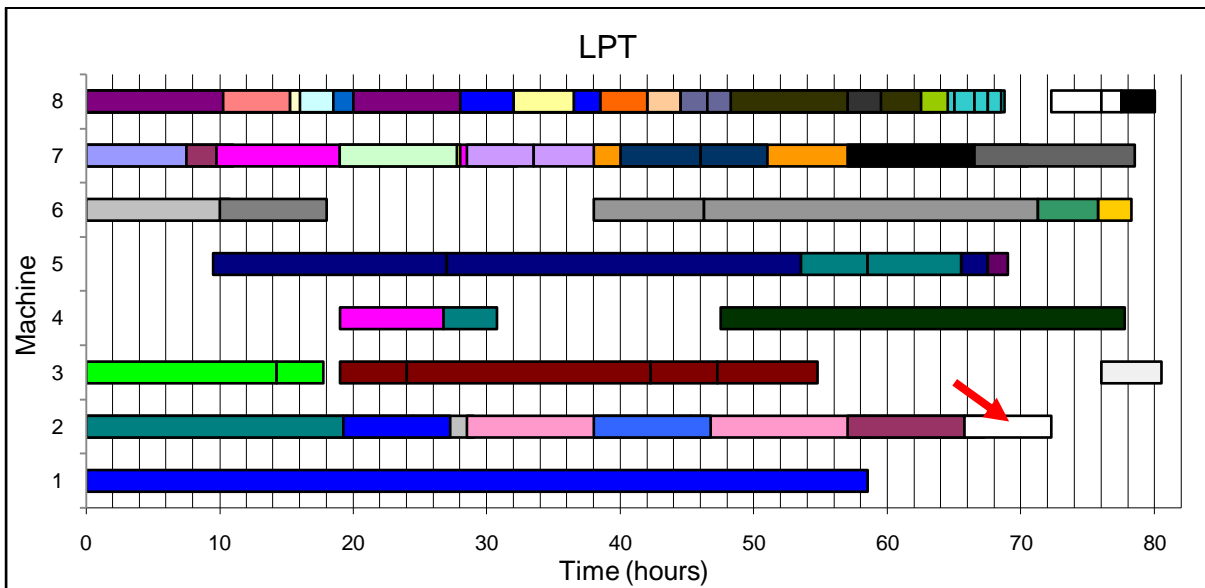


Figure 57 Proposed schedule for the LPT rule

Figure 58 displays the proposed schedule developed using the EDD rule. The makespan of production on machine two is also quite short, but similar to the LPT rule is the makespan of machine eight long, indicating fair results.

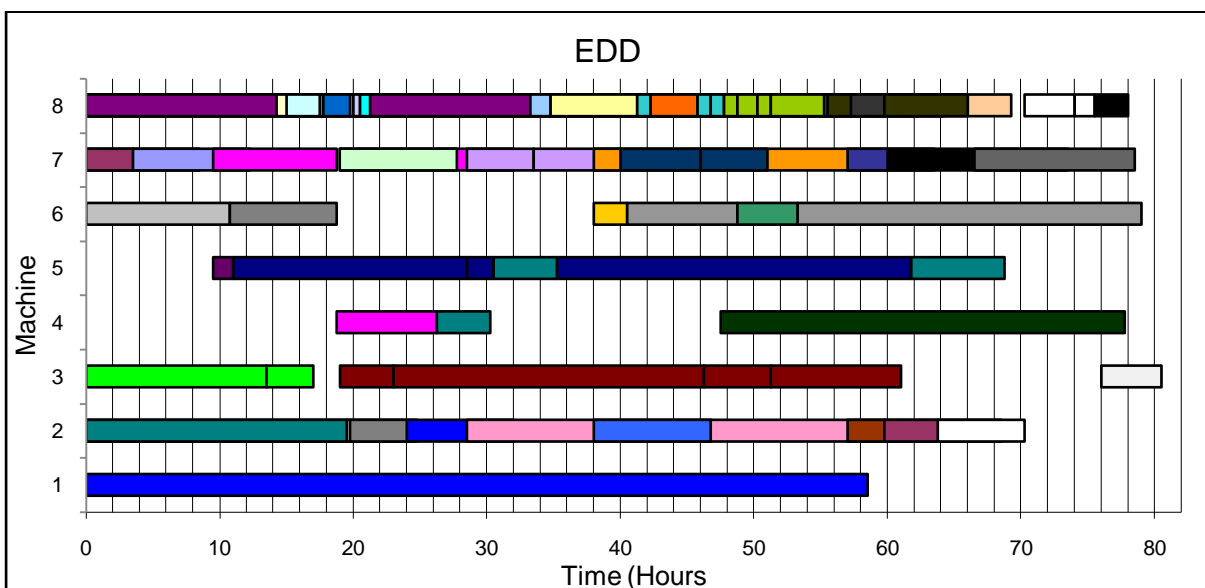


Figure 58 Proposed schedule for the EDD rule

The SS schedule (Figure 59) proposed by the scheduling mechanism is the second best schedule. Referring to machines four, six, seven and eight the schedule looks quite similar to the EDD. The hours saved with the SS schedule is evident when looking at machine three, where processing of the second part that is processed on machine three (shown by the marked rectangles) ends much earlier than the EDD schedule.

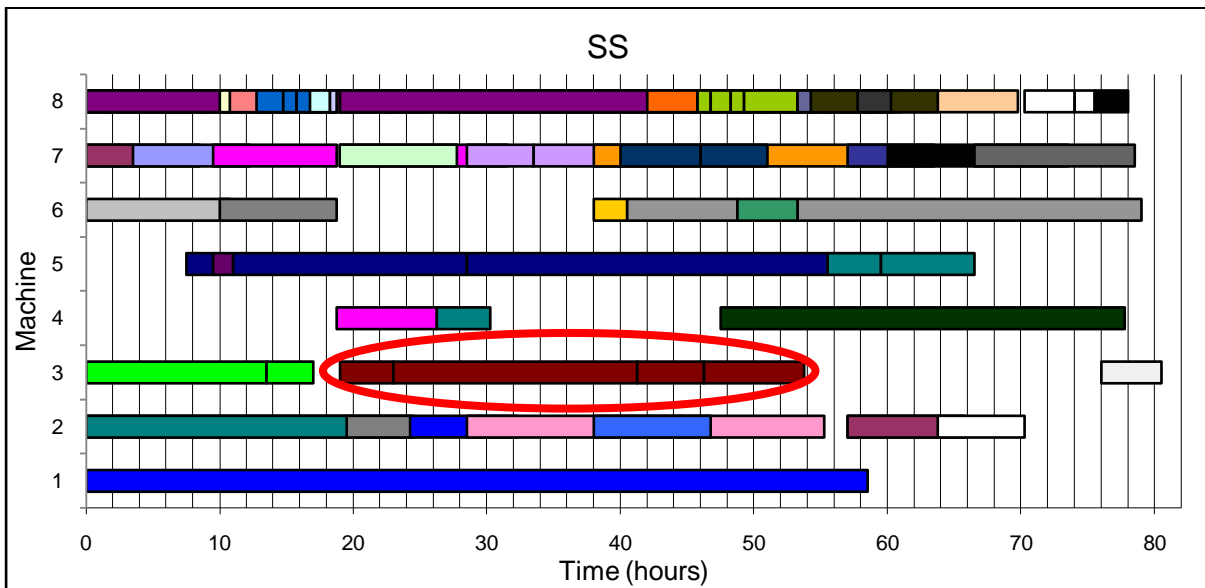


Figure 59 Proposed schedule for the SS rule

The proposed schedule developed from using only the CR rule is shown in Figure 60. The figure shows that the rule resulted in processing the operations with the longer remaining processing time first. Seeing that the due dates of the orders are quite close to one another, the critical ratios of the longer processing time operations are bigger and these operations are thus processed first.

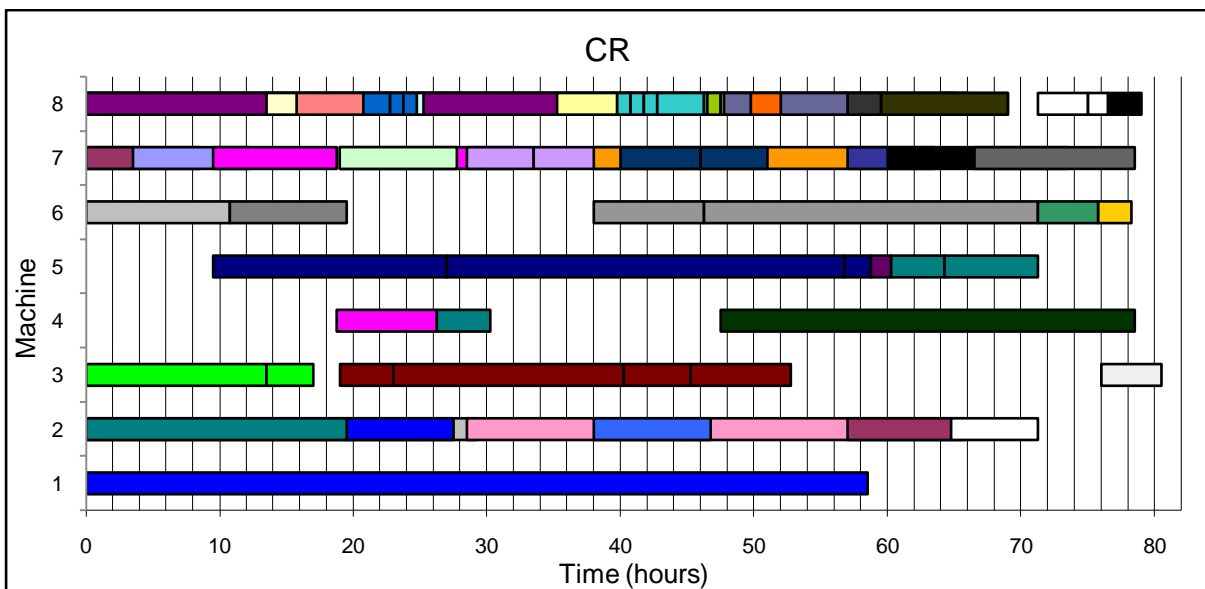


Figure 60 Proposed schedule for the CR rule

There is no significant evidence that explains why the CR schedule does not have better results than the actual schedule, the reason could be hidden in the overlapping scheduling periods of machines two, seven and eight.

The schedule that was developed by using the scheduling mechanism according to implementation guidelines stated previously in this thesis is shown in Figure 61. Although the makespan for production on machine eight is much shorter than some of the other schedules, its performance is not better than any of those individual scheduling rules. This is because the operations are finished in a different order, resulting in parts being finished on different times than in other schedules. Consequently, different numbers of hours are gained or lost with regards to part earliness and lateness.

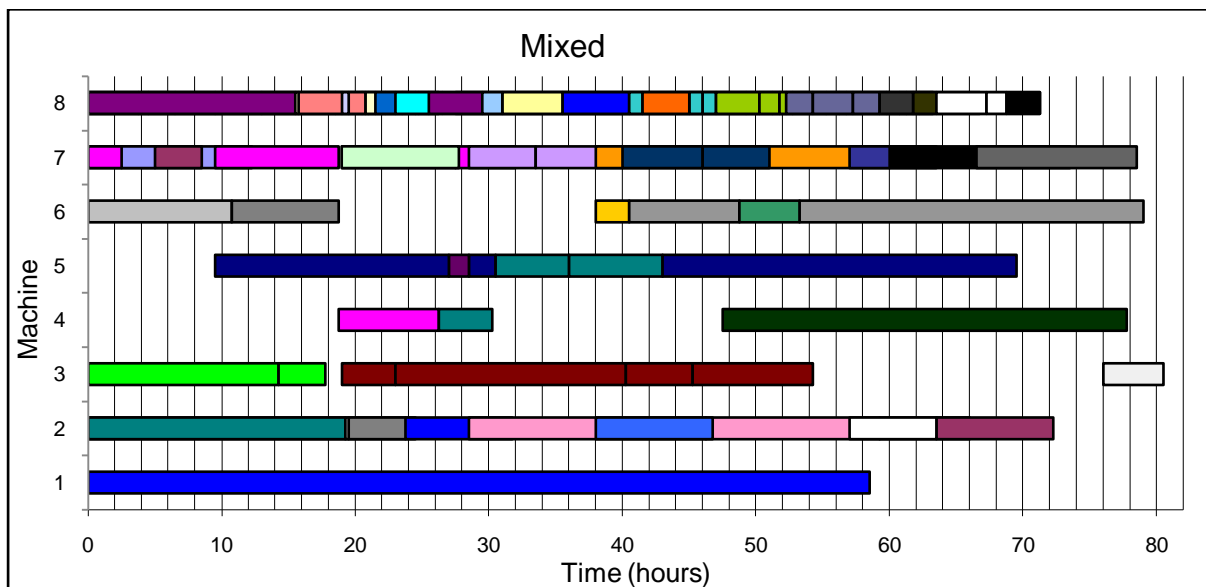


Figure 61. Proposed schedule for the Mixed rules

9.3 CHAPTER OVERVIEW

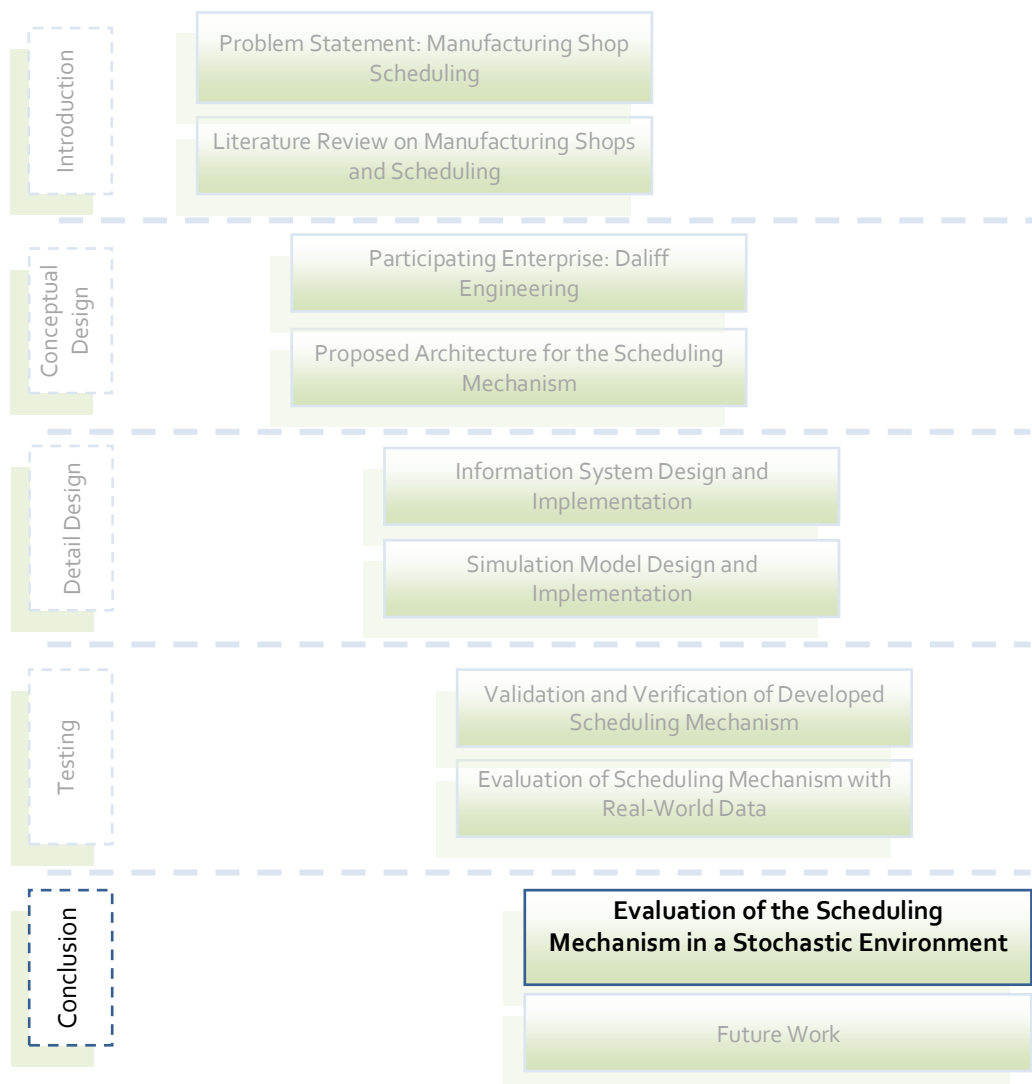
This chapter describes the process that was followed to test the performance of the scheduler with real-world data compared to the current scheduling mechanism. The comparison of the proposed schedules developed by the scheduler and the actual schedules were stated by referring to the total hours gained or lost with regards to part delivery dates. The process of schedule generation was repeated for every scheduling rule, i.e. running through the evaluation time period only using one scheduling rule. It was also completed using a mixture of scheduling rules, choosing the rule with the best result for the particular day, as the scheduling mechanism is supposed to be used for on-line scheduling.

The results indicate that using only the SPT rule will result in the best performance, regarding early parts, for the particular historic data. The results thus indicate that using the scheduler as intended is less desirable, if this is true it can only be clarified by implementing the scheduling mechanism for a longer period of time.

In this chapter the performance of the scheduling mechanism was tested in a deterministic environment, where in the real-world it should be able to operate similarly in a stochastic environment, because the job shop is a stochastic environment. This will be addressed in the next chapter.

10. THE SCHEDULING MECHANISM IN A STOCHASTIC ENVIRONMENT

The evaluation and validation of the scheduling mechanism were discussed in previous sections of this thesis (see road map). From the evidence presented it can be concluded that the scheduling mechanism correctly generates schedules, and that it implements the best performing scheduling rule for the short-term period for which order information is known. In this chapter the functionality of the scheduling mechanism in a stochastic environment is described, as indicated in the road map.



In the previous chapters the scheduler was evaluated in a deterministic environment in order to compare the performance of the scheduling mechanism to the current scheduling method Daliff uses. It was shown that the scheduling mechanism improves the scheduling of the enterprise, when compared to existing data of a specific historical time window.

When the scheduling mechanism is implemented for use in the real-world, it should also operate well in a stochastic environment, because the job shop is a stochastic environment. In this section it is explained how the scheduling mechanism adapts to a stochastic environment, and a demonstration of the functionality of the mechanism in such an environment is given. It is now explained why the job shop is of stochastic nature.

10.1 STOCHASTIC PROCESSING TIMES

During the production planning phase of orders, the production planner must predict a setup and a process time for each operation which determines the processing time of the operation. Setup and process times may vary because of different operators and changing circumstances on the shop floor, which makes it difficult for the planner to specify exact times for each operation. Instead, the planner may assign estimated values to the setup and process time of an operation, resulting in simple time distributions.

The scheduling mechanism exploits these distributions by drawing random observations for the setup and process time of an operation. In the implementation of the scheduling mechanism, the setup and process time of an operation are drawn from one of possible two distributions, these are the uniform distribution and triangular distribution. Each distribution will now be discussed, after which the implementation of the distributions in the scheduling mechanism is explained. Finally, a demonstration example will be presented to conclude this chapter.

10.2 THE CONTINUOUS UNIFORM DISTRIBUTION

The continuous uniform distribution (hereafter referred to as the uniform distribution), shown in Figure 62, implies that the probability of choosing any value between the two distribution parameters is equal. The distribution is denoted by $U[a, b]$, where a and b are the lower and upper bound respectively. The probability density function $f(x)$, the cumulative distribution function $F(x)$, the inverse cumulative distribution function $F^{-1}(U)$ and the expected value $E(x)$ of the uniform distribution are as follows:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & x < a; x > b \end{cases} \quad (1)$$

$$F(x) = \begin{cases} \frac{x-a}{b-a}, & a \leq x \leq b \\ 0, & x < a \\ 1, & x > b \end{cases} \quad (2)$$

$$F^{-1}(U) = U(b - a) + a \quad (3)$$

$$E(x) = \frac{a+b}{2} \quad (4)$$

The graphical representation of the probability density function is as follows:

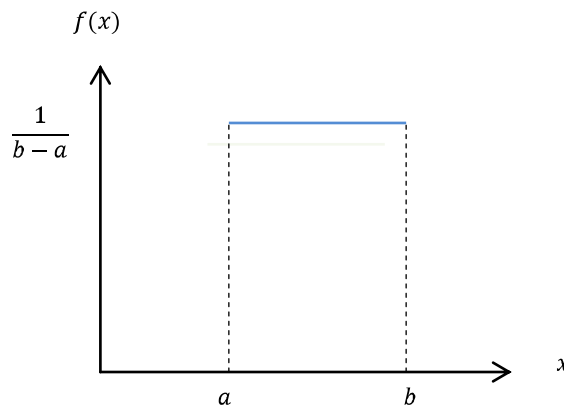


Figure 62 Uniform distribution

10.3 THE TRIANGULAR DISTRIBUTION

The triangular distribution, shown in Figure 63, is used where the distribution of a random variable can be approximated via three parameters, these are the minimum and maximum values (a and b), as well as the most likely value (mode) denoted by c . The probability density function $f(x)$, the cumulative distribution function $F(x)$, the inverse cumulative distribution function $F^{-1}(U)$ and the expected value $E(x)$ of the triangular distribution are as follows:

$$f(x) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)}, & a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)}, & c < x \leq b \end{cases} \quad (5)$$

$$F(x) = \begin{cases} \frac{(x-a)^2}{(b-a)(c-a)}, & a \leq x \leq c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)}, & c < x \leq b \end{cases} \quad (6)$$

$$F^{-1}(U) = \begin{cases} a + \sqrt{U(b-a)(c-a)}, & 0 \leq U \leq \frac{c-a}{b-a} \\ b - \sqrt{(1-U)(b-a)(b-c)}, & \frac{c-a}{b-a} \leq U \leq 1 \end{cases} \quad (7)$$

$$E(x) = \frac{a+b+c}{3} \quad (8)$$

The graphical representation of the probability density function is as follows:

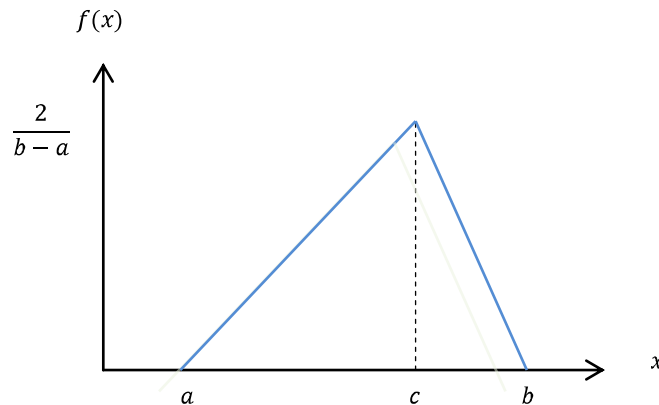


Figure 63 Triangular distribution

10.4 IMPLEMENTATION OF THE DISTRIBUTION

When a quote is changed into a new order, the production planner decides what operations will be necessary to manufacture the part(s) on the order, and on which machines the operations will be executed. Also, the planner is able to estimate the duration of each operation (setup and process time), and the information system allows for specifying either an uniform distribution or a triangular distribution. The input of the user in the information system determines which distribution is applicable to each operation. A snapshot of the input choices the user has, is shown in Figure 64.

Next to each operation there are six input fields, three each for the process time and setup time (see ovals in Figure 64). In both cases of the times, if the user only enters values into the first two of the three input fields, the distribution is uniform with parameters a and b . If the user enters values into each of the three fields the distribution is triangular. In Figure 64 the process time for *Ops 1* has an uniform distribution with $a = 2$ and $b = 4$ and the setup time of *Ops 1* has a triangular distribution with $a = 1$, $c = 2$ and $b = 5$.

DALIFF
PRECISION ENGINEERING (PTY) LTD

"We don't just make parts - we offer solutions to your engineering problems."

Add Ops

Account Number: D1
Account Name: David
Order Number: Dav007
Number of parts: 2

Wing QTY of Parts: 1

Ops Description	Enforce Machine	(check to enforce)	Processing Time			Setup Time		
Ops 1: Cut	V80	<input type="checkbox"/>	2	4	0	1	2	5
Ops 2: Finish	Megatum	<input type="checkbox"/>	2	4	5	3	8	9

Body QTY of Parts: 1

Ops Description	Enforce Machine	(check to enforce)	Processing Time			Setup Time		
Ops 1: Bend	5Axil	<input type="checkbox"/>	5	7	0	2	4	0

Add Ops Reset

Figure 64 Processing times input

The simulation model schedules the operations according to their respective expected setup and process times. Take for example the times of *Ops 1* indicated above, the expected value of the process time is $E(x) = \frac{a+b}{2} = \frac{2+4}{2} = 3$ and the expected value of the setup time is $E(x) = \frac{a+b+c}{3} = \frac{1+5+2}{3} = 2.67$. When operations are processed in the simulation model of the scheduling mechanism, it uses a random value that is drawn from the time distribution of the specific operation. Running a large number of replications each with a new random value for the processing times will provide a good estimation of how the real-world can be expected to perform. The simulation model uses the inverse transform method to generate random values from the distribution implemented, which will now be discussed.

The mechanism of the Inverse transform method is shown in Figure 65. Generating observations from a distribution using this method is accomplished by using the cumulative distribution function F of a distribution, and the assumption is that the inverse F , namely F^{-1} exists and can be determined. A pseudorandom number U is then generated from $U[0,1]$ which is entered in the inverse function, to return $X = F^{-1}(u)$.

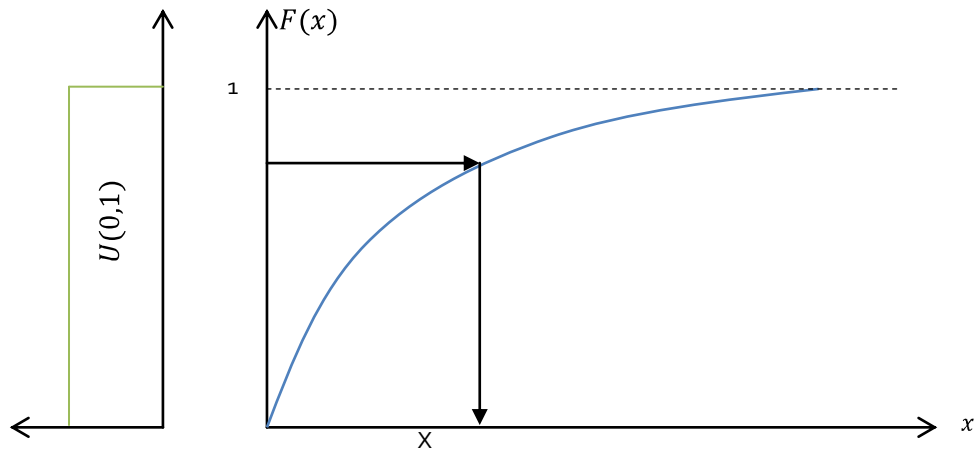


Figure 65 Random number generation by means of the inverse transform

A random value for each operation is calculated at the beginning of each simulation replication and saved in the simulation run table for further use. This is done by code in VBA that is called when a replication is started. The pseudo code for the inverse transform of the uniform distribution is as follows:

```
u = Rnd()
r = a + u * (b-a)
```

A pseudorandom number between 0 and 1 is generated by the simulation model, and this number is then inserted into the inverse cumulative function of the uniform distribution to return a random value drawn from the distribution. Referring to the process time of *Ops 1* mentioned in the examples above, it has a uniform distribution $U[2,4]$. Thus if a random number $U = 0.45$ is inserted into the inverse cumulative function of the distribution, the resulting random value drawn from the distribution is $r = 2 + 0.45 * (4 - 2) = 2.9$.

The pseudo code for generating observations from the triangular distribution using the inverse transform, is as follows:

```
u = Rnd()
If u <= (c-a) / (b-a) then
    r = a + sqr[u * (b-a) * (c-a)] ..... (Line 1)
else
    r = b - sqr[(1-u) * (b-a) * (b-c)] ..... (Line 2)
end if
```

A pseudorandom number between 0 and 1 is generated by the simulation model. If its value is less than or equal to the ratio of the area between a and c to the whole area, the random value is inserted into the inverse function (Line 1 above), else it is inserted into the inverse

function (Line 2 above). Referring to the setup time of *Ops 1* shown in Figure 64, it has a triangular function with parameters $a = 1$, $c = 2$ and $b = 5$. Thus if a random number $U = 0.76$ is generated by the simulation model, it is greater than $\frac{(c-a)}{(b-a)} = \frac{(2-1)}{(5-1)} = 0.25$ and the random value drawn from the distribution is $r = b - \sqrt{(1-u)(b-a)(b-c)} = 5 - \sqrt{(1-0.76)(5-1)(5-2)} = 3.31$.

With the knowledge of the different distributions and how they are implemented by the simulation model, an example of a schedule built by the scheduling mechanism using stochastic times, is subsequently discussed.

10.5 FUNCTIONALITY OF THE SCHEDULING MECHANISM IN A STOCHASTIC ENVIRONMENT

In the real world the scheduler will be activated when a new order or a set of new orders need to be scheduled. The input for the scheduling mechanism is the current shop-floor state and the information about the new information. The scheduler will activate the simulation model that runs a few replications, each using random values drawn from the appropriate time distributions, with the aim to predict the performance of the specific scheduling rule under investigation. The performance of a scheduling rule in this stochastic environment is estimated by averaging the performance over all the simulation replications used to run the scheduling rule in the current system state. This process is repeated for each scheduling rule and the performance of each is compared to select a specific scheduling rule that must be followed for the current system state and the new set of orders.

The scheduling mechanism uses the simulation model to compile a schedule from the expected times of the processing distributions following the chosen scheduling rule, this schedule is used to control the real world shop-floor. The new schedule is followed on the shop-floor until a new order or a set of new orders arrives in the system, at which point this process is repeated.

10.6 DEMONSTRATION OF THE FUNCTIONALITY OF THE SCHEDULING MECHANISM IN A STOCHASTIC ENVIRONMENT

To demonstrate the functioning of the scheduling mechanism, a simulation model of the shop floor will be used as if it is the real world system. The simulation model, i.e. the "real world", operates under a certain scheduling rule and processes current scheduled orders. The current state of the shop-floor at the point in time when a new order arrives is recorded by the information system, i.e. the status of the simulation model is read. This information is then used as input together with the information of the new order set for the scheduling mechanism.

The following steps will be followed to demonstrate the functionality of the scheduling mechanism in a stochastic environment:

- Step 1. Enter the information about the set of new orders into the information system.
- Step 2. Run the simulation model, using the database as input, for 25 replications for each of the five scheduling rules, using stochastic processing times. The simulation model will develop schedules for each scheduling rule based on the simulated results.
- Step 3. Evaluate the performance of each schedule, and choose the scheduling rule that generated the best schedule.
- Step 4. Build a new schedule using the expected processing times of the candidate orders, according to the chosen scheduling rule.
- Step 5. "Execute" the real world: Run the simulation model applying the chosen scheduling rule for an arbitrary length of time to imitate the flow of jobs under the chosen scheduling rule. The arbitrary length is determined by the arrival of a new set of orders.
- Step 6. Record the current imitated state of the system (from the simulation model) into the information system – for example which of the existing orders are completed, which are processed and which of the orders still need to be processed.
- Step 7. If there are any new orders, go to step 1, else stop.

These steps were followed to demonstrate the functionality of the scheduling mechanism in a stochastic environment, and the reader will now be guided through the process.

The scheduling mechanism was used to develop a schedule for a total of 20 orders that arrived in the system over a time period of ten days. The orders did not arrive at the same time, but at "random" times, i.e. the arrivals were chosen by the author. The orders consisted of 49 operations that needed to be scheduled and processed. The processing

times of each operation are stochastic and have certain distributions associated with them. The time-line of the order arrivals is shown in Figure 66, the arrows representing the events of arrivals.

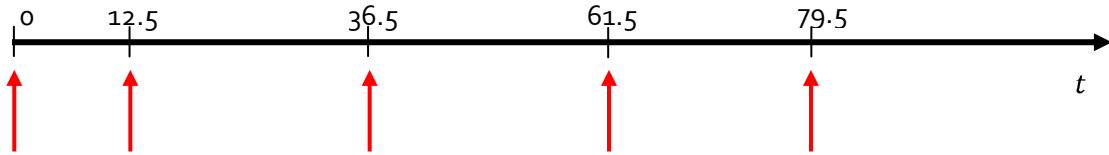
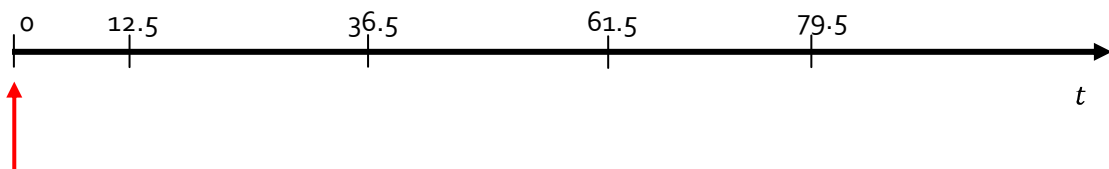


Figure 66 Time-line of order arrival

Figure 66 is repeated several times in the remainder of this section to indicate at what point in time a new set of orders arrived in the system. The arrow represents the time of this event, as shown below for $t = 0$:



At time zero four orders consisting of 13 operations are in the system, these orders are shown in Table 11. The scheduling mechanism is used to compile different schedules for each scheduling rule so that their performance could be compared to choose the best schedule to follow.

Table 11 Order set 1

Job No	Approved Date	Prom Date	Ops No	Mach ID	Mach name	Setup timeA	Setup timeB	Setup timeC	Insp time	Proc TimeA	Proc TimeB	Proc TimeC	Status Id
1	2008/06/01	2008/08/02	1	1	V80	2	3	0	1	2	3	4	1
1	2008/06/01	2008/08/02	2	8	V40_New	2	3	0	1	20	30	0	0
1	2008/06/01	2008/08/02	3	2	V40_Old	2	3	0	1	1	2	0	0
2	2008/06/01	2008/07/25	4	1	V80	2	3	0	1	1	3	0	1
2	2008/06/01	2008/07/25	5	4	T7	2	3	0	1	7	8	13	0
2	2008/06/01	2008/07/25	6	3	Megaturn	2	3	0	1	1	7	8	0
3	2008/06/01	2008/08/01	7	1	V80	2	3	0	1	3	8	9	1
3	2008/06/01	2008/08/01	8	8	V40_New	2	3	0	1	10	12	13	0
3	2008/06/01	2008/08/01	9	3	Megaturn	2	3	0	1	2	4	0	0
3	2008/06/01	2008/08/01	10	4	T7	2	3	0	1	1	3	4	0
3	2008/06/01	2008/08/01	11	5	5Axil	2	3	0	1	4	7	0	0
4	2008/06/01	2008/06/30	12	2	V40_Old	2	3	0	1	1	4	5	1
4	2008/06/01	2008/06/30	13	1	V80	2	3	0	1	5	17	0	0

The performance of each schedule developed for the orders in Order set 1 is shown in Figure 67.

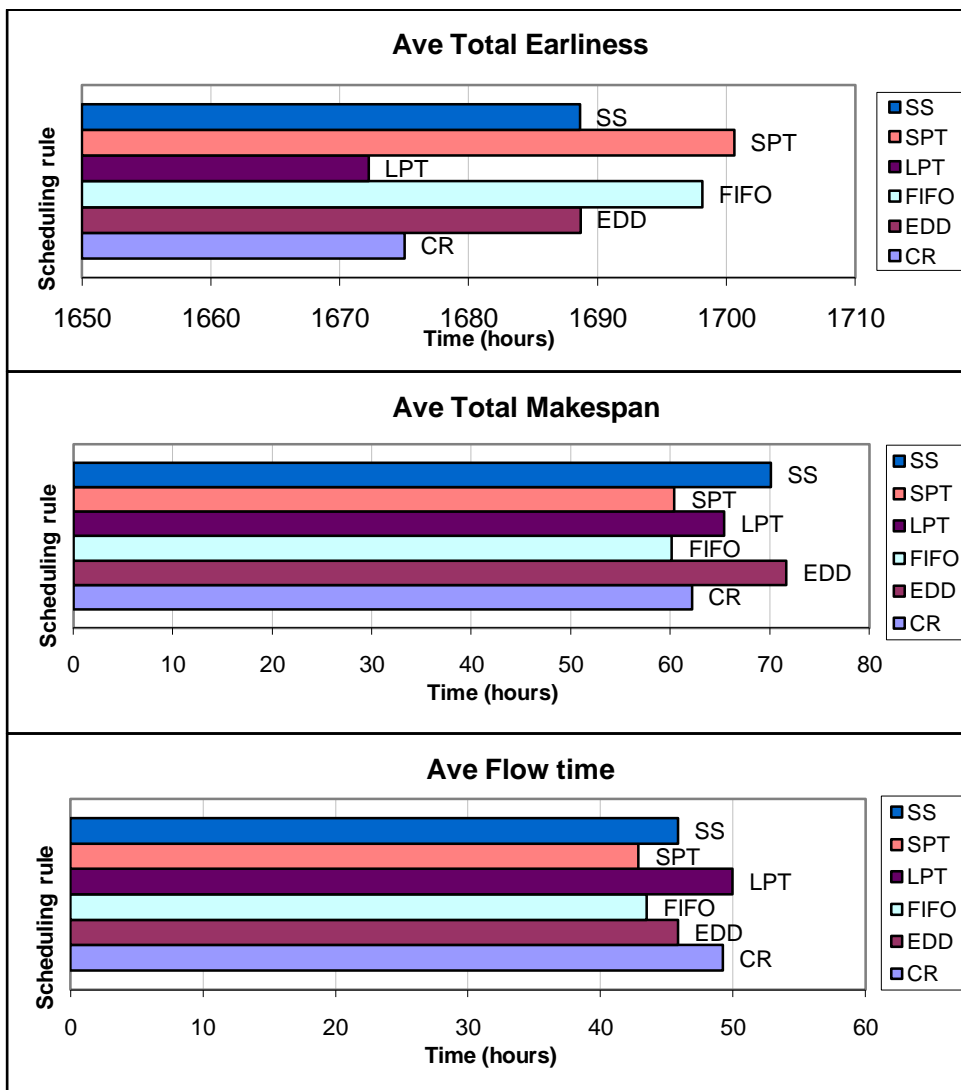


Figure 67 Different schedule performances of Order set 1

The total earliness performance criterion indicates the SPT and FIFO rule will deliver parts the earliest, with SPT outperforming FIFO with two hours. The total makespan and average flow time of SPT and FIFO are almost equal, and shorter than the other rules. Not one of the scheduling rules finished an operation after its due date, and therefore the total lateness performance criterion is omitted. Considering these performance criteria, the best performing rules are FIFO and SPT, and they performed very similar. The FIFO was chosen as it has the shorter makespan, and the proposed schedule for the shop under the FIFO rule is shown in Figure 68.

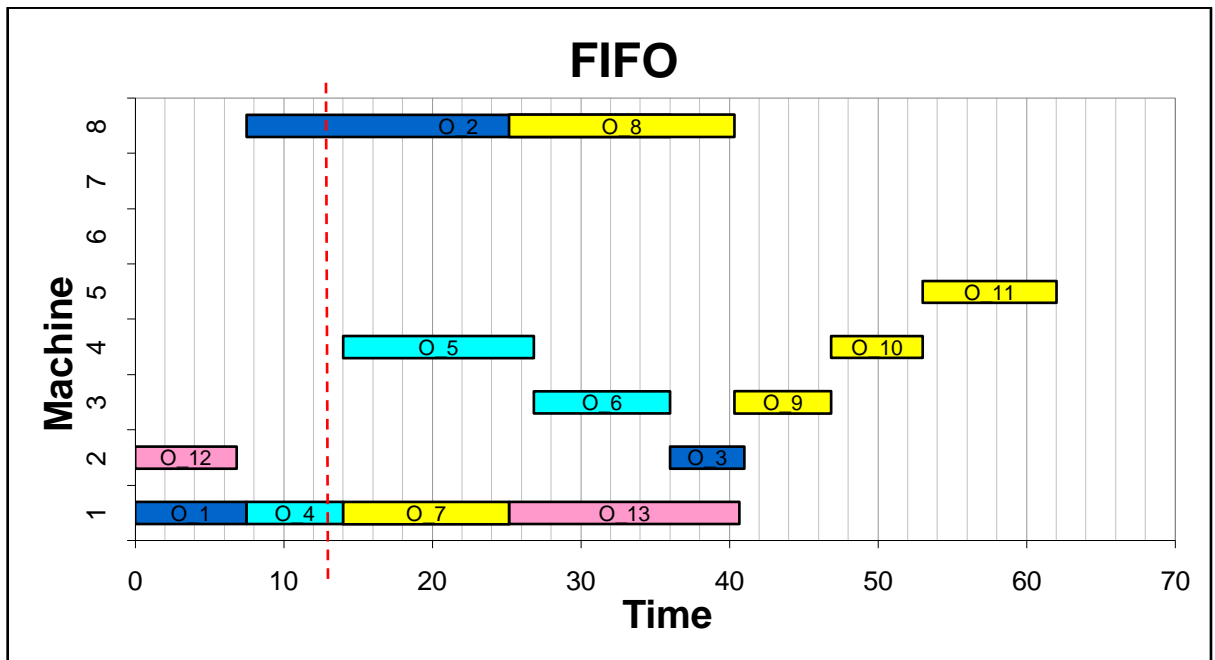


Figure 68 Proposed FIFO schedule for Order set 1

The flow of jobs through the system under the FIFO rule was imitated by a simulation run. The simulation was only run for 12.5 hours as the next set of orders arrived in the system at that time (see Figure 66). The state of the system, i.e. what operations are finished, active and the remaining processing time of the busy operations, is recorded at this point in time. The dotted line in Figure 68 indicates what the system state is at $t = 12.5$ hours. Operation Four (O₄) and Two (O₂) are still busy being processed and Operation One (O₁) and Twelve (O₁₂) are finished. Table 12 shows an illustration of what the database looks like at the current point in time.

Table 12 Order set 1 after 12.5 hours

Job No	Ops No	Mach ID	Mach name	Completed Setup	Completed Insp	Completed Proc	Status Id	Start Time	End Time
1	1	1	V80	2.5	2	3	3	0	7.5
1	2	8	V40_New	2.5	1	1.5	2	7.5	0
1	3	2	V40_Old	0	0	0	0	0	0
2	4	1	V80	2.5	2	0.5	2	7.5	0
2	5	4	T7	0	0	0	0	0	0
2	6	3	Megaturn	0	0	0	0	0	0
3	7	1	V80	0	0	0	1	0	0
3	8	8	V40_New	0	0	0	0	0	0
3	9	3	Megaturn	0	0	0	0	0	0
3	10	4	T7	0	0	0	0	0	0
3	11	5	5Axil	0	0	0	0	0	0
4	12	2	V40_Old	2.5	1	3.3	3	0	6.8
4	13	1	V80	0	0	0	1	0	0

It follows from Table 12 that Operation 1 has started at time zero continuing until time 7.5, the value of its Status Id is 3 which means that it is completed. The successor operation of

job 1, operation 2, is started at time 7.5. Operation 2 is not completed at time 12.5, thus it has a busy status, denoted by 2 in the Status Id field, the completed time is recorded as 2.5 hours setup, 1 hour inspection and 1.5 hours completed process time. The remaining process time is thus 23.5 hours as the expected process time is 25 hours (the expected time of the uniform distribution $U(20,30)$ is $E(x) = \frac{20+30}{2} = 25$). Operation 12 has started on time zero and ended at time 6.8, leaving its successor operation (operation 13) with an active status. Processing of operation 13 has not started as it has been scheduled after other operations, see Figure 68. In the remainder of this section a snapshot of the database at each point in time when new orders arrive could be included, but is omitted as it is seen as duplication. The system is at the following point in time:



After the schedule was followed for 12.5 hours, three orders consisting of nine operations joined the system, shown in Table 13. At this point in time a total of 20 operations, of which two (O_4 and O_2) are busy being processed, are in the system and of these, 18 need to be scheduled. The orders that are busy cannot be stopped due to the pre-emption rule, thus will be forced to start the new schedule that is going to be developed. The scheduling mechanism is activated to compile the different schedules for each scheduling rule for the new order set containing 18 operations.

Table 13 Order set 2

Job No	Approved Date	Prom Date	Ops No	Mach ID	Mach name	Setup timeA	Setup timeB	Setup timeC	Insp time	Proc TimeA	Proc TimeB	Proc TimeC	Status Id
5	2008/06/02	2008/07/30	14	6	Ap20	2	3	0	1	5	8	9	1
5	2008/06/02	2008/07/30	15	7	DMU	2	3	0	1	5	7	15	0
5	2008/06/02	2008/07/30	16	6	Ap20	2	3	0	1	3	20	22	0
6	2008/06/02	2008/06/24	17	6	Ap20	2	3	0	1	8	12	13	1
6	2008/06/02	2008/06/24	18	3	Megaturn	2	3	0	1	2	9	0	0
6	2008/06/02	2008/06/24	19	4	T7	2	3	0	1	3	6	9	0
6	2008/06/02	2008/06/24	20	5	5Axil	2	3	0	1	7	9	0	0
7	2008/06/02	2008/06/30	21	2	V40_Old	2	3	0	1	1	9	11	1
7	2008/06/02	2008/06/30	22	1	V80	2	3	0	1	5	8	0	0

The performance of each schedule developed for the combination of remaining orders in Order set 1 and the new orders in Order set 2 is shown in Figure 69.

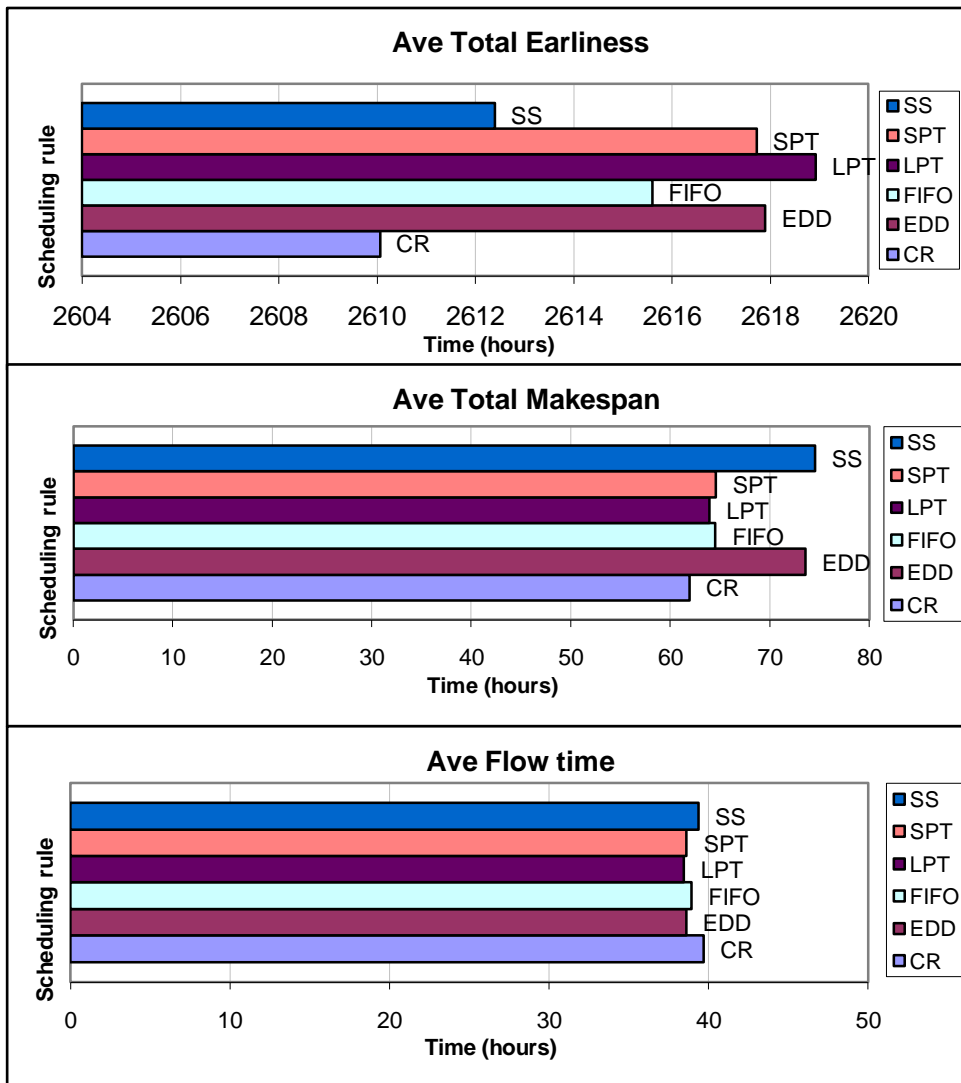


Figure 69 Different schedule performances of Order set 2

The total earliness performance criterion indicates that there is small difference between the rules in terms of the degree that operations are finished before their due dates, the difference between the best and worst performance is only eight hours. The total makespan of CR is the shortest, while the average flow times delivered by the scheduling rules are almost equal. Considering these performance criteria, the CR was chosen as the scheduling rule that will perform the best, and its proposed schedule is shown in Figure 70. Note that the time axis shows a relative time, i.e. $t = 0$ is $t = 12.5$ in real time.

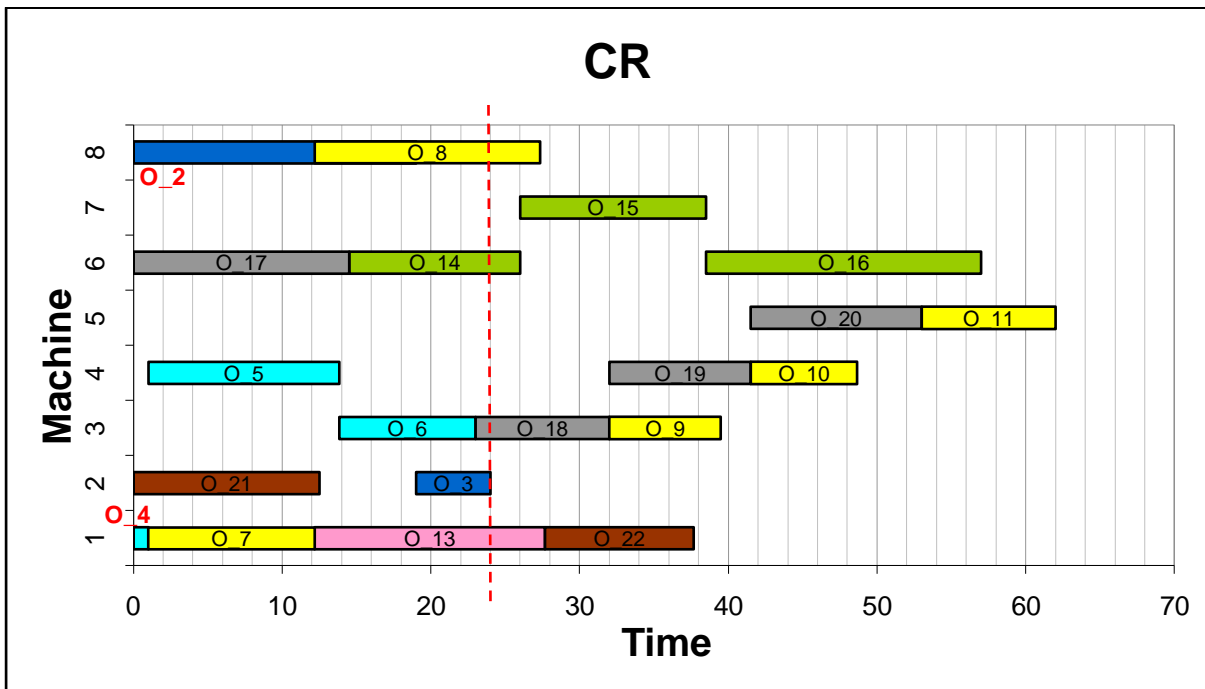
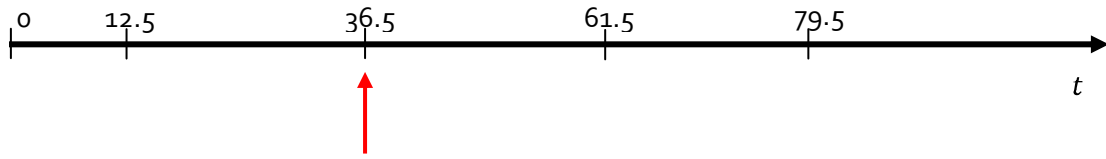


Figure 70 Proposed CR schedule for order set 2

From Figure 70 it follows that the operations that were busy when the new orders arrived (O₄ and O₂) start at time zero as they could not be stopped and processed later; they are processed for their total time less the time they have already been processed according to the previous schedule. Take for example Operation Four (O₄): when the new orders arrived in the system causing the rescheduling, Operation Four was processed for about 5.5 hours. In the new schedule it is processed for an hour, which causes its total processing time over the two schedules to equal 6.5 hours. This is equal to its total time indicated by the user.

The flow of jobs through the system under the CR rule was imitated by a simulation run. The simulation was run for 24 hours (up to 36.5 hours in absolute time, see Figure 66), when the next set of orders arrived in the system. The current state of the system, i.e. what operations are finished, active and the remaining processing time of the busy operations, were recorded. The dotted line in Figure 70 indicates the system state after 24 hours since the last reschedule occurred, five operations (O₃, O₈, O₁₃, O₁₄ and O₁₈) are currently being processed and another eight still need to be processed.



After the schedule was followed for 24 hours since the last reschedule (see time line above), two orders consisting of four operations join the system, shown in Table 14. At this point in time a total of 17 operations, of which five are already busy being processed, are in the system that need to be scheduled. The orders that are busy cannot be stopped, due to the pre-emption rule, thus will be forced to start the new schedule that is going to be developed. The scheduling mechanism is activated to compile the different schedules for each scheduling rule for the new order set containing 12 operations.

Table 14 Order set 3

Job No	Approved Date	Prom Date	Ops No	Mach ID	Mach name	Setup timeA	Setup timeB	Setup timeC	Insp time	Proc TimeA	Proc TimeB	Proc TimeC	Status Id
8	2008/06/05	2008/07/15	23	4	T7	2	3	0	1	1	3	14	1
8	2008/06/05	2008/07/15	24	3	Megaturn	2	3	0	1	8	9	12	0
8	2008/06/05	2008/07/15	25	7	DMU	2	3	0	1	3	8	9	0
9	2008/06/05	2008/06/13	26	2	V40_Old	2	3	0	1	1	17	19	1

The performance of each schedule developed for the combination of remaining orders from the previous order sets and the orders in Order set 3 is shown in Figure 71.

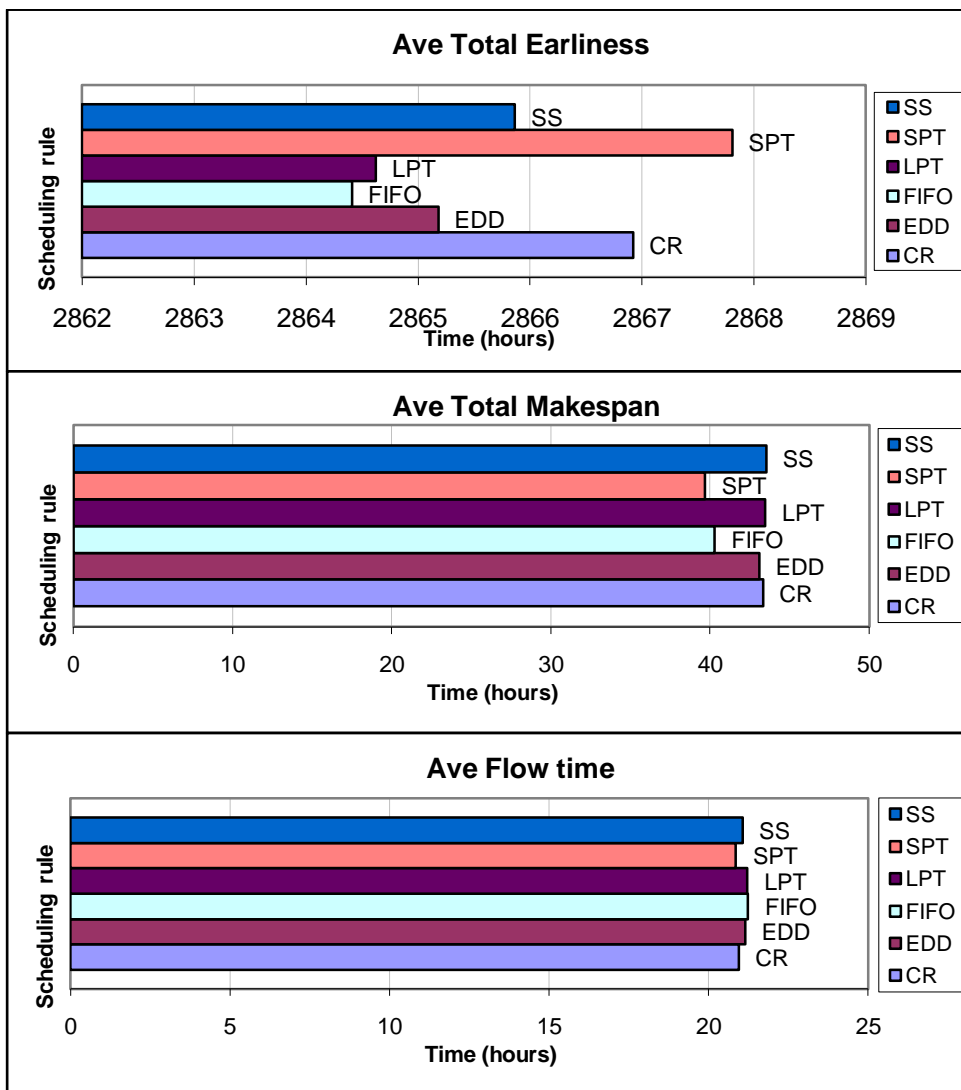


Figure 71 Different schedule performances of Order set 3

The SPT scheduling rule is chosen as it yields the best performance for each of the different criteria, it has the best total time of operations delivered before their due dates, the shortest makespan and the shortest average flow time. The proposed schedule from the SPT rule (see Figure 72), is implemented from this point in time until a new order causes a rescheduling. Note that the time axis shows a relative time, i.e. $t = 0$ is $t = 36.5$ in real time.

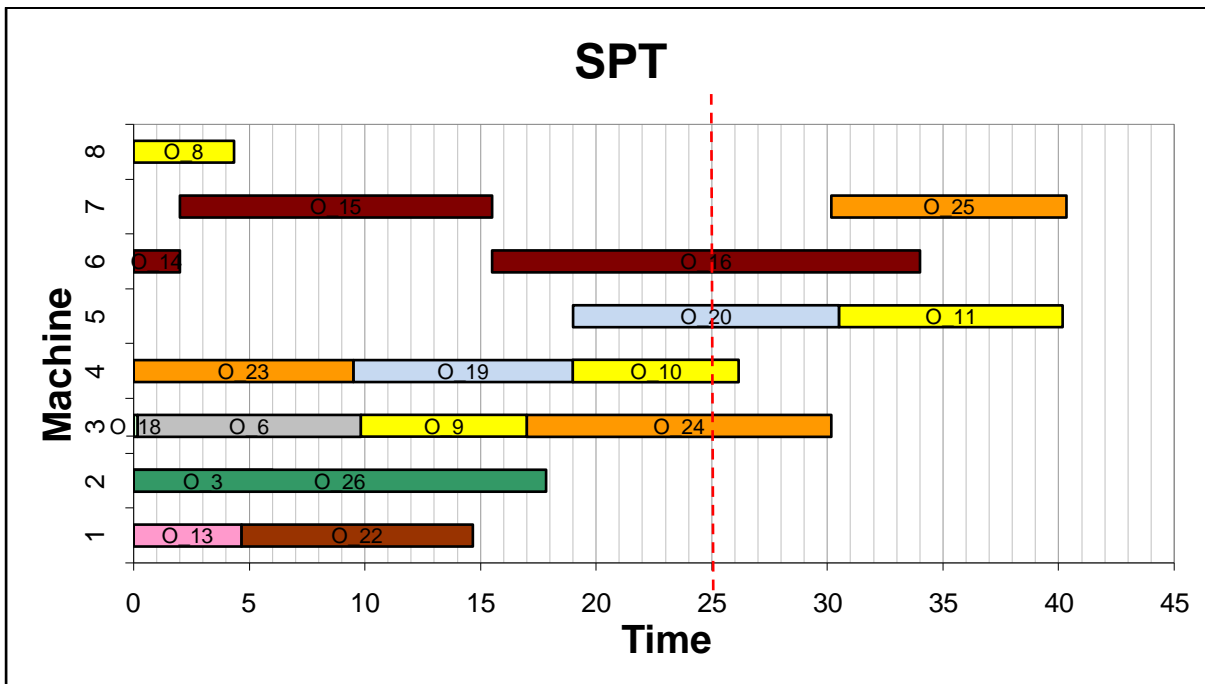
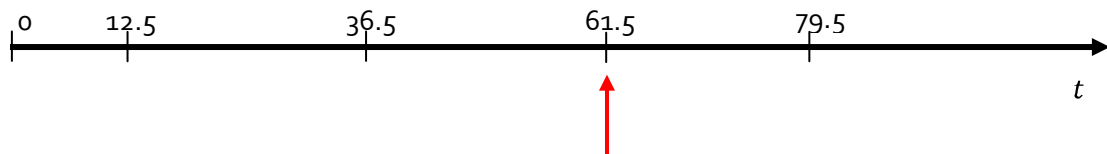


Figure 72 Proposed SPT schedule for order set 3

The flow of jobs through the system under the SPT rule was imitated by a simulation run. The simulation was only run for 25 hours, when the next set of orders arrived in the system. The current state of the system, i.e. what operations are finished, active and the remaining processing time of the busy operations, were recorded. The dotted line in Figure 72 indicates the system state after 25 hours since the last reschedule occurred, four operations (O₁₀, O₁₆, O₂₀ and O₂₄) are currently being processed and two more need to be processed.



After the schedule was followed for 25 hours since the last reschedule (see time line above), six orders consisting of 14 operations join the system, shown in Table 15. At this point in time a total of 20 operations, of which four are already busy being processed, are in the system that need to be scheduled. The orders that are busy cannot be stopped, due to the pre-emption rule, thus will be forced to start the new schedule that is going to be developed, as in previous cases. The scheduling mechanism is activated to compile the different schedules for each scheduling rule for the new order set containing 16 operations.

Table 15 Order set 4

Job No	Approved Date	Prom Date	Ops No	Mach ID	Mach name	Setup timeA	Setup timeB	Setup timeC	Insp time	Proc TimeA	Proc TimeB	Proc TimeC	Status Id
10	2008/06/08	2008/07/30	27	4	T7	2	3	0	1	1	9	20	1
11	2008/06/08	2008/08/30	28	5	5Axil	2	3	0	1	8	9	0	1
11	2008/06/08	2008/08/30	29	7	DMU	2	3	0	1	3	8	14	0
11	2008/06/08	2008/08/30	30	8	V4o_New	2	3	0	1	1	7	0	0
12	2008/06/08	2008/08/12	31	3	Megaturn	2	3	0	1	3	4	13	1
12	2008/06/08	2008/08/12	32	4	T7	2	3	0	1	1	12	13	0
12	2008/06/08	2008/08/12	33	1	V8o	2	3	0	1	2	3	4	1
13	2008/06/08	2008/06/12	34	8	V4o_New	2	3	0	1	20	30	0	0
14	2008/06/08	2008/06/15	35	2	V4o_Old	2	3	0	1	1	5	0	1
14	2008/06/08	2008/06/15	36	1	V8o	2	3	0	1	1	3	14	0
14	2008/06/08	2008/06/15	37	6	Ap2o	2	3	0	1	7	8	0	0
15	2008/06/08	2008/06/25	38	3	Megaturn	2	3	0	1	1	7	12	1
15	2008/06/08	2008/06/25	39	1	V8o	2	3	0	1	3	8	0	0
15	2008/06/08	2008/06/25	40	8	V4o_New	2	3	0	1	10	12	19	0

The performance of each schedule developed for the combination of remaining orders from the previous order sets and the orders in Order set 4 is shown in Figure 73.

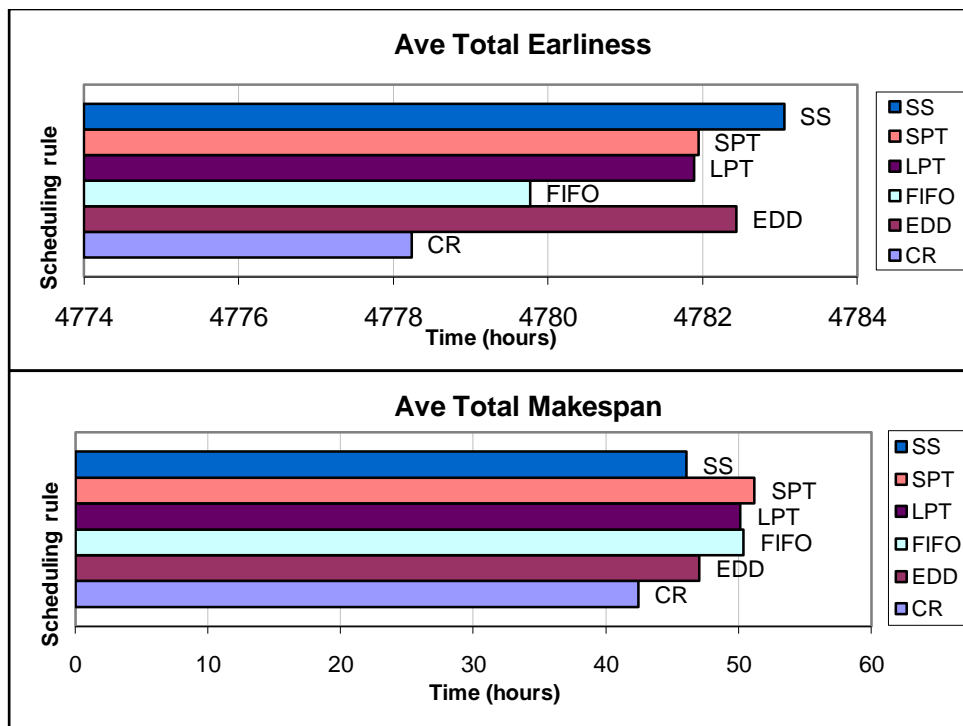


Figure 73 Different schedule performances of Order set 4 (continued on next page)

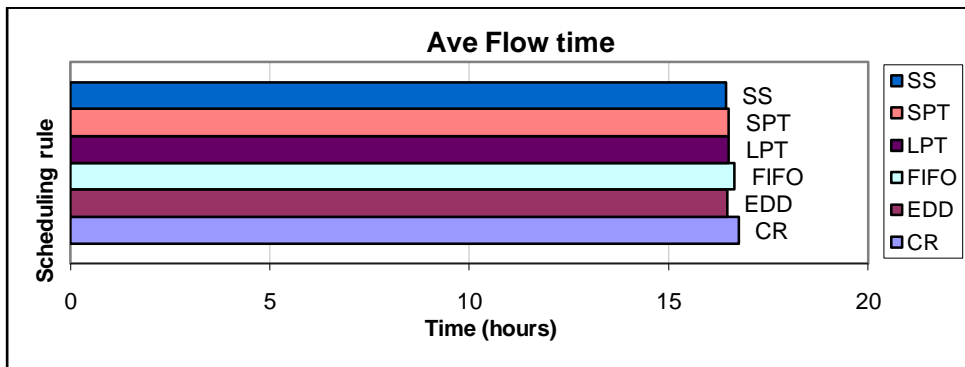


Figure 73 Different schedule performances of Order set 4

According to Figure 73, the different schedules do not differ much in terms of earliness and average flow time, the difference between the minimum and maximum values for the earliness criterion is only five hours and the difference between the minimum and maximum values for the flow time criterion is less than an hour. The makespan performance is thus used to determine which scheduling rule has the best performance, resulting in implementing the CR schedule as it has the smallest makespan. The proposed schedule developed with the CR rule is shown in Figure 74. Note that the time axis shows a relative time, i.e. $t = 0$ is $t = 61.5$ in real time.

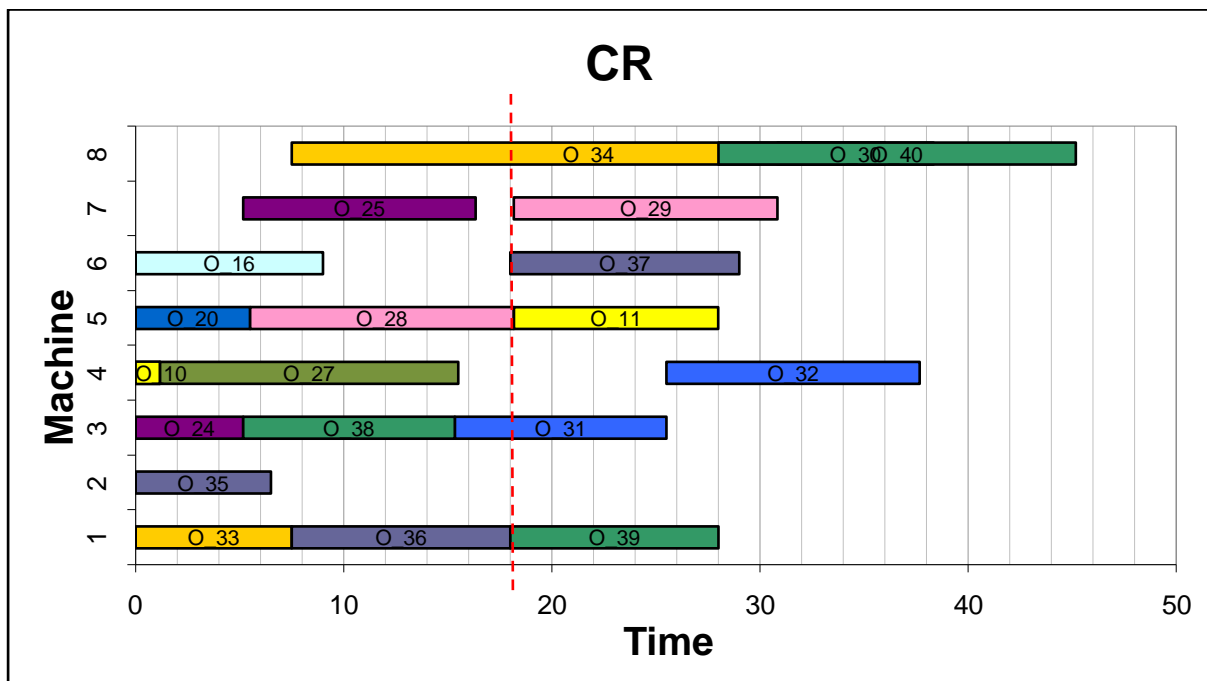
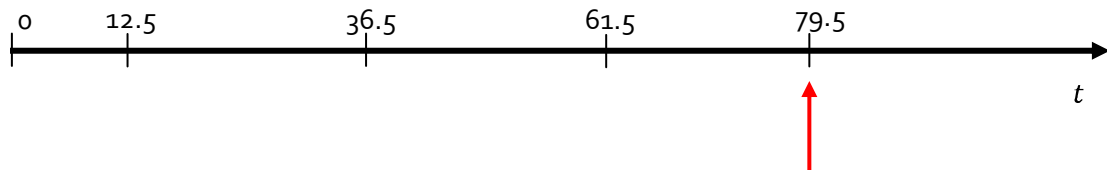


Figure 74 Proposed CR schedule for order set 4

The flow of jobs through the system under the CR rule was imitated by a simulation run. The simulation was only run for 18 hours, when the next set of orders arrived in the system. The current state of the system, i.e. what operations are finished, active and the remaining processing time of the busy operations, were recorded. The dotted line in Figure 74 indicates the system state after 18 hours since the last reschedule occurred, three operations (O₂₈, O₃₁ and O₃₄) are currently being processed and six more needs to be processed.



After the schedule was followed for 18 hours since the last reschedule, five orders consisting of nine operations join the system, shown in Table 16. At this point in time a total of 18 operations, from which three are already busy being processed, are in the system that needs to be scheduled. The orders that are busy cannot be stopped, due to the pre-emption rule, thus will be forced to start the new schedule that is going to be developed. The scheduling mechanism is activated to compile the different schedules for each scheduling rule for the new order set containing 15 operations.

Table 16 Order set 5

Job No	Approved Date	Prom Date	Ops No	Mach ID	Mach name	Setup timeA	Setup timeB	Setup timeC	Insp time	Proc TimeA	Proc TimeB	Proc TimeC	Status Id
16	2008/06/10	2008/07/25	41	3	Megaturn	2	3	0	1	1	4	8	1
17	2008/06/10	2008/07/01	42	1	V8o	2	3	0	1	3	8	14	1
17	2008/06/10	2008/07/01	43	8	V4o_New	2	3	0	1	8	12	13	0
18	2008/06/10	2008/07/18	44	3	Megaturn	2	3	0	1	2	6	0	1
19	2008/06/10	2008/08/04	45	3	Megaturn	2	3	0	1	2	9	12	1
19	2008/06/10	2008/08/04	46	4	T7	2	3	0	1	3	6	0	0
20	2008/06/10	2008/07/27	47	5	5Axil	2	3	0	1	7	12	0	1
20	2008/06/10	2008/07/27	48	2	V4o_Old	2	3	0	1	1	7	11	0
20	2008/06/10	2008/07/27	49	1	V8o	2	3	0	1	5	16	0	0

The performance of each schedule developed for the combination of remaining orders from the previous order sets and the orders in Order set 5 is shown in Figure 75.

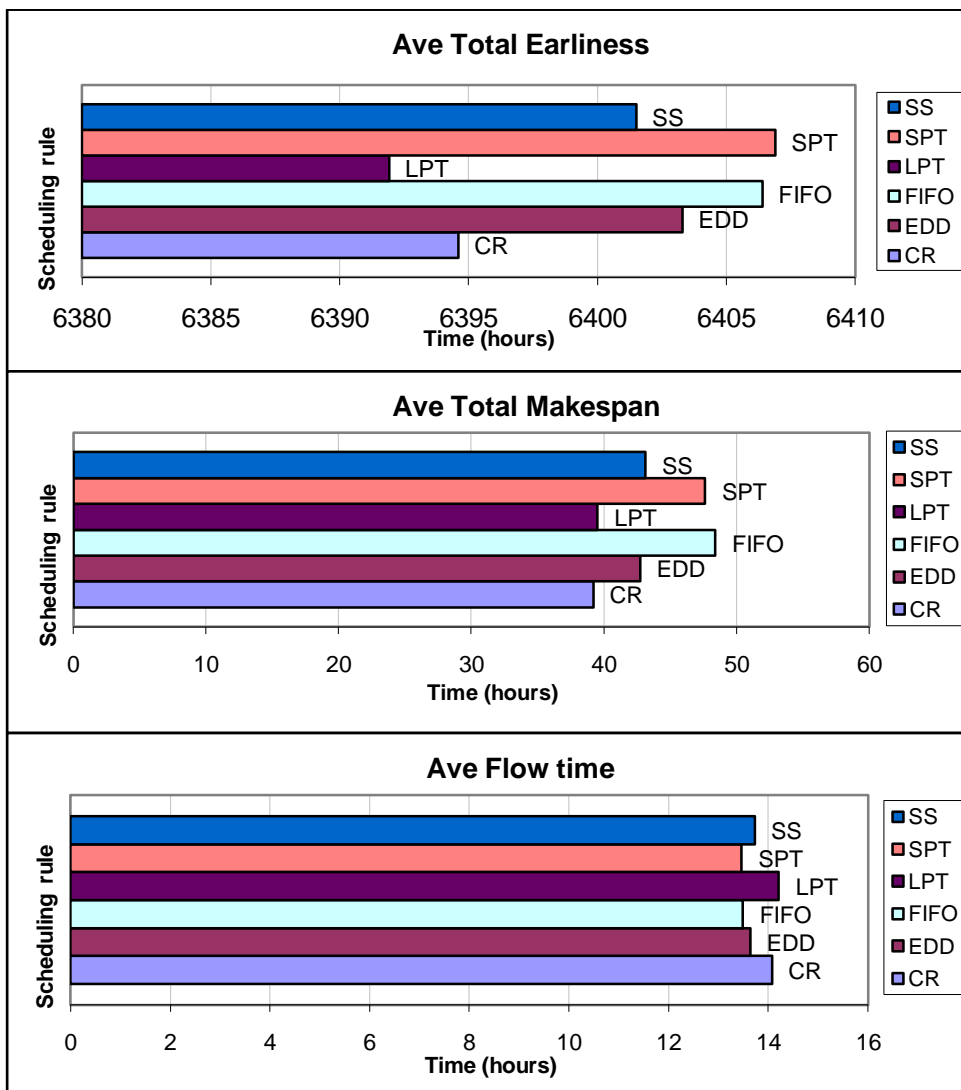


Figure 75 Different schedule performances of Order set 5

The difference between the flow time performances of the schedules is very small and therefore not considered when comparing the schedules. Although LPT and CR have worse earliness performances than the other schedules, their better performing makespans make them the better schedules to follow as makespan is considered more important. The difference between the makespan performances of the two schedules is very small, their earliness performance is therefore used to determine which schedule is the best. CR has a better earliness performance and is therefore considered to be the best current scheduling rule, and the consequent schedule developed with the CR rule is shown in Figure 76. Note that the time axis shows a relative time, i.e. $t = 0$ is $t = 79.5$ in real time.

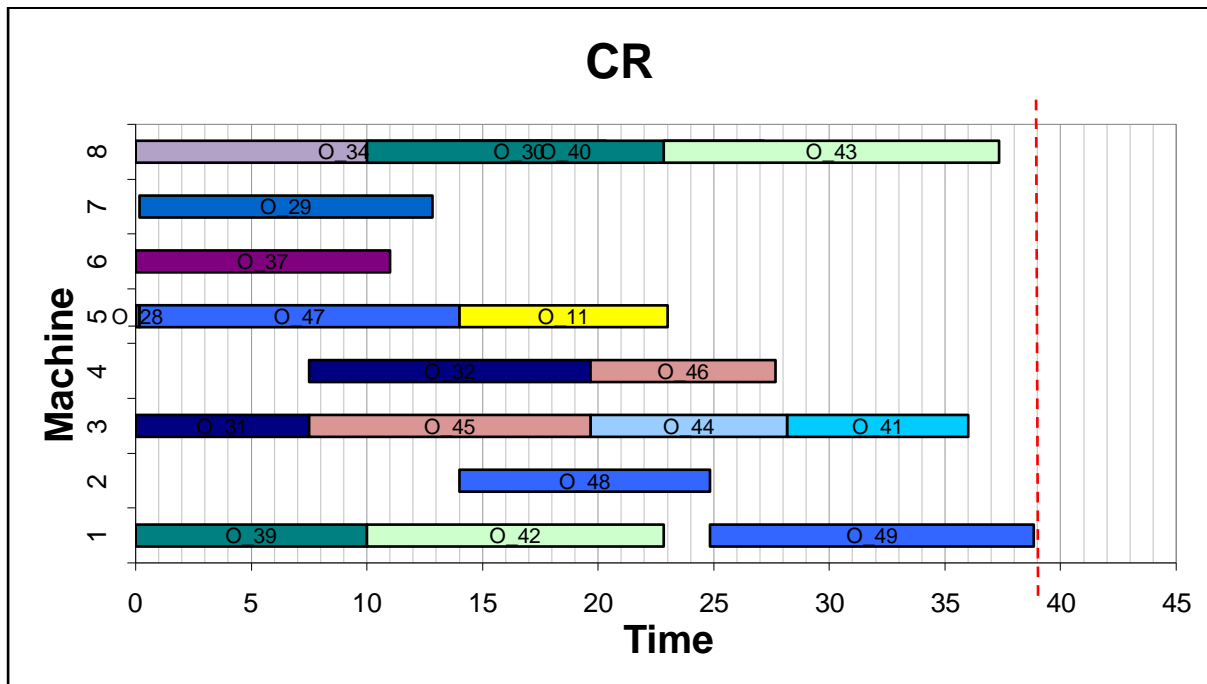


Figure 76 Proposed CR schedule for order set 5

The flow of operations through the system under the CR rule was imitated by a simulation run. The simulation was run until all the operations were finished as no new orders entered the system.

This concludes the demonstration example of how the scheduling mechanism functions in a stochastic environment. Next this chapter is concluded with a discussion on how the online capability of the scheduling mechanism is revealed.

10.7 CHAPTER CONCLUSION

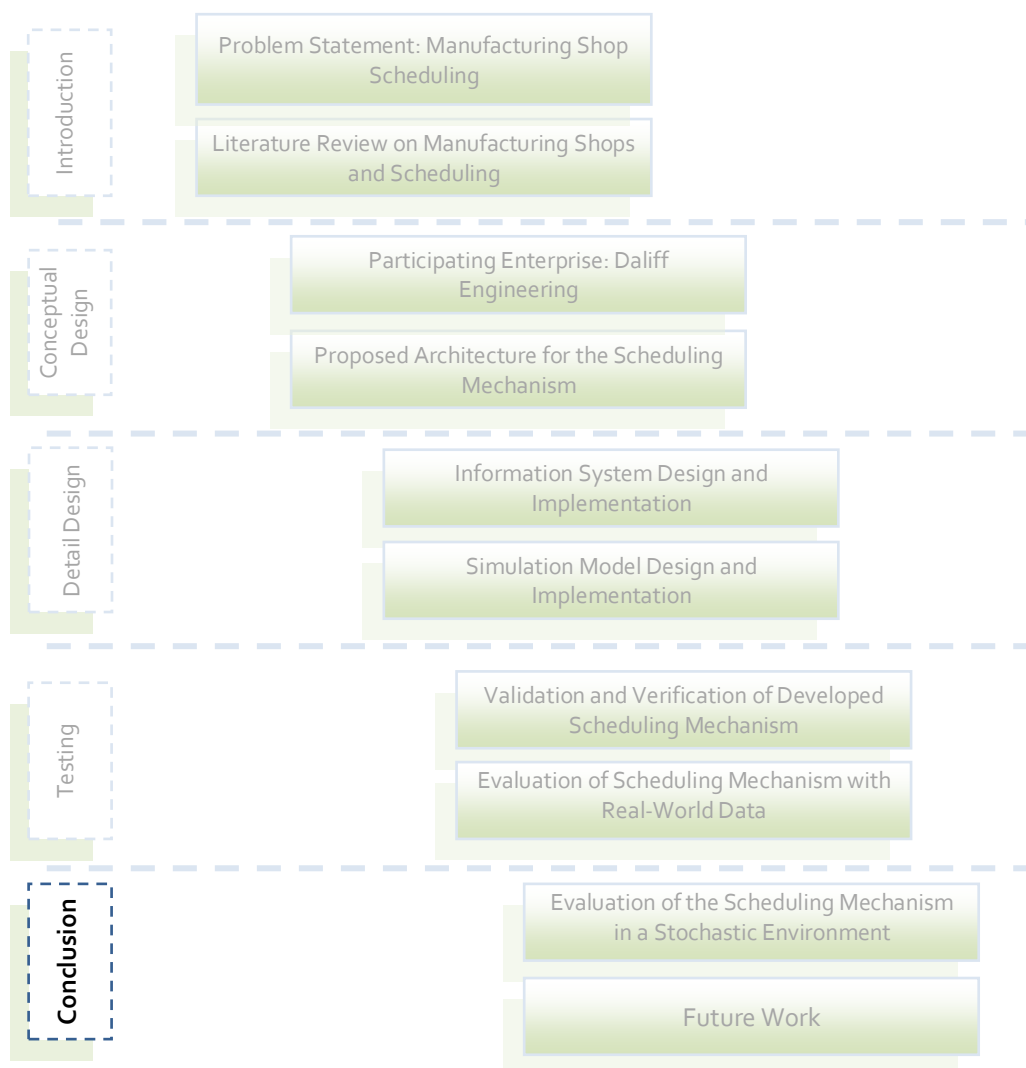
The online scheduling capability of the scheduling mechanism is confirmed by this demonstration example. In the fast changing environment where the distant future was unknown, a schedule was developed according to *what was known*, this schedule was followed until *what was known* was changed by the arrival of a new order. A new schedule was then easily developed using the new *what was known* information. Every time a new schedule was developed it was believed to be the best schedule for *what was known*, and it is followed until more is known. This phenomenon is called myopic scheduling, also known as short-sighted scheduling.

In the demonstration, a "best scheduling rule" was selected via reasoning at each reschedule epoch. Future work on this problem should consider a normalized, and possibly weighted function that considers all performance criteria which can lead to automatically ranking and selection of the best rule. Suggested methods are Simple Additive Weighting (see Ma *et al.* [58]) and TOPSIS (see Hwang and Yoon [59] and Jahanshahloo *et al.* [60]).

A conclusion of the work done for the purpose of this thesis is stated in the next chapter.

11. CONCLUSION

Drawing a conclusion is the final phase of the thesis road map as shown below. The results obtained in this study have already been stated and a conclusion can now be made, which is stated in this chapter.



Solving the problem of online scheduling of a manufacturing shop that has a high arrival rate of orders, where orders consist of small quantities of complex parts, which are unique to the order, was stated as the aim of this research. Using simulation as scheduling device was suggested as the tool of choice.

After a survey of the scheduling literature, and specifically scheduling in job-shops, an architecture was developed for the proposed scheduling mechanism. It required the use of an information system, a simulation model and a real-world shop-floor.

The implementation of the architecture for a local enterprise was discussed thereafter, describing the information system and simulation model design and implementation. The resulting schedules that were obtained from the proposed scheduling mechanism for the enterprise were stated and discussed.

These resulting schedules indicate that the scheduling mechanism developed according to the proposed architecture for the enterprise is indeed applicable to use as a scheduler, as the schedules delivered predict better performance of the manufacturing shop. It was also shown how the scheduler functions in a stochastic environment.

11.1 SUGGESTIONS TO DALIFF ENGINEERING REGARDING THE IMPLEMENTATION OF THE SCHEDULING MECHANISM

Before implementing the current scheduling mechanism, Daliff Engineering should run a trial period, during which the scheduling mechanism is run parallel to the current production process, while keeping the information system up to date as business changes. It will entail extra work as a person has to collect system state information from the shop floor manually and enter it into the information system. It is also suggested that the person that will be responsible for the total implementation of the scheduling mechanism and eventual user of it, is responsible for the trial period and the work associated. Conducting the trial period will allow the person to get grips with the scheduler and gain valuable experience regarding the scheduler and its functions.

The first stage of the implementation process will be to implement the information system. The ideal is to get the information system fully automated, by using programmable logic controllers and a computer network. Detail on this will be left for the implementation process. When the information system is fully incorporated into the enterprise's daily activities and is concurrently updated with the shop-floor status, the scheduling model can be implemented. First the scheduler must not be allowed to control the shop floor according to the schedules it generates, rather implement the schedules through following schedule sheets manually. This enables the person responsible for the implementation to

verify the schedules before implementation. When the schedules developed by the scheduling mechanism are deemed acceptable, control can be given to the scheduling mechanism.

During the trial and implementation periods, the person responsible is advised to always look for possible improvements by modifications as it will satisfy the concept of *Quality at the Source*, eliminating time consuming problems at full implementation.

11.2 COMPLIANCE WITH ACADEMIC REQUIREMENTS

The academic requirements were stated at the beginning of this thesis in the Terms of Reference section. The work reported in this thesis confirms that the requirements were met. The figure on the next page is a summary of the work completed in compliance with the academic requirements.

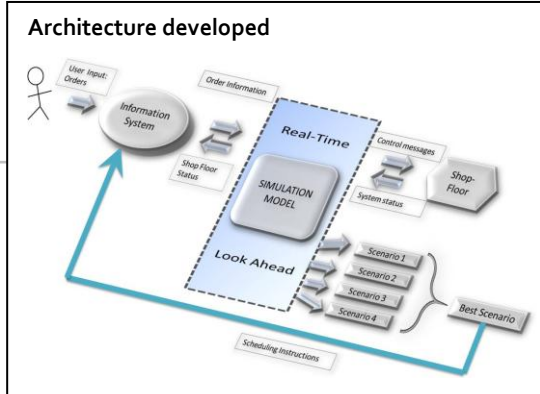
The academic requirements stated in the Terms of Reference section are repeated in the figure, but are faded. It acts as a skeleton framework to represent the work that is completed. A list of the characteristics of the components developed to comply with the requirement is included along with a visual representation of the component. The interfaces between the components are shown by the arrows.

In previous sections, the functionality of the scheduling mechanism is proven to be satisfactorily and it achieves desirable results. The scheduling mechanism can now be used in the Department of Industrial Engineering at the Stellenbosch University as platform for further studies regarding simulation-based online scheduling. Some recommendations are made in the next section.

Online Scheduling Mechanism

Information System developed

- Frontpage and MS Access
- ASP code



Input: Information System

- Enterprise status
- Order information

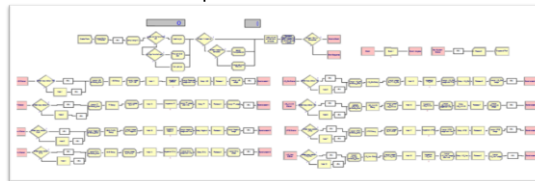
Make-to-Order

- Fast changing
- Small order sizes
- Once-off production

Interface established

Simulation Model Developed

- Rockwell Arena
- VBA code
- Adapts to current system state
- Implements stochastic processing times
- Implements scheduling rules
- Develops schedules
- Measures performance



```
Private Sub DetermineStartTimes(Origins As LongArray)
    pt = Entries(EntryID) - temp + a, Entries(a)
    While a = 1 then it is the first entity of the part set
    If a = 1 Then
        *start times can only change while entity is inactive.
        *except for the first instance of a part set it can change if
        *its not completed
        If Entries(EntryID) - temp + a, Entries(a) = Active Then
            Slack = Entries(EntryID) - temp + a, Entries(a) - Entries(EntryID) - temp + a, Entries(a)
            Check if it is not the first time that start times are being calculated
            origin = 0 for the first time start times are being calculated
            *origin = 0 then
            *get the new start time of the first entity in the part set, it will be the difference
            *between the original start time and the current start time (assumed to start on current time)
            If Entries(EntryID) - temp + a = 0, Entries(a) = Entries(EntryID) - temp + a, Entries(a) Then
                at = (Entries(EntryID) - temp + a, Entries(a)) - Entries(EntryID) - temp + a, Entries(a)
                Entries(EntryID) - temp + a, Entries(a) = at
            Slack = Entries(EntryID) - temp + a, Entries(a) - Entries(EntryID) - temp + a, Entries(a) - Entries(EntryID) - temp + a, Entries(a)
            End If
        End If
    End Sub
```

Manufacturing process

Industry Partner

- Make-to-order

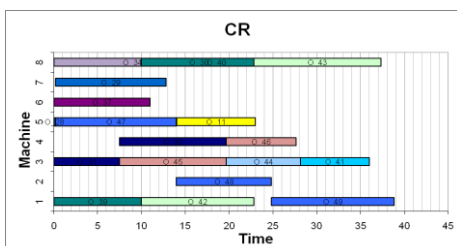
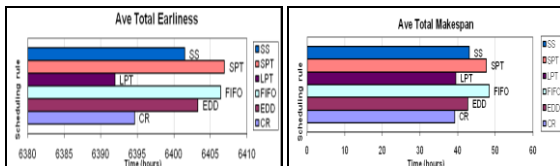


Means: Simulation Model

- Discrete event simulation
- Stochastic
- Adaptable
- Established scheduling rules
- Develop schedules
- Record performances

Output developed

- MS Excel
- Measures according to performance criteria
- Delivers schedule



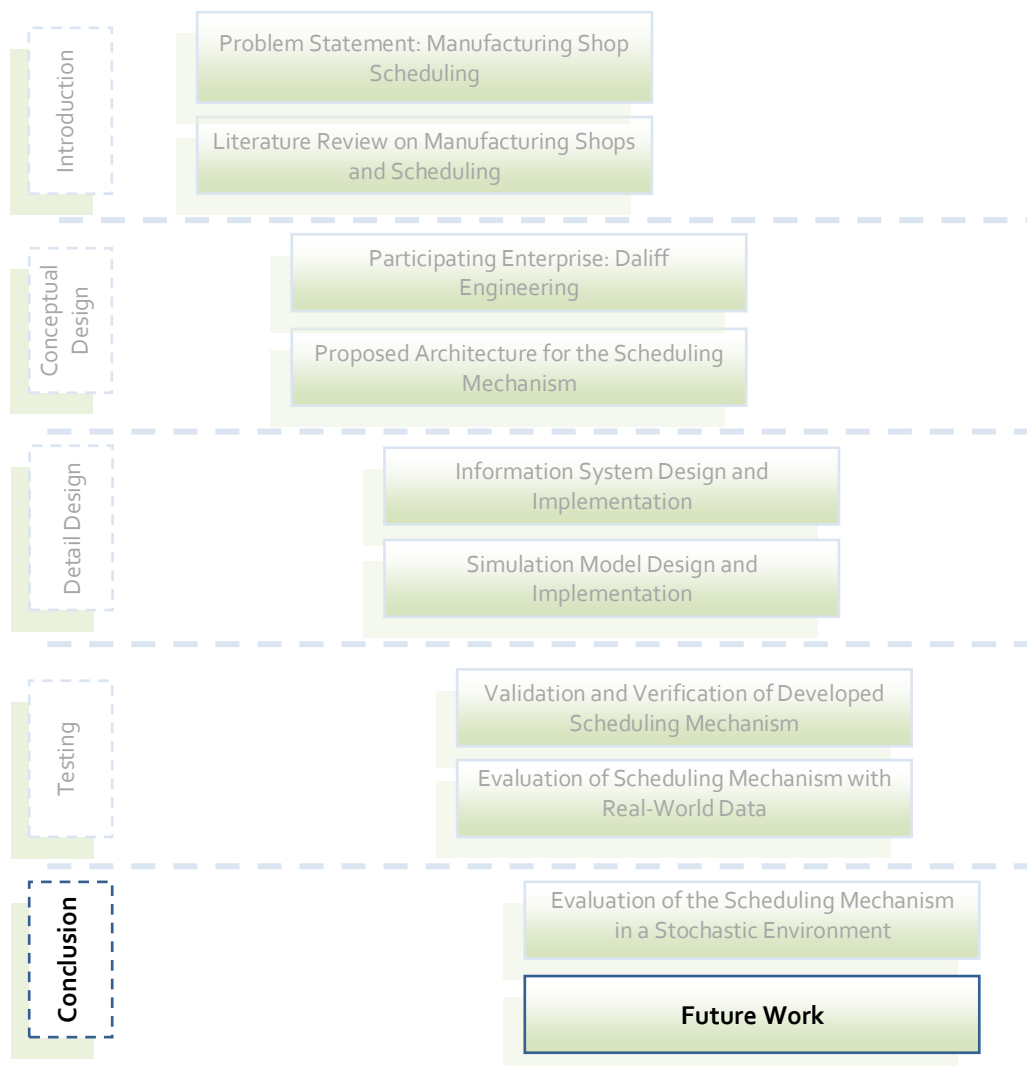
Interface established

Output: Excel Worksheet and Gantt-style Chart

- Myopic
- Established performance criteria

12. SUGGESTED FURTHER DEVELOPMENTS OF THE SCHEDULING MECHANISM

The options of further modifications and alterations of the scheduling mechanism developed in this study are endless. This chapter states some of these options as identified by the student (see road map below).



Enhancing the online capabilities of the mechanism by incorporating real time control can be a challenging, but worthwhile exercise. Real time control, in this context, entails the control of the manufacturing shop processes by the scheduling mechanism. The mechanism would thus implement the schedule that was developed by itself, whilst being a controller the response to system disturbances will be much quicker as the scheduler is directly connected to the shop. Arena RT is the suggested tool as the scheduler already uses Arena for its scheduling component.

Enhancing the scheduler by adding learning capabilities would answer the concern presented in the evaluation sections. With learning capabilities the scheduler will in a sense get to know the shop floor and production environment. If records are kept of the schedule rules that were implemented and those that were not, it could be used to evaluate the functionality of the scheduler on the specific shop floor. Take for example the situation that was noticed during the evaluation process. It was noticed that using only the SPT rule in that particular time frame instead of the scheduling rule that performed best for each simulation period, a better result could have been achieved. If it were the case that after examining the records that were kept, as explained above, over a long period of time, it became evident that the SPT rule in fact would have resulted in better results. A conclusion could be made with strong certainty that the shop floor must rather be run on a SPT rule basis, and the need for online scheduling could consequently disappear. Whether this will happen or not is open to speculation and would only become evident after the scheduler is implemented and used for a long period.

Incorporating learning capabilities for the probability distributions of the processing times can be included, to make the scheduling mechanism evolve as time elapses.

Alternative machine routing capabilities will enhance the scheduling capabilities of the mechanism. Giving the scheduler a set of machines on which an operation can be performed will enable it to shorten schedules, but at the same time make the scheduling problem more complex.

More scheduling rules can be made available for the user to choose from, or better yet, automate the process of running the simulation model for all the scheduling rules, including the newly added ones. The comparison of the performance of the different scheduling rules can then also be automated. The scheduler will eventually be fully automated and the need for any human intervention will be unnecessary.

Instead of evaluating the performance of each schedule separately and visually, an objective function could be developed in which each performance criterion has a certain weight assigned to it. Each schedule will thus only have one performance criterion value that enables an easy mathematical comparison of schedules.

A test environment could also be developed to test and 'play' with the scheduling mechanism. A typical environment can be built using programmable logic controllers (PLC) that represents resources. The interface of the scheduling mechanism with the real-world manufacturing shop and its response to system disturbances can be demonstrated and tested to some extreme.

13. REFERENCES

- [1] Womack JP, Jones DT, Roos D. *The Machine that Changed the World*. New York: Rawson Associates; 1990. pp. 11-15.
- [2] Groover MP. *Automation, Production Systems and Computer-Integrated Manufacturing*, Englewood Cliffs, NJ: Prentice Hall, Inc.; 1987. pp. 1-9.
- [3] Shnits B, Rubinovitz J, Sinreich D. Multi-criteria dynamic scheduling methodology for controlling a flexible manufacturing system. *International Journal of Production Research* 2004; 42(17):3457-3472.
- [4] Shnits B, Sinreich D. Controlling flexible manufacturing systems based on a dynamic selection of the appropriate operational criteria and scheduling policy. *International Journal of Flexible Manufacturing Systems* 2006; 18:1–27
- [5] Banks J. *Handbook of Simulation*. John Wiley and Sons, Inc; 1998. pp. 465-515.
- [6] Byrne MD, Chutima P. Real-time Operational Control of a FMS with Full Routing Flexibility. *International Journal of Production Economics* 1997; 51(1- 2):109-113.
- [7] Kim MH. Simulation-based real-time scheduling in a flexible manufacturing system. *Journal of Manufacturing Systems* 1994; 13(2):85-93.
- [8] Tung L-F, Lin L, Nagi R. Multiple-objective scheduling for the hierarchical control of flexible manufacturing systems. *International Journal of Flexible Manufacturing Systems* 1999; 11(4):379-409.
- [9] Hopp WJ, Spearman ML. *Factory Physics: Foundations of Manufacturing Management*. Second edition. New York: Irwin McGraw-Hill; 2001.
- [10] Pinedo M. *Scheduling Theory, Algorithms, and Systems*. Prentice Hall international series in industrial and systems engineering; 1995.
- [11] Perregaard M, Clausen J. Parallel branch-and-bound methods for the job-shop scheduling problem. *Annals of Operations Research* 1998; 83:137–160.
- [12] Sadeh N. *Look-Ahead Techniques for Micro-Opportunistic Job-shop Scheduling* [Doctoral Thesis]. Carnegie Mello University; Pennsylvania; 1991.
- [13] Leung J Y-T. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman & Hall/CRC. 2004.
- [14] Fisher H, Thompson GL. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling*. Prentice-Hall, Englewood Cliffs. 1963; pp. 225–241.

-
- [15] Schutten JMJ. Practical job-shop scheduling. *Annals of Operations Research* 1998; 83:161–177.
- [16] Pezzella F, Morganti G, Ciaschetti G. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers & Operations Research* 2008; 35:3203 – 3212.
- [17] Baker KR. *Introduction to sequencing and scheduling*, Wiley & Sons: New York; 1974.
- [18] Church LK, Uzsoy R. Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer-Integrated Manufacturing* 1992; 5:153–163.
- [19] Artigues C, Michelon P, Reusser S. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research* 2003; 149(2): 249–267.
- [20] Wysk RA, Smith JS. A Formal Characterization of Shop Floor Control. *Computers in Industrial Engineering* 1995; 28(3):631-644.
- [21] Yamamoto Y, Nof SY. Scheduling/Rescheduling in the Manufacturing Operating System Environment. *International Journal of Production Research* 1985; 23(4):705-722.
- [22] Yih Y, Thesen A. Semi-Markov Decision Models for Real-time. *International Journal of Production Research* 1991; 29(11):2331-2346.
- [23] Maley JG, Ruiz-Meir S, Solberg JJ. Dynamic Control in Automated Manufacturing: A Knowledge-Integrated Approach. *International Journal of Production Research* 1988; 26(11):1739-1748.
- [24] Sarin SC, Salgame RR. Development of a Knowledge-Based System for Dynamic Scheduling. *International Journal of Production Research* 1990. 28(8):1499-1512.
- [25] Maimon OZ. Real-time Operational Control of Flexible Manufacturing Systems. *Journal of Manufacturing Systems* 1987; 6(2):125-136.
- [26] Shaw MJ. Dynamic scheduling in Cellular Manufacturing Systems: A Framework for Networked Decision Making. *Journal of Manufacturing Systems* 1988; 7(2):83-94.
- [27] Shaw MJ, Park S, Raman N. Intelligent scheduling with machine learning capabilities: the induction of scheduling knowledge. *IIE Transactions* 1992; 24:156-168.

- [28] Piramuthu S, Raman N, Shaw MJ. Learning-based scheduling in a flexible manufacturing flow line. *IEEE Transactions on Engineering Management* 1994; 41:172–182.
- [29] Sun Y-L, Yih Y. An intelligent controller for manufacturing cells. *International Journal of Production Research* 1996; 34:2353–2373.
- [30] Soon TH, Desouza R. Intelligent simulation-based scheduling of workcells: an approach. *Integrated Manufacturing Systems* 1997; 8:6–23.
- [31] Min HS, Yih Y, Kim C-O. A competitive neural network approach to multiobjective FMS scheduling. *International Journal of Production Research* 1998; 36:1749–1765.
- [32] Jahangirian M, Conroy GV. Intelligent dynamic scheduling system: the application of genetic algorithms. *Integrated Manufacturing Systems* 2000; 11:247–257.
- [33] Smith JS, Wysk RA, Sturrock DT, Ramaswamy SE, Smit GD, Joshi SB. Discrete Event Simulation for Shop Floor Control. In: Tew JD, Manivannan MS, Sadowski DA and Seila AF, editors. *Simulation: Proceedings of the 1994 Winter Simulation Conference*. Florida: Lake Buena Vista; 1994. pp. 962-969.
- [34] Jones A, Rabelo L, Yuehwern Y. A Hybrid Approach for Real-Time Sequencing and Scheduling. *International Journal of Computer Integrated Manufacturing* 1995; 8(2):145-154.
- [35] Julia S, Valette R. Real time scheduling of batch systems. *Simulation Practice and Theory* 2000; 8:307-319.
- [36] Kim MH, Kim YD. Simulation-based real-time scheduling in a flexible manufacturing system. *Journal of Manufacturing Systems* 1994; 13:85–93.
- [37] Jeong KC, Kim YD. A real-time scheduling mechanism for a flexible manufacturing system: using simulation and dispatching rules. *International Journal of Production Research* 1998; 36:2609–2626.
- [38] Cho H, Wysk RA. A robust adaptive scheduler for an intelligent workstation controller. *International Journal of Production Research* 1993; 31:771–789.
- [39] Ishii N, Talavage JJ. A transient-based real-time scheduling algorithm in FMS. *International Journal of Production Research* 1991; 29:2501–2520.
- [40] Wu S-Y D, Wysk RA. An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing. *International Journal of Production Research* 1989; 27:1603–1623.

-
- [41] Kim C-O, Min H-S, Yih Y. Integration of inductive learning and neural networks for multi-objective FMS scheduling. *International Journal of Production Research* 1998; 36:2497–2509.
- [42] Aydin ME, Oztemel E. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems* 2000; 33:169–178.
- [43] Reisin-Fournier F. Scheduling with learning capabilities. DSc thesis. Technion. Israel Institute of Technology; 1998.
- [44] Kazerooni A, Chan FTS, Abhary K. A fuzzy integrated decision-making support system for scheduling of FMS using simulation. *Computer Integrated Manufacturing Systems* 1997; 10:27–34.
- [45] Fanti MP, Maione B, Naso D, Turchiano B. Genetic multi-criteria approach to flexible line scheduling. *International Journal of Approximate Reasoning* 1998; 19:5–21.
- [46] Buyurgan N, Mendoza A. Performance-based dynamic scheduling model for flexible manufacturing systems. *International Journal of Production Research* 2006; 44(7):1273-1295.
- [47] Montazeri M, van Wassenhove LN. Analysis of scheduling rules for an FMS. *International Journal of Production Research* 1990; 28(4):785-802.
- [48] Vinod V, Sridharan R. Simulation-based metamodels for scheduling a dynamic job-shop with sequence-dependent setup times. *International Journal of Production Research* 2007; 1-23.
- [49] Conway RW. Priority dispatching and work-in-progress inventory in a job-shop. *Journal of Industrial Engineering* 1965; 16(2):123-130.
- [50] Hershauer JC, Ebert J. Search and simulation selection of a job-shop scheduling rule. *Management Science* 1975; 21(7):833-843.
- [51] Blackstone JH Jr, Phillips DT, Hoog GL. A state-of-the-art survey of dispatching rules for manufacturing job-shop operations. *International Journal of Production Research* 1982; 20(1):27-45.
- [52] McCartney J, Hinds BK. Interactive scheduling procedures for flexible Manufacturing Systems. *Proceedings of 22nd International Machine Tool Design and Research Conference*. Manchester; September 1981. pp. 47-54.
- [53] Dar-El EM, Wysk RA. Job-shop scheduling: A systematic approach. *Journal of Manufacturing Systems* 1982; 1(1):77-88.

-
- [54] Bekker J. Simulation. Unpublished course notes for the module Simulation 873. Stellenbosch: University of Stellenbosch; 2007.
- [55] Slota A, Malopolski W. Integration of simulation software Arena with FMS control system. *International Journal of Simulation Modeling* 2007; 6(3):165-172.
- [56] Drake GR, Smith JS. Simulation system for real-time planning, scheduling, and control. *Proceedings of the 28th Winter Simulation Conference* 1996; pp.1083 – 1090.
- [57] Rockwell Software. www.arenasimulation.com; 2008
- [58] Ma J, Fan ZP, Huang LH. A subjective and objective integrated approach to determine attribute weights. *European Journal of Operational Research* 1999; 112(2):397-404.
- [59] Hwang CL, Yoon K. Multiple attribute decision making: Methods and applications. *Tech. Rep., Lecture Notes in Economics and Mathematical Systems* 1981.
- [60] Jahanshahloo GR, Lofti FH, Izadikhah M. An algorithmic method to extend TOPSIS for decision-making problems with interval data. *Applied mathematics and computation* 2006; 175(2):1375-1384.

APPENDIX I. DATA DICTIONARY

The data dictionary acts as a document that can be used to assure that the elements of the data are correctly understood. It helps` to keep the data consistent as developers can use the data dictionary when adjusting the information system, which will support clean data.

Table		Name	Type	Size
Customers		Cust_Id	Text	255
		Cust_Name	Text	255
Machine		Mach_ID	Long Integer	4
		Mach_name	Text	255
Materials		Mat_Id	Long Integer	4
		Mat_Name	Text	255
		Stock_Level	Long Integer	4
Ops		Ops_No	Long Integer	4
		Ops_Name	Text	50
		Setup_time	Double	8
		Insp_time	Double	8
		Prod_Time	Double	8
		Start_DateTime	Date/Time	8
		End_DateTime	Date/Time	8
	CSetup_time	Double	8	














Table	Name	Type	Size
Ops_Machine	CInsp_time	Double	8
	CProd_time	Double	8
	 Status_ID_FK	Long Integer	4
	 Ops_No	Long Integer	4
	 Mach_ID_FK	Long Integer	4
Order_Parts	Enforce	Yes/No	1
	 Job_No_FK	Long Integer	4
Orders	 Part_ID_FK	Long Integer	4
	 Job_No	Long Integer	4
Parts	 Order_No	Text	50
	Cust_ID	Text	50
	RecDate	Date/Time	8
	ApprovedDate	Date/Time	8
	PromDate	Date/Time	8
	OrderStatus	Text	50
	 Part_ID	Long Integer	4
	Part_Name	Text	50
	Qty	Long Integer	4
	Drawing_No	Text	50
NumOps	Long Integer	4	
Parts_Materials	 Part_ID_FK	Long Integer	4

Table	Name	Type	Size
Parts_Ops	 Mat_Id_FK	Long Integer	4
	Mat_Qty	Text	255
	 Part_ID	Long Integer	4
	 Ops_No	Long Integer	4
Status	 Status_Id	Long Integer	4
	OpsStatus	Text	50

APPENDIX II. INFORMATION SYSTEM ASP CODE

This appendix contains the code that determines the functionality of the information system. The code was implemented using the MS FrontPage software programme. The information system is web-based, meaning that the information system can be updated from several different stations and not just from one computer station.

Normal *html* (Hypertext Markup Language) code, which is generally used as programming language to create documents for the World Wide Web, was developed to design the static homepage. Static means that the page does not change unless deliberately edited.

As it is a web page that configures and updates an information system, dynamic pages are needed. *ASP* (Active Server Pages) code is used for the dynamic functions like displaying current information system data or updating the information system. The *ASP* code in the next few pages of code is distinguished by the characters "<%>" and "%>", all the code between these two directives are *ASP* code and implements a dynamic function.

Add new Customer

```
<html>
<!--#include file=ADOVBS.INC -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<script language="JavaScript">
<!--
function FP_preloadImgs() { //v1.0
var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_swapImg() { //v1.0
var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_getObjectByID(id,o) { //v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[m]); if(el) return el; } }
return null;
}
// -->
</script>
</head>
<%
set conn = server.CreateObject("ADODB.Connection")
```

```

Conn.open                "provider=Microsoft.Jet.OLEDB.4.0;                Data
Source=C:\Users\David\Documents\2007\tesis
\daliff\daliff.mdb"
%>
<body onload="FP_preloadImgs (/url*/'button2E.jpg', /url*/'button2F.jpg')">
<p align="center"></p>
<p align="center">&nbsp;</p>
<p align="center"><font face="Andy" size="7">Customers added</font></p>
<%
Cust_ID = request.form("txtAcct_No")
Cust_Name = request.form("txtCust_Name")
Set RS = Server.CreateObject("ADODB.Recordset")
RS.Open "Customers", Conn, adOpenKeySet, adLockOptimistic, adCmdTable
RS.AddNew
RS.Fields("Cust_ID") = Cust_ID
RS.Fields("Cust_Name") = Cust_Name
RS.Update
Response.write "Customer account created as follows:" & "<br>"
Response.write Cust_ID & "&nbsp;&nbsp;&nbsp;" & Cust_Name & "<br>"
RS.close
Conn.close
SET RS = NOTHING
SET conn = NOTHING
%>&nbsp;  
</p>
<p align="center">&nbsp;</p>
<p align="center">

```

Generate Quote 1

```

<html>
<!-- #include file=connexion.inc -->
<!-- #include file=openrecordset.inc -->
<!-- #include file=makedropbox.inc -->
<head>
<meta http-equiv="Content-Language" content="en-za">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Daliff</title>
<script language="JavaScript">
<!--
function FP_swapImg() { //v1.0
var doc=document, args=arguments, elm, n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() { //v1.0
var d=document, a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_getObjectByID(id,o) { //v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[m]); if(el) return el; } }
return null;
}
// -->
</script>

```



```
SET RS_Customers = Nothing
SET Conn = Nothing
%>
</body>
</html>
```

Generat Quote 2

```
<html>
<!-- #include file=openrecordset.inc -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
</head>
<body>
<p align="center"></p>
<p align="center">&nbsp;</p>
<p align="center"><font face="Andy" size="7">Assign Part Information</font></p>
<form method="POST" action="Add%20material.asp" style="float: right; width:123px; "
name="frmAddNewMat">
<input type="submit" value="Add New Material to materials list" name="B4"></p>
</form>
</p>
<%
'Create connection object:
'-----
SET Conn = Server.CreateObject("ADODB.Connection")
Conn.Open "Provider=Microsoft.Jet.OLEDB.4.0; Data
Source=C:\Users\David\Documents\2007\tesis
\daliff\daliff.mdb"
call Openrecordset(Conn, RS_Customers, "Customers")
call Openrecordset(Conn, RS_Orders, "Orders")
call Openrecordset(Conn, Parts, "Parts")
call Openrecordset(Conn, Order_Parts, "Order_Parts")
call Openrecordset(Conn, Parts_Materials, "Parts_Materials")
'Set reference to a table named Materials:
'-----
SET MyGoodie = Conn.Execute("Materials", ,adCmdTable)
'Initialize array, 2 rows, init. 1 col. The first row will keep the Mat_ID,
'and the second the Mat_Name.
ReDim MaterialsIDList(2,1)
MaterialsIDList(1,1) = -1
ID_Index = 1 'For easy reference, e.g. MaterialsIDList(ID_Index, Name_Index)
Name_Index = 2
TempVar = "" 'String var to return multiple selection identifiers.
PartNum = session("PartNum")
Counter = session("Counter")
Dim d_today
d_today=Date
'Add order to database if first time entering page
'-----
If (Counter = 1) and (session("Matadded") <> 1) then
OddNumber = 1
session("oddnumber") = 1
'Get Acct no and name
'-----
Rs_customers.MoveFirst
if request.form("cmbACCT") <> "" Then
do while not Rs_customers.eof
if rs_customers.fields("cust_id")=request.form("cmbACCT") then
ACCT_No = rs_customers.fields("cust_id")
ACCT_Name = rs_customers.fields("Cust_Name")
```



```

exit do
end if
rs_customers.movenext
loop
end if
Ord_No = request.Form("Ord_No")
NumParts = request.Form("txtNumParts")
'Update Orders
'-----
RS_Orders.AddNew
RS_Orders.Fields("Order_no") = Ord_No
RS_Orders.Fields("Cust_ID") = Acct_No
RS_Orders.Fields("RecDate") = d_today
RS_Orders.Fields("OrderStatus") = "Quote"
RS_Orders.Update
'Get Job_No - the last record of the recordset
'-----
RS_Orders.MoveFirst
DO WHILE NOT RS_Orders.EOF
Job_No = RS_Orders.Fields("Job_No")
RS_Orders.MOVENEXT
LOOP
else
'Order allready added to database, add information
'-----
OddNumber = session("OddNumber")
NumParts = session("NumParts")
Job_No = session("Job_No")
ACCT_No = session("ACCT_No")
ACCT_Name = session("ACCT_Name")
Ord_No = session("Ord_No")
'Add Material and qty's to Parts_Material, Parts and Order_Parts tables
'-----
if (Counter = OddNumber) and (session("MatAdded") <> 1) then
Part_name = request.form("hdnPart_Name")
NumOps = request.form("hdnOps_Qty")
Part_Qty = request.form("hdnPart_Qty")
Part_drawing = request.form("hdnPart_drawing")
'Update Parts
'-----
Parts.AddNew
Parts.Fields("Part_Name") = Part_Name
Parts.Fields("Qty") = clng(Part_Qty)
Parts.Fields("Drawing_No") = Part_Drawing
Parts.Fields("NumOps") = clng(NumOps)
Parts.Update
'Get Part_ID
'-----
Parts.MoveFirst
DO WHILE NOT Parts.EOF
Part_ID = Parts.Fields("Part_ID")
Parts.MOVENEXT
LOOP
'Update Order_Parts
'-----
Order_Parts.AddNew
Order_Parts.Fields("Job_No") = clng(Job_No)
Order_Parts.Fields("Part_ID") = clng(Part_ID)
Order_Parts.Update
for i = 1 to request.form("txtMat_ID").count
Material_qty = request.form("txtMat_Qty")(i)
Material_ID = request.form("txtMat_ID")(i)
'Update Parts_Materials
'-----
Parts_Materials.AddNew

```



```

'Start to fill the list box with entries
'(we assumed there are rec's in the table):
'-----
Do While NOT(MyGoodie.EOF)
'Mechanically write the HTML code as responses; note how Quote is used:
'-----
Response.Write "<option value=" & Quote & MyGoodie.Fields("Mat_ID") & Quote
Response.Write ">" & MyGoodie.Fields("Mat_Name") & "</option>"
MyGoodie.MoveNext
Loop
%>
</select>
<input type="submit" value="Choose Material" name="B1" style="float: left">
<%
else
'Display part info section and select qty of materials
'-----
If Request.Form("slctListMaterials") <> "" then
Part_Name = Request.Form("txtPart_Name")
Part_Qty = Request.Form("txtPart_Qty")
%>
<div align="right">
<%Response.write "Added material for part: "%>
<input type="hidden" name="hdnPart_Name" value = "<%=Part_Name%>">
<input type="hidden" name="hdnPart_Qty" value = "<%=Part_Qty%>">
<input type="hidden" name="hdnPart_Drawing" value = "<%
=Request.Form("txtPart_Drawing")%>">
<input type="hidden" name="hdnOps_Qty" value = "<%=Request.Form("txtOps_Qty")%>">
<br>
<table border="2" cellspacing="2" width="50%" bordercolordark="#333333"
bordercolorlight="#666666">
<td><u>Material ID:</u></td>
<td><u>Material Name:</u></td>
<td><u>Qty per part:</u></td>
<%
'The multi-selection box returns a string with the values of all selected
'identifiers. These are separated by spaces & commas, e.g. "1, 2, 7, 9"
'We are extracting each significant number from the string and ignore
'the commas & spaces.
'Get the string:
TempVar = Trim(Request.Form("slctListMaterials")) ' "Trim" takes away spaces on
left & right of str
'Make sure something is selected, the first entry in the box is meaningless,
'and I have assigned its identifier the value "-1".
If TempVar <> CStr(-1) Then
i = 0
Do While (Len(TempVar) > 0)
'Find the first space in the string:
SpacePos = Instr(TempVar, " ")
'If there is a space, then remove everything 2 places
'(space + comma) to its left:
If SpacePos > 0 then
Current = Left(TempVar, SpacePos - 2)
'"Chop off" the removed digit(s), and the comma & space:
TempVar = Right(TempVar, Len(TempVar) - SpacePos)
Else
'We are at the last meaningful entry in the string, but it has no
'commas or spaces:
Current = trim(TempVar)
TempVar = "" 'Make it empty to terminate outer Loop
End if
'If the user (also) selected the first entry, i.e. the artificial
'entry, don't store it etc:
If Current <> "-1" Then
i = i + 1

```



```

<input type="submit" value="Submit" name="B2" ><input type="reset" value="Reset"
name="B3"></p>
</form>
<%end if
if session("Job_No") = "" then%>
<p align="center"><a href="Daliff.htm">
</a></p>
<%end if%>
</body>
</html>

```

Add Material 2

```

<html>
<script language="JavaScript">
<!--
function FP_swapImg() { //v1.0
var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() { //v1.0
var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_getObjectByID(id,o) { //v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[m]); if(el) return el; } }
return null;
}
// -->
</script>
<script language="JavaScript">
<!--
function FP_swapImg() { //v1.0
var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() { //v1.0
var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_getObjectByID(id,o) { //v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }

```



```

<td align="center">
<%response.write Mat_ID%>
</td>
<td >
<p align="center">
<%response.write Mat_name%>
</td>
<td>
<p align="center">
<%response.write Mat_Qty%>
</td>
</tr>
<%
next
%>
</table>
<p align="center">&nbsp;</p>
<% if session("Job_No") = "" then %>
<p align="center"><a href="Daliff.htm">
</a></p>
<% else
session("MatAdded") = 1
%>
<p><a href="Bom.asp">
</a></p>
<%end if
Materials.close
Conn.Close
SET Materials = Nothing
SET Conn = Nothing
%>
</body>
</html>

```

Update status

```

<html>
<!-- #include file=connexion.inc -->
<!-- #include file=openrecordset.inc -->
<!-- #include file=makedropbox.inc -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
<script language="JavaScript">
<!--
function FP_swapImg() {/v1.0

```

```

var doc=document, args=arguments, elm, n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() { //v1.0
var d=document, a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_getObjectByID(id, o) { //v1.0
var c, el, els, f, m, n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id, c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id, els[n]); if(el) return el; } }
return null;
}
// -->
</script>
</head>
<SCRIPT LANGUAGE="JavaScript" SRC="calendar.js"></SCRIPT>
<%
call Connect(Conn, "C:\Users\David\Documents\2007\thesis\daliff\daliff.mdb")
set Orders = conn.Execute("Orders")
set Customers = conn.Execute("Customers")
call Openrecordset(Conn, RS_Orders, "Orders")
Dim d_today
d_today=Date
%>
<body onload="FP_preloadImgs(/*url*/'button6.jpg', /*url*/'button5.jpg')">
<p align="center"></p>
<p align="center">&nbsp;</p>
<p align="center"><font face="Andy" size="7">Update Quote to an Order</font></p>
<%
MakeQuoteTable = "SELECT Orders.Job_No, Orders.Order_No, Orders.Cust_ID,
Orders.OrderStatus,
Customers.Cust_Name "
& "FROM Customers INNER JOIN Orders ON Customers.[Cust_Id] = Orders.[Cust_ID]
"
& "WHERE (((Orders.OrderStatus)='Quote'))"
set RS_Status = Conn.Execute(MakeQuoteTable)
if (Orders.BOF and Orders.EOF) then
Response.write "No Orders!"
else
Orders.MoveFirst
if request.form("cmbStatus") <> "" Then
do while not RS_Orders.eof
if RS_Orders.fields("Job_No") = clng(request.form("cmbStatus")) then
Set Session("Orders") = RS_Orders
exit do
end if
RS_Orders.movenext
loop
set orders = session("Orders")
Job_No = clng(Orders.fields("Job_No"))
Order_No = Orders.fields("Order_No")
ApprovedDate = cdate(request.form("AccpDate"))
PromDate = cdate(request.form("PromDate"))
SQLStmt = "UPDATE Orders SET Orders.[OrderStatus] = 'Order', Orders.ApprovedDate =
'" &
ApprovedDate & "', Orders.PromDate = '" & PromDate & "' "

```


Show Orders

```

<html>
<!-- #include file=connexion.inc -->
<!-- #include file=openrecordset.inc -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
<script language="JavaScript">
<!--
function FP_swapImg() {/v1.0
var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() {/v1.0
var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_getObjectByID(id,o) {/v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[n]); if(el) return el; } }
return null;
}
// -->
</script>
</head>
<body onload="FP_preloadImgs( /*url*/'button6.jpg', /*url*/'button5.jpg')">
<p align="center"></p>
<p align="center">&nbsp;</p>
<p align="center"><font face="Andy" size="7">Orders Log</font></p>
<p> <%
call Connect(Conn, "C:\Users\David\Documents\2007\thesis\daliff\daliff.mdb")
set Orders = conn.Execute("Orders")
set Customers = conn.Execute("Customers")
set Parts = conn.Execute("Parts")
set Order_Parts = conn.Execute("Order_Parts")
SQLStmnt = "SELECT Order_Parts.Job_No, Orders.Order_No, Orders.Cust_ID,
Customers.Cust_Name,
Orders.RecDate, "_
& "Orders.ApprovedDate, Orders.PromDate, Order_Parts.Part_ID AS
Order_Parts_Part_ID,
Parts.Part_Name, "_
& "Parts.Drawing_No, Parts.Qty FROM Customers INNER JOIN (Parts INNER JOIN (Orders
INNER JOIN Order_Parts "_
& "ON Orders.[Job_No] = Order_Parts.[Job_No]) ON Parts.[Part_ID] =
Order_Parts.[Part_ID]) ON Customers.Cust_Id = Orders.Cust_ID "_
& "WHERE (((Orders.OrderStatus)='Order'))"
Set RS = conn.Execute(SQLStmnt)
if (RS.BOF AND RS.EOF) then
%>
</p>
<p align="center">No Records of any Orders Found!</p>
<%
else
RS.Movefirst

```

```

%>
<table border="2" cellspacing="1" Cols="<%=RS.Fields.Count %>">
<tr style="border: 1px solid #000000; padding-left: 4px; padding-right: 4px;
padding-top:
1px; padding-bottom: 1px">
<td width="3%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold" >Job No
</td>
<td width="9%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Order No
</td>
<td width="4%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Cust ID
</td>
<td width="20%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Cust Name
</td>
<td width="7%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Received
</td>
<td width="7%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Approved
</td>
<td width="7%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Promised
</td>
<td width="9%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Part ID
</td>
<td width="18%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Part Name
</td>
<td width="9%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Drawing No
</td>
<td width="5%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Quantity
</td>
</tr>
<%
Do While NOT RS.EOF%>
<tr>
<td width="3%" align=center><%=RS.Fields("Job_No") %>
</td>
<td width="9%" align=center><%=RS.Fields("Order_No") %>
</td>
<td width="4%" align=center><%=RS.Fields("Cust_ID") %>
</td>
<td width="20%" align=center><%=RS.Fields("Cust_Name") %>
</td>
<td width="7%" align=center><%=RS.Fields("RecDate") %>
</td>
<td width="7%" align=center><%=RS.Fields("ApprovedDate") %>
</td>

```

```

<td width="7%" align=center><%=RS.Fields("PromDate")%>
</td>
<td width="9%" align=center><%=RS.Fields("Order_Parts_Part_ID")%>
</td>
<td width="18%" align=center><%=RS.Fields("Part_Name")%>
</td>
<td width="9%" align=center><%=RS.Fields("Drawing_No")%>
</td>
<td width="5%" align=center><%=RS.Fields("Qty")%>
</td>
</tr>
<%
RS.MoveNext
Loop
End if
RS.close
orders.close
customers.close
parts.close
order_parts.close
Conn.close
set RS = nothing
set Orders = nothing
set Customers = nothing
set Parts = nothing
set Order_Parts = nothing
set conn = nothing
%>
</table>
<p align="center"><a href="Daliff.htm">
</a></p>
</body>
</html>

```

Show Quotes

```

<html>
<!-- #include file=connexion.inc -->
<!-- #include file=openrecordset.inc -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
<script language="JavaScript">
<!--
function FP_swapImg() { //v1.0
var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array(); for(n=2; n<args.length; n+=2) { elm=FP_getObjectByID(args[n]); if(elm) { doc.$imgSwaps[doc.$imgSwaps.length]=elm; elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() { //v1.0
var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}

```

```

}
function FP_getObjectByID(id,o) { //v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[n]); if(el) return el; } }
return null;
}
// -->
</script>
</head>
<body onload="FP_preloadImgs (/ *url* / 'button6.jpg', / *url* / 'button5.jpg') ">
<p align="center"></p>
<p align="center">&nbsp;</p>
<p align="center"><font face="Andy" size="7">Quotes Log</font></p>
<%
call Connect (Conn, "C:\Users\David\Documents\2007\thesis\daliff\daliff.mdb")
set Orders = conn.Execute("Orders")
set Customers = conn.Execute("Customers")
set Parts = conn.Execute("Parts")
set Order_Parts = conn.Execute("Order_Parts")
SQLStmnt = "SELECT Order_Parts.Job_No, Orders.Order_No, Orders.Cust_ID,
Customers.Cust_Name,
Orders.RecDate, "_
& "Orders.ApprovedDate, Orders.PromDate, Order_Parts.Part_ID AS
Order_Parts_Part_ID,
Parts.Part_Name, "_
& "Parts.Drawing_No, Parts.Qty FROM Customers INNER JOIN (Parts INNER JOIN (Orders
INNER JOIN Order_Parts "_
& "ON Orders.[Job_No] = Order_Parts.[Job_No]) ON Parts.[Part_ID] =
Order_Parts.[Part_ID]) ON Customers.Cust_Id = Orders.Cust_ID "_
& "WHERE (((Orders.OrderStatus)='Quote'))"
Set RS = conn.Execute(SQLStmnt)
if (RS.BOF AND RS.EOF) then
%>
<p align="center">No Records of any Quotes Found!</p>
<%
else
RS.Movefirst
%>
<table border="2" cellspacing="1" Cols="<% =RS.Fields.Count %>">
<tr style="border: 1px solid #000000; padding-left: 4px; padding-right: 4px;
padding-top:
1px; padding-bottom: 1px">
<td width="3%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold" >Job No
</td>
<td width="9%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Order No
</td>
<td width="4%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Cust ID
</td>
<td width="20%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Cust Name
</td>

```

```

<td width="7%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Received
</td>
<td width="9%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Part ID
</td>
<td width="18%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Part Name
</td>
<td width="9%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Drawing No
</td>
<td width="5%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Quantity
</td>
</tr>
<%
Do While NOT RS.EOF%>
<tr>
<td width="3%" align=center><%=RS.Fields("Job_No")%>
</td>
<td width="9%" align=center><%=RS.Fields("Order_No")%>
</td>
<td width="4%" align=center><%=RS.Fields("Cust_ID")%>
</td>
<td width="20%" align=center><%=RS.Fields("Cust_Name")%>
</td>
<td width="7%" align=center><%=RS.Fields("RecDate")%>
</td>
<td width="9%" align=center><%=RS.Fields("Order_Parts_Part_ID")%>
</td>
<td width="18%" align=center><%=RS.Fields("Part_Name")%>
</td>
<td width="9%" align=center><%=RS.Fields("Drawing_No")%>
</td>
<td width="5%" align=center><%=RS.Fields("Qty")%>
</td>
</tr>
<%
RS.MoveNext
Loop
End if
RS.close
orders.close
customers.close
parts.close
order_parts.close
Conn.close
set RS = nothing
set Orders = nothing
set Customers = nothing
set Parts = nothing
set Order_Parts = nothing
set conn = nothing
%>
</table>
<p align="center"><a href="Daliff.htm">
</a></p>
</body>
</html>

```

Show Operations

```

<html>
<!-- #include file=connexion.inc -->
<!-- #include file=openrecordset.inc -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>New Page 1</title>
<script language="JavaScript">
<!--
function FP_swapImg() { //v1.0
var doc=document,args=arguments,elm,n; doc.$imgSwaps=new Array(); for(n=2;
n<args.length;
n+=2) { elm=FP_getObjectByID(args[n]); if(elm) {
doc.$imgSwaps[doc.$imgSwaps.length]=elm;
elm.$src=elm.src; elm.src=args[n+1]; } }
}
function FP_preloadImgs() { //v1.0
var d=document,a=arguments; if(!d.FP_imgs) d.FP_imgs=new Array();
for(var i=0; i<a.length; i++) { d.FP_imgs[i]=new Image; d.FP_imgs[i].src=a[i]; }
}
function FP_getObjectByID(id,o) { //v1.0
var c,el,els,f,m,n; if(!o)o=document; if(o.getElementById) el=o.getElementById(id);
else if(o.layers) c=o.layers; else if(o.all) el=o.all[id]; if(el) return el;
if(o.id==id || o.name==id) return o; if(o.childNodes) c=o.childNodes; if(c)
for(n=0; n<c.length; n++) { el=FP_getObjectByID(id,c[n]); if(el) return el; }
f=o.forms; if(f) for(n=0; n<f.length; n++) { els=f[n].elements;
for(m=0; m<els.length; m++){ el=FP_getObjectByID(id,els[m]); if(el) return el; } }
return null;
}
// -->
</script>
</head>
<body onload="FP_preloadImgs( /*url*/'button6.jpg', /*url*/'button5.jpg')">
<p align="center"></p>
<p align="center">&nbsp;</p>
<p align="center"><font face="Andy" size="7">Ops Log of Orders</font></p>
<%
call Connect(Conn, "C:\Users\David\Documents\2007\tesis\daliff\daliff.mdb")
set Orders = conn.Execute("Orders")
set Customers = conn.Execute("Customers")
set Parts = conn.Execute("Parts")
set Order_Parts = conn.Execute("Order_Parts")
set Parts_Ops = conn.Execute("Parts_Ops")
set Ops = conn.Execute("Ops")
set Ops_Machine = conn.Execute("Ops_Machine")
set Machine = conn.Execute("Machine")
SQLOps = "SELECT Orders.Job_No, Order_Parts.Part_ID AS Order_Parts_Part_ID,
Parts.Part_Name,
Parts_Ops.Ops_No, Ops.Ops_Name, Ops.Est_Time, Ops_Machine.Mach_ID, "_
& "Machine.Mach_name, Ops_Machine.Enforce FROM (Parts INNER JOIN (Orders INNER JOIN
Order_Parts ON Orders.[Job_No] = Order_Parts.[Job_No]) "_

```

```

& "ON Parts.[Part_ID] = Order_Parts.[Part_ID]) INNER JOIN ((Ops INNER JOIN (Machine
INNER JOIN Ops_Machine ON Machine.Mach_ID = "
& "Ops_Machine.Mach_ID) ON Ops.Ops_No = Ops_Machine.Ops_No) INNER JOIN Parts_Ops ON
Ops.Ops_No = Parts_Ops.Ops_No) ON Parts.Part_ID "
& "= Parts_Ops.Part_ID WHERE (((Orders.OrderStatus)='Order')))"
Set RSAll = conn.Execute(SQLOps)
if (RSAll.BOF AND RSAll.EOF) then
%>
<p align="center">No Records of any Ops Found!</p>
<%
else
RSAll.Movefirst
%>
<table border="2" cellspacing="1" Cols="<% =RSAll.Fields.Count %>">
<tr style="border: 1px solid #000000; padding-left: 4px; padding-right: 4px;
padding-top:
1px; padding-bottom: 1px">
<td width="3%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold" >Job No
</td>
<td width="5%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Part Id
</td>
<td width="13%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Part Name
</td>
<td width="5%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Ops No
</td>
<td width="13%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Ops Name
</td>
<td width="13%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Ops Time
</td>
<td width="5%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Mach ID
</td>
<td width="13%" align=center bordercolor="#000000" style="font-size: 12pt;
fontweight:
bold">Mach Name
</td>
<td width="6%" align=center bordercolor="#000000" style="font-size: 12pt; font-
weight:
bold">Enforce?
</td>
</tr>
<%
Do While NOT RSAll.EOF%>
<tr>
<td width="3%" align=center><%=RSAll.Fields("Job_No") %>
</td>
<td width="9%" align=center><%=RSAll.Fields("Order_Parts_Part_ID") %>
</td>
<td width="4%" align=center><%=RSAll.Fields("Part_Name") %>
</td>
<td width="20%" align=center><%=RSAll.Fields("Ops_No") %>
</td>

```



```

<td width="7%" align=center><%=RSall.Fields("Ops_name")%>
</td>
<td width="9%" align=center><%=RSall.Fields("Est_Time")%>
</td>
<td width="9%" align=center><%=RSall.Fields("mach_ID")%>
</td>
<td width="18%" align=center><%=RSall.Fields("mach_name")%>
</td>
<td width="9%" align=center><%=RSall.Fields("Enforce")%>
</td>
</tr>
<%
RSall.MoveNext
Loop
End if
rsall.close
orders.close
customers.close
parts.close
order_parts.close
Parts_Ops.close
Ops.close
Ops_Machine.close
Machine.close
Conn.close
set rsall = nothing
set Parts_Ops = nothing
set Ops = nothing
set Ops_Machine = nothing
set Machine = nothing
set Orders = nothing
set Customers = nothing
set Parts = nothing
set Order_Parts = nothing
set conn = nothing
%>
</table>
<p align="center"><a href="Daliff.htm">
</a></p>
</body>
</html>

```

APPENDIX III. EXPLANATION OF SUBROUTINES IN VBA CODE

This appendix gives a non technical overview of the custom subroutines coded for the simulation model. The purpose of each subroutine, the other subroutines it calls, its inputs and its outputs are stated. The subroutines are called during the execution of the simulation model, and it customizes the simulation model according to the shop floor and order status. The actual code of the subroutines is included in the next appendix.

Sub startModel ()*Purpose:*

- Open database and record sets
- Clean temporary simulation record set and populate with current system state information
- Set entity count
- Set number of entities to create

Subs Called:

- setVariables
- createEntities
- AssignEntArray
- AssignEntSuccessor
- AssignDispatchrule
- DetermineNumOps

Input:

- Database

Output:

- Temporary simulation record set
- Entity count

Sub AssignTotalEntCount ()*Purpose:*

- Set total entity count variable in Arena

Subs Called:

- None

Input:

- TotalEntCount variable in VBA

Output:

- Arena variable TotalEntCount

Sub UpdateSimTableFromTempTable ()

Purpose:

- Update Simulation query record set

Subs Called:

- None

Input:

- Temporary simulation record set

Output:

- Simulation query record set

Sub setVariables ()

Purpose:

- Reset all the variables used in the VBA code

Subs Called:

- None

Input:

- None

Output:

- All VBA variables set to initial values

Sub AssignDispatchRule ()

Purpose:

- Set the queue disciplines according to the chosen scheduling rule

Subs Called:

- None

Input:

- Dispatch variable that represents the chosen scheduling rule

Output:

- Queue disciplines

Sub createEntities ()

Purpose:

- Create entities and assign names to them

Subs Called:

- None

Input:

- Simulation query record set

Output:

- Entities with names

Sub AssignEntArray (Origin As Integer)

Purpose:

- Populate the entity array for each entity with its attribute values

Subs Called:

- None

Input:

- Simulation query record set
- TotalEntCount variable
- Origin variable

Output:

- EntityRecord array
- EntityDateRecord array

Sub AssignEntSuccessor ()

Purpose:

- Assign entity successor if one exists

Subs Called:

- None

Input:

- Simulation query record set
- EntityRecord array
- Simulation query record set

Output:

- Updated EntityRecord array

Sub CleanEntities ()

Purpose:

- Delete entities created from *Entity* spreadsheet

Subs Called:

- None

Input:

- EntityRecord array
- TotalEntCount variable
- Module data

Output:

- Clean *Entity* spreadsheet

Sub AssignEntAttr ()

Purpose:

- Assign attributes to an entity

Subs Called:

- None

Input:

- EntityRecord array
- Simulation run data
- TotalEntCount variable

Output:

- Entity attribute values

Sub DetermineNumOps ()

Purpose:

- Determine the number of operations each part has

Subs Called:

- DetermineDueDate
- DetermineStartTime

Input:

- EntityRecord array
- TotalEntCount variable

Output:

- Entity attribute values

Sub DetermineDueDate ()

Purpose:

- Determine the due date of the entities

Subs Called:

- None

Input:

- EntityRecord array
- EntityDateRecord array

Output:

- Updated EntityRecord array

Sub DetermineStartTime (Origin As Integer)

Purpose:

- Determine the earliest possible start time of entities
- Determine entity slack

Subs Called:

- None

Input:

- EntityRecord array
- Origin variable

Output:

- Updated EntityRecord array

Sub UpdateStartTimes ()

Purpose:

- Update the earliest possible start time of entities

Subs Called:

- DetermineStartTime

Input:

- EntityRecord array

Output:

- Updated EntityRecord array

Sub UpdateSimOpsTable (Ops_No As Integer, Column As String, NewValue As Double)

Purpose:

- Writes information to the temporary simulation record set

Subs Called:

- None

Input:

- EntityRecord array
- Temporary simulation record set

Output:

- Updated temporary simulation record set

Sub updateCompleteProgress ()

Purpose:

- Update system as processing of the entity finishes

Subs Called:

- UpdateSimOpsTable
- UpdateStartTimes
- UpdateSuccessor
- DetermineCR

Input:

- EntityRecord array
- Simulation run data

Output:

- Updated Entity attributes
- Updated EntityRecord

Sub UpdateSnapShotAtEndRep ()

Purpose:

- Records current system state when replication run ends

Subs Called:

- UpdateSimOpsTable

Input:

- EntityRecord array
- Simulation run data
- Temporary simulation record set

Output:

- Updated temporary simulation record set

Sub UpdateSuccessor ()

Purpose:

- Updates the successor entity attributes and array

Subs Called:

- Reshuffle

Input:

- EntityRecord array
- Simulation run data
- TotalEntCount

Output:

- Updated entity attributes
- Updated EntityRecord array

Sub DetermineCR ()

Purpose:

- Determine the critical ratio of the entity

Subs Called:

- None

Input:

- EntityRecord array
- Simulation run data

Output:

- Updated EntityRecord array

Sub Reshuffle (qNum As Integer)

Purpose:

- Reshuffles the queue according to the chosen dispatching rule

Subs Called:

- None

Input:

- qNum variable
- Simulation run data

Output:

- Reshuffled queue

Sub DetermineLateness ()

Purpose:

- Determine the lateness of the part

Subs Called:

- None

Input:

- Simulation run data
- Arena variable varTotLateness

Output:

- Lateness value

Sub DetermineEarliness ()

Purpose:

- Determine the earliness of the part

Subs Called:

- None

Input:

- Simulation run data
- Arena variable varTotEarliness

Output:

- Earliness value

Sub UpdateAttrTimesforBusyEnt ()

Purpose:

- Set attribute values of entity that has a busy state at the beginning of the simulation run

Subs Called:

- None

Input:

- Simulation run data
- EntityRecord array

Output:

- Entity attributes

Sub VBA_Block_19_Fire ()

Purpose:

- Write part statistics to Excel results file

Subs Called:

- DetermineLateness
- DetermineEarliness

Input:

- Simulation run data

Output:

- Excel results file

APPENDIX IV. SIMULATION MODEL CODE

The VBA code used to customize the simulation model for each scheduling scenario is included in this appendix. The code can be followed using the previous appendix that states the purpose of every subroutine, the chapter about the simulation model will also help the reader understand the code. Comments are included in the code that acted as guidance for the designer, it is distinguished by the character ` and a different colour of text.

```
Dim oModel As Model, osiman As SIMAN
Dim j, k, TotalEntCount, Counter, CellCount, PartIndex, Dispatch, DDSlack,
    Slack, nReps As Integer
Dim CleanCount, EntityCount, temp, tempPart, HoursToDD, EntDD As Integer
Dim dateProm, dateStart As Integer
Dim CR(200) As Double
Dim EntityRecord(200, 21)
Dim entityDaterecord(200, 2) As Date
Dim Inactive, Active, Complete, Busy, HoursPerWorkDay As Integer
Dim EntLoc, EntName, EntPart, EntSuc, EntProg, EntAddr, EntTotTime, EntST,
    EntCR, EntMachName As Integer
Dim EntTime, EntSet, EntInsp, PartSize, EntType, EntStart, EntEnd, EntSlack,
    EntPredecessor, entTotMakespan As Integer
Dim attrPartsize, attrEnt_Name, attrStartTime, attrSlack, attrLocation,
    attrPart_ID, attrTime, attrTotTime As Integer
Dim attrEnt_Successor, attrDD, attrSetup, attrCompletionTime, attrProgress,
    attrCR, attrInspection As Integer
Dim attrCSetup_Time, attrCProd_Time, attrCInsp_Time, attrCSetup_Process,
    attrCInsp_Process, attrCProd_Process As Double
Dim attrInsp_Start, attrProd_Start, attrSetup_Start, attrTotMakespan As
    Double
Dim HoursInDays, DaysInMinutes As Double
Dim x, y, z, q As Long
Dim txtSeqName, dispatchrule(10) As String
Dim TheDb As DAO.Database 'Ref to database
Dim WS As DAO.Workspace 'Ref to Direct Access Object
Dim RS, TempRS As Recordset 'The query in the db
Dim CurrentPartID As Long
```

Private Sub startModel()

```
Set oModel = ThisDocument.Model 'Init ref to this Arena model
Set osiman = oModel.SIMAN 'Ref to its Siman obj.
Set WS = DBEngine(0) 'Create the DAO workspace
Set TheDb = WS.OpenDatabase(Model.Path & "daliff.mdb") 'Open our db in the
workspace
Set RS = TheDb.OpenRecordset("SimOps") 'Execute the query
Set TempRS = TheDb.OpenRecordset("TempSimOps")

'clear the temp table of previous records
If TempRS.EOF And TempRS.BOF Then
    'no records in temp table
Else
```

```

TempRS.MoveFirst
TempRS.Edit
Do While Not TempRS.EOF
    TempRS.Delete
    TempRS.MoveNext
Loop
End If

setVariables

'In case query is empty:
If RS.EOF And RS.BOF Then Exit Sub ' Nothing to seq

'Right, let's begin:
RS.MoveFirst

'Run through the query. Each time the Part ID changes, we need a new
sequence:
Do While Not RS.EOF

    'write current state of simulation to temp table
    TempRS.AddNew

    For i = 0 To 16
        TempRS.Fields(i).value = RS.Fields(i).value
    Next i

    'start and end dates?
    For i = 17 To 18
        TempRS.Fields(i).value = 0
    Next i
    TempRS.Update

    CurrentPartID = RS.Fields("Part_ID")

    createEntities
    AssignEntArray (0)

    TotalEntCount = TotalEntCount + 1

    RS.MoveNext

    AssignEntSuccessor

    If RS.EOF Then Exit Do

Loop    'Do for the whole query

'create    ammount    of    entities    for    this    run
oModel.Modules(oModel.Modules.Find(smFindTag, "object.80")).Data("Max_
Batches") = TotalEntCount - 1

    AssignDispatchrule
    DetermineNumOps

End Sub

```

```

Private Sub ModelLogic_RunEndReplication ()

    UpdateSnapShotAtEndRep

    'make MS Project schedule

    'For i = 1 To TotalEntCount - 1
    '    MSPProject.Application.ActiveProject.Tasks.Add ("Ops_" &
    EntityRecord(i, EntName))
    '    HoursInDays = EntityRecord(i, EntStart) / HoursPerWorkDay
    '    DaysInMinutes = (Fix(HoursInDays) * 24 * 60) + ((HoursInDays -
    Fix(HoursInDays)) * 9 * 60)
    '    MSPProject.Application.ActiveProject.Tasks(i).Predecessors =
    EntityRecord(i, EntPredecessor)
    '    MSPProject.Application.ActiveProject.Tasks(i).Start = DateAdd("n",
    DaysInMinutes, MSPProject.Application.ActiveProject.ProjectStart)
    '    MSPProject.Application.ActiveProject.Tasks(i).Duration =
    (EntityRecord(i, EntEnd) - EntityRecord(i, EntStart)) & "h"
    '    MSPProject.Application.ActiveProject.Tasks(i).ResourceNames =
    EntityRecord(i, EntMachName)
    'Next i

End Sub

Private Sub ModelLogic_RunBegin ()

    frmDispRule.Show
    startModel

End Sub

Private Sub ModelLogic_RunBeginReplication ()

    DetermineCR
    AssignTotalEntCount
    TotalEntCount = 1
    Counter = 1

    'In case query is empty:
    If RS.EOF And RS.BOF Then Exit Sub ' Nothing to seq

    'Right, let's begin:
    RS.MoveFirst

    'Run through the query. Each time the Part ID changes, we need a new
    sequence:
    Do While Not RS.EOF

        AssignEntArray (1)
        TotalEntCount = TotalEntCount + 1
        RS.MoveNext
        AssignEntSuccessor
        If RS.EOF Then Exit Do
    Loop

End Sub

```

```
Private Sub AssignTotalEntCount ()
Dim i As Integer
```

```
    i = osiman.SymbolNumber("TotalEntCount")
    osiman.VariableArrayValue(i) = TotalEntCount - 1
```

```
End Sub
```

```
Private Sub ModelLogic_RunBeginSimulation ()
```

```
    nReps = oModel.NumberOfReplications
    Excel.Application.Workbooks.Open (Model.Path & "results.xls")
    'MSProject.Application.FileOpen (Model.Path & "Schedule.mpp")
```

```
    'Clear schedule
    'j = MSProject.Application.ActiveProject.Tasks.Count
    'For i = 0 To j - 1
    '    MSProject.Application.ActiveProject.Tasks(j - i).Delete
    'Next i
```

```
    'clear excel ranges
```

```
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("A2:E2000").Clear
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("T2:X2000").Clear
```

```
End Sub
```

```
Private Sub ModelLogic_RunEndSimulation ()
```

```
    'update the simulation table from the temp simulation table
    UpdateSimTableFromTempTable
```

```
    'statistic outputs
```

```
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("Y2").value = oModel.SIMAN.OutputStatisticValue(1)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("Y3").value = oModel.SIMAN.OutputStatisticValue(2)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("Y4").value = oModel.SIMAN.OutputStatisticValue(3)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("Y5").value = oModel.SIMAN.OutputStatisticValue(8)
```

```
    'usage of each machine:
```

```
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK2").value = oModel.SIMAN.OutputStatisticValue(4)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK3").value = oModel.SIMAN.OutputStatisticValue(5)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK4").value = oModel.SIMAN.OutputStatisticValue(6)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK5").value = oModel.SIMAN.OutputStatisticValue(7)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK6").value = oModel.SIMAN.OutputStatisticValue(9)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK7").value = oModel.SIMAN.OutputStatisticValue(10)
Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
    ("AK8").value = oModel.SIMAN.OutputStatisticValue(11)
```

```

Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range_
("AK9").value = oModel.SIMAN.OutputStatisticValue(12)

k = 2
For j = 1 To 8
  For i = 1 To TotalEntCount - 1
    If EntityRecord(i, EntST) = j Then
      TempRS.MoveFirst
      Do While Not TempRS.EOF
        If TempRS.Fields("Ops_No") = EntityRecord(i, EntName) Then
          Excel.Application.Workbooks(1).Worksheets(Dispatch +
            1).Range("A" & k).value = TempRS.Fields("Part_ID")
          Excel.Application.Workbooks(1).Worksheets(Dispatch +
            1).Range("B" & k).value = TempRS.Fields("Ops_No")

'If then entity start time is smaller than zero, then the starting time
must be set to zero, that the remaining processing time is also shown on
the schedule

          If EntityRecord(i, EntStart) >= 0 Then
            Excel.Application.Workbooks(1).Worksheets(Dispatch_
              + 1).Range("C" & k).value =TempRS._
              Fields("Start_DateTime")
          Else
            Excel.Application.Workbooks(1).Worksheets(Dispatch_
              + 1).Range("C" & k).value = 0
          End If

'If the entity end time is smaller than zero, the ops has been completed
before this simulation run and must then be ignored
          If EntityRecord(i, EntEnd) >= 0 Then
            Excel.Application.Workbooks(1).Worksheets(Dispatch_
              + 1).Range("D" & k).value = TempRS._
              Fields("End_DateTime")
          Else
            Excel.Application.Workbooks(1).Worksheets(Dispatch_
              + 1).Range("D" & k).value = 0
          End If

          Excel.Application.Workbooks(1).Worksheets(Dispatch +
            1).Range("E" & k).value = TempRS.Fields("Mach_ID")

'Configure chart to give schedule type bar chart
          Excel.Application.Workbooks(1).Worksheets(Dispatch +
            1).Select
          Excel.Worksheets(Dispatch + 1).ChartObjects(1).Activate
          Excel.ActiveChart.SeriesCollection(2).DataLabels.Select
          Excel.ActiveChart.SeriesCollection(2).Points(k -
            1).DataLabel.Select

          If EntityRecord(i, EntEnd) < 0 Then
            Selection.Characters.Text = " "
          Else
            Selection.Characters.Text = "O_" & EntityRecord(i,
              EntName)
          End If

'give each ops its own colour
          Excel.Worksheets(Dispatch + 1).ChartObjects(1).Activate

```

```

Excel.ActiveChart.SeriesCollection(2).Points(k -_
1).Select

With Selection.Interior
    .ColorIndex = 5 * EntityRecord(i, EntPart) + 18
    .Pattern = xlSolid
End With

    k = k + 1
End If
TempRS.MoveNext
Loop
End If
Next i
Next j
End Sub

Private Sub UpdateSimTableFromTempTable ()

TempRS.MoveFirst
RS.MoveFirst
Do While Not TempRS.EOF

    RS.Edit
    For i = 13 To 16
        RS.Fields(i).value = TempRS.Fields(i).value
    Next i
    If RS.Fields("Status_Id").value >= Busy Then
        If RS.Fields("Start_DateTime").value > 0 Then

            Else
                RS.Fields("Start_DateTime").value = DateAdd("n", _
                TempRS.Fields("Start_DateTime").value * 60, _
                oModel.StartDateTime)
            End If

        If RS.Fields("Status_Id").value = Complete Then
            If RS.Fields("End_DateTime").value > 0 Then

                Else
                    RS.Fields("End_DateTime").value = DateAdd("n", _
                    TempRS.Fields("End_DateTime").value * 60, _
                    oModel.StartDateTime)
                End If
            End If
        End If
        RS.Update
        TempRS.MoveNext
        RS.MoveNext
    Loop

End Sub

Private Sub ModelLogic_RunEnd()

CleanEntities

Excel.Application.Workbooks.Close

```

```
Excel.Application.Quit

'MSProject.Application.FileCloseAll (pjSave)

'Go home:
RS.Close
TheDb.Close
WS.Close

Set RS = Nothing
Set TheDb = Nothing
Set WS = Nothing

Set osiman = Nothing
Set oModel = Nothing

End Sub

Private Sub setVariables()

    TotalEntCount = 1
    Counter = 1
    DDSlack = 0
    CleanCount = 1
    CellCount = 1

    'reset entity array
    For i = 1 To 200
        For j = 1 To 21
            If j = 19 Then
                EntityRecord(i, j) = ""
            Else
                EntityRecord(i, j) = -1
            End If
        Next j
    Next i

    'assign dispatch rule chosen by user
    If frmDispRule.FIFO.value = True Then
        Dispatch = 1
    End If
    If frmDispRule.LPT.value = True Then
        Dispatch = 2
    End If
    If frmDispRule.SPT.value = True Then
        Dispatch = 3
    End If
    If frmDispRule.EDD.value = True Then
        Dispatch = 4
        DDSlack = frmSlack.txtSlack.value
    End If
    If frmDispRule.EST.value = True Then
        Dispatch = 5
    End If
    If frmDispRule.SS.value = True Then
        Dispatch = 6
    End If
```



```
If frmDispRule.CR.value = True Then
    Dispatch = 7
End If

'set entity record attributes
EntLoc = 1
EntName = 2
EntPart = 3
EntSuc = 4
EntProg = 5
EntST = 6
EntTime = 7
EntSet = 8
EntInsp = 9
PartSize = 10
EntAddr = 11
EntType = 12
EntDD = 13
EntStart = 14
EntEnd = 15
EntSlack = 16
EntCR = 17
EntMachName = 18
EntPredecessor = 19
EntTotTime = 20
entTotMakespan = 21

'set entity date record attributes
dateProm = 1
dateStart = 2

'Set hours per work day
HoursPerWorkDay = CDb1(9.5)

'set progress attributes
Inactive = 0
Active = 1
Busy = 2
Complete = 3

'set attribute for dispatch rule
dispatchrule(1) = "attrProgress"
dispatchrule(2) = "attrTotTime"
dispatchrule(3) = "attrTotTime"
dispatchrule(4) = "attrDD"
dispatchrule(5) = "attrStartTime"
dispatchrule(6) = "attrSlack"
dispatchrule(7) = "attrCR"

'set entity attributes according to siman.txt file
attrPartsize = 1
attrEnt_Name = 2
attrStartTime = 3
attrSlack = 4
attrLocation = 5
attrCInsp_Time = 6
attrTotMakespan = 7
attrPart_ID = 8
```

```

attrTime = 9
attrCProd_Process = 10
attrProgress = 11
attrEnt_Successor = 12
attrCSetup_Time = 13
attrCInsp_Process = 14
attrProd_Start = 15
attrCompletionTime = 16
attrCR = 17
attrTotTime = 18
attrDD = 19
attrSetup = 20
attrCProd_Time = 21
attrSetup_Start = 22
attrInspection = 23
attrInsp_Start = 24
attrCSetup_Process = 25

```

End Sub

Private Sub AssignDispatchrule()

```
Dim strDispatchRule As String
```

```
'change attribute for dispatch rule of all queues
```

```

Model.Modules (Model.Modules.Find(smFindTag, _
"object.2584")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.2585")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.2586")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.2587")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.2588")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.2589")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.2590")).Data("Attribute") = dispatchrule(Dispatch)
Model.Modules (Model.Modules.Find(smFindTag, _
"object.23282")).Data("Attribute") = dispatchrule(Dispatch)

```

```
'change rule type
```

```

If Dispatch = 3 Or Dispatch = 4 Or Dispatch = 5 Or Dispatch = 6 Then
    'SPT, EDD, EST, SS
    strDispatchRule = "Lowest Attribute Value"
Else
    'FIFO, LPT
    strDispatchRule = "Highest Attribute Value"
End If

```

```

Model.Modules (Model.Modules.Find(smFindTag, _
"object.2584")).Data("Type") = strDispatchRule
Model.Modules (Model.Modules.Find(smFindTag, _ "object.2585")).Data("Type") =
strDispatchRule
Model.Modules (Model.Modules.Find(smFindTag, _ "object.2586")).Data("Type") =
strDispatchRule

```

```

Model.Modules (Model.Modules.Find(smFindTag, _ "object.2587")).Data("Type") =
strDispatchRule
Model.Modules (Model.Modules.Find(smFindTag, _ "object.2588")).Data("Type") =
strDispatchRule
Model.Modules (Model.Modules.Find(smFindTag, _ "object.2589")).Data("Type") =
strDispatchRule
Model.Modules (Model.Modules.Find(smFindTag, _ "object.2590")).Data("Type") =
strDispatchRule
Model.Modules (Model.Modules.Find(smFindTag, _ "object.23282")).Data("Type")
= strDispatchRule

```

End Sub

Private Sub createEntities()

```

'insert entities to entity module
oModel.Modules.Create "BasicProcess", "Entity", 0, 0
EntityCount = oModel.Modules.Count 'Module counter to assign name
oModel.Modules (EntityCount).Data("Name") = "Ops_" & RS.Fields("Ops_No")

```

End Sub

Private Sub AssignEntArray(Origin As Integer)

'ASSUMPTION: all orders to be delivered at 12:00

'Entity (Ops) Attributes saved in array

'if it is the first replication of the simulation, the address could be found - it is the same for all the reps once found

If Origin = 0 Then

'entityLocation

EntityRecord(TotalEntCount, EntAddr) = oModel.Shapes.Count

End If

'Entity (Ops) name

EntityRecord(TotalEntCount, EntName) = RS.Fields("Ops_No")

'Entity (Ops) part ID

EntityRecord(TotalEntCount, EntPart) = RS.Fields("Part_ID")

'Entity (Ops) Station

EntityRecord(TotalEntCount, EntST) = RS.Fields("Mach_ID")

EntityRecord(TotalEntCount, EntMachName) = RS.Fields("Mach_Name")

EntityRecord(TotalEntCount, EntSuc) = "0"

EntityRecord(TotalEntCount, EntProg) = RS.Fields("Status_id")

EntityRecord(TotalEntCount, entTotMakespan) = RS.Fields("CSetup_Time")_

+ RS.Fields("CInsp_Time") + RS.Fields("CProd_Time")

'check status off entities to determine start times and remaining processing times

If EntityRecord(TotalEntCount, EntProg) = 2 Then

'busy entity = starttime < sim starttime but start is set to 0 and remaining set/insp/est time are assigned

EntityRecord(TotalEntCount, EntStart) = DateDiff("h",_ oModel.StartDateTime, RS.Fields("Start_DateTime"))

EntityRecord(TotalEntCount, EntTime) = RS.Fields("Prod_Time") - RS.Fields("CProd_Time") * (RS.Fields("Qty") - 1)

'Entity (Ops) process time

EntityRecord(TotalEntCount, EntSet) = RS.Fields("Setup_Time") - RS.Fields("CSetup_Time") + RS.Fields("Prod_Time")

'Entity (Ops) Setup time

```

EntityRecord(TotalEntCount, EntInsp) = RS.Fields("Insp_Time") -_
RS.Fields("CInsp_Time")
'Entity (Ops) Inspection time"
EntityRecord(TotalEntCount, EntTotTime) =_ EntityRecord(TotalEntCount,
EntInsp) + EntityRecord(TotalEntCount, _ EntSet) +
EntityRecord(TotalEntCount, EntTime)

Else
  If EntityRecord(TotalEntCount, EntProg) = 3 Then
    'Completed entity = start time < sim starttime
EntityRecord(TotalEntCount, EntStart) = DateDiff("h", -
oModel.StartDateTime, RS.Fields("Start_DateTime"))
EntityRecord(TotalEntCount, EntTime) = 0
EntityRecord(TotalEntCount, EntSet) = 0
EntityRecord(TotalEntCount, EntInsp) = 0
EntityRecord(TotalEntCount, EntTotTime) = 0
  Else
    'Queueable entity = start time > sim starttime
EntityRecord(TotalEntCount, EntStart) = 0
    'Entity (Ops) process time
EntityRecord(TotalEntCount, EntTime) = RS.Fields("Prod_Time")
    'Entity (Ops) Setup time
EntityRecord(TotalEntCount, EntSet) = RS.Fields("Setup_Time")
    'Entity (Ops) Inspection time"
EntityRecord(TotalEntCount, EntInsp) = RS.Fields("Insp_Time")
EntityRecord(TotalEntCount, EntTotTime) =_ EntityRecord(TotalEntCount,
EntInsp) +_ EntityRecord(TotalEntCount, EntSet) +_
EntityRecord(TotalEntCount, EntTime)
  End If
End If

'enter due time on due date (use constant value of 12:00:00)
entityDaterecord(TotalEntCount, dateProm) = RS.Fields("PromDate") &_
"12:00:00"
entityDaterecord(TotalEntCount, dateStart) = RS.Fields("ApprovedDate")_
& " 08:00:00"

End Sub

Private Sub AssignEntSuccessor()

  'Set entity successor - no sucesor if last ops of part
  If Not RS.EOF Then
    'check if current entity has the same part_id as previous entity, then
    current entity is successor of previous entity
    If EntityRecord(TotalEntCount - 1, EntPart) = RS.Fields("Part_ID")_
      Then
        EntityRecord(TotalEntCount - 1, EntSuc) = RS.Fields("Ops_No")
        EntityRecord(TotalEntCount, EntPredecessor) =_
        EntityRecord(TotalEntCount - 1, EntName)
      Else
        EntityRecord(TotalEntCount - 1, EntSuc) = "0"
      End If
    End If
  End If

End Sub

Private Sub CleanEntities()

```

```

'Clean entity modules
Do While CleanCount < TotalEntCount

    Model.ActiveView.Selection.DeselectAll
    Model.Shapes(EntityRecord(TotalEntCount - CleanCount, EntAddr) -_
        67).Selected = True
    Model.ActiveView.Selection.Delete

    CleanCount = CleanCount + 1

Loop
End Sub

Private Sub VBA_Block_1_Fire()

    ' assign attribute values to entity
    AssignEntAttr

End Sub

Private Sub AssignEntAttr()

    'Assign entity attributes from entity array
    If Counter < TotalEntCount Then

        osiman.entityType(osiman.ActiveEntity) = Counter + 1
        'if entity is active, then real value must be assigned from entity array
        If EntityRecord(Counter, EntProg) < Busy Then
            If EntityRecord(Counter, EntProg) = Active Then
                osiman.EntityAttribute(osiman.ActiveEntity, attrDD) =_
                    EntityRecord(Counter, EntDD)
                osiman.EntityAttribute(osiman.ActiveEntity, attrStartTime) =_
                    EntityRecord(Counter, EntStart)
                osiman.EntityAttribute(osiman.ActiveEntity, attrTotTime) =_
                    EntityRecord(Counter, EntTotTime)
                osiman.EntityAttribute(osiman.ActiveEntity, attrSlack) =_
                    EntityRecord(Counter, EntSlack)
                osiman.EntityAttribute(osiman.ActiveEntity, attrCR) =_
                    EntityRecord(Counter, EntCR)
            Else
                'assign dummy value to entity processing time attribute, because entity is
                inactive
                If Dispatch = 3 Or Dispatch = 4 Or Dispatch = 5 Or_ Dispatch = 6 Then
                    'big value of DD for EDD
                    osiman.EntityAttribute(osiman.ActiveEntity, attrDD) = 999999999
                    'big value of StartTime for EST
                    osiman.EntityAttribute(osiman.ActiveEntity, attrStartTime) = 999999999
                    'big value of Processing time for SPT
                    osiman.EntityAttribute(osiman.ActiveEntity, attrTotTime) = 999999999
                    'big value of Slack for SS
                    osiman.EntityAttribute(osiman.ActiveEntity, attrSlack) = 999999999
                Else
                    'Assign real value to DD, because DD doesn't apply
                    osiman.EntityAttribute(osiman.ActiveEntity, attrDD) =
                    EntityRecord(Counter, EntDD)
                    'Assign real value to slack, because slack doesn't apply

```

```

osiman.EntityAttribute (osiman.ActiveEntity, _ attrSlack) =
EntityRecord(Counter, EntSlack)
'Assign real value to StartTime, because EST doesn't apply
osiman.EntityAttribute (osiman.ActiveEntity, attrStartTime) =
EntityRecord(Counter, EntStart)
'small value of Processing time for LPT
osiman.EntityAttribute (osiman.ActiveEntity, attrTotTime) = 0
'small value of CR for CR
osiman.EntityAttribute (osiman.ActiveEntity, _ attrCR) = 0
End If
End If
Else
osiman.EntityAttribute (osiman.ActiveEntity, attrDD) =
EntityRecord(Counter, EntDD)
osiman.EntityAttribute (osiman.ActiveEntity, attrSlack) =
EntityRecord(Counter, EntSlack)
osiman.EntityAttribute (osiman.ActiveEntity, attrStartTime) =
EntityRecord(Counter, EntStart)
osiman.EntityAttribute (osiman.ActiveEntity, attrTotTime) =
EntityRecord(Counter, EntTotTime)
End If

osiman.EntityAttribute (osiman.ActiveEntity, attrTime) =
EntityRecord(Counter, EntTime)
osiman.EntityAttribute (osiman.ActiveEntity, attrEnt_Name) =
EntityRecord(Counter, EntName)
osiman.EntityAttribute (osiman.ActiveEntity, attrEnt_Successor) =
EntityRecord(Counter, EntSuc)
osiman.EntityAttribute (osiman.ActiveEntity, attrCompletionTime) =
EntityRecord(Counter, EntInsp)
osiman.EntityAttribute (osiman.ActiveEntity, attrLocation) =
EntityRecord(Counter, EntAddr)
osiman.EntityAttribute (osiman.ActiveEntity, attrSetup) =
EntityRecord(Counter, EntSet)
osiman.EntityAttribute (osiman.ActiveEntity, attrInspection) =
EntityRecord(Counter, EntInsp)
osiman.EntityAttribute (osiman.ActiveEntity, attrProgress) =
EntityRecord(Counter, EntProg)
osiman.EntityAttribute (osiman.ActiveEntity, attrPartsize) =
EntityRecord(Counter, PartSize)
osiman.EntityAttribute (osiman.ActiveEntity, attrPart_ID) =
EntityRecord(Counter, EntPart)

'get entity location as stored in Arena
EntityRecord(Counter, EntLoc) = osiman.ActiveEntity

'assign station number
osiman.EntityStationAttribute (osiman.ActiveEntity) =
EntityRecord(Counter, EntST)

Counter = Counter + 1
End If

End Sub

Private Sub DetermineNumOps ()

temp = 1

```

```

parts = 1
CompletedHoursAtSimStart = 0
'determine number of ops per part
For x = 1 To TotalEntCount - 1
  If x > 1 Then
    'check if previous entity in array is same as current, then
    count it as temp
    If EntityRecord(x - 1, EntPart) = EntityRecord(x, EntPart) Then
      temp = temp + 1
    'repeat until new parts ops starts
  Else
    'assign partsize to entities from last entity of part set
    backwards
    y = x - 1      'entity number in entity record array(1 -
                  TotalEntCount)
    z = 1          'Temp var used in DetermineStartTime
    tempPart = temp 'number of entities in part set

    'Do loop is used to start from last entity of part set
    'and move backwards to first entity of part set, temp is
    'set to zero when first entity of part set ha been assigned
    Do While temp > 0
      EntityRecord(y, PartSize) = tempPart
      EntityRecord(y, EntType) = parts

      DetermineDueDate
      DetermineStartTime (0)

      CompletedHoursAtSimStart = CompletedHoursAtSimStart +_
        EntityRecord(y, entTotMakespan)

      z = z + 1
      y = y - 1
      temp = temp - 1
    Loop

    EntityRecord(x - 1, entTotMakespan) =_
      CompletedHoursAtSimStart

    'reset temp that next part set can start at 1 entity
    temp = 1
    'count number of parts (not number of entities)
    parts = parts + 1
  End If
  CompletedHoursAtSimStart = 0
  'The last entity must be treated differently
  If x = TotalEntCount - 1 Then

    y = x
    z = 1
    tempPart = temp
    Do While temp > 0

      EntityRecord(y, PartSize) = tempPart
      EntityRecord(y, EntType) = parts

      DetermineDueDate
      DetermineStartTime (0)

```

```

CompletedHoursAtSimStart = CompletedHoursAtSimStart +_
    EntityRecord(y, entTotMakespan)

z = z + 1
y = y - 1
temp = temp - 1
Loop

EntityRecord(x, entTotMakespan) = CompletedHoursAtSimStart

temp = 1
End If
End If
Next x

```

End Sub

Private Sub DetermineDueDate()

Dim TempDD As Date

TempDD = CDate(entityDaterecord(y, dateProm))

'determine due dates of all the entities of the current part set

If tempPart = temp Then

'DD of last entity of part set = dd of part set

'HoursToDD is equal to the difference in model start time and

'promised date, equals the total hours difference - it must be

'converted into work hours difference the DD slack must be removed

'as well

DaysInMinutes = DateDiff("n", oModel.StartDateTime, TempDD)

Days = DaysInMinutes / 1440

daysinhours = (Fix(Days) * 9.5) + ((Days - Fix(Days)) * 24)

EntityRecord(y, EntDD) = daysinhours

For i = 1 To tempPart - 1

'determine dd of entities in part set from backwards

daysinhours = daysinhours - EntityRecord(y - i + 1, EntTotTime)

EntityRecord(y - i, EntDD) = daysinhours

Next i

End If

End Sub

Private Sub DetermineStartTime(Origin As Integer)

st = EntityRecord(y - temp + z, EntStart)

'when z = 1 then it is the first entity of the part set

If z = 1 Then

'start times can only change whilst entity is inactive,

'accept for the first entities of a part set it can change if

'its not completed


```

If EntityRecord(y - temp + z, EntProg) = Active Then

    Slack = EntityRecord(y - temp + z, EntDD) - EntityRecord(y_
        - temp + z, EntStart) - EntityRecord(y - temp + z, _
        EntTotTime)
    'check if it is not the first time that start times are
    'being calculated origin = 0 for the first time start times
    'are being calculated
    If Origin = 1 Then
        'get the new start time of the first entity in the part
        'set, it will be the difference between the original
        'starttime and the current starttime (assumed to start
        'on current time)
        If EntityRecord(y - temp + z + 1, EntPart) = _
            EntityRecord(y - temp + z, EntPart) Then
            st = osiman.RunCurrentTime - EntityRecord(y - temp_
                + z, EntStart)
            EntityRecord(y - temp + z, EntStart) = st
            Slack = EntityRecord(y - temp + z, EntDD) - _
                EntityRecord(y - temp + z, EntStart) - _
                EntityRecord(y - temp + z, EntTotTime)
        End If
    End If
End If
Else
If EntityRecord(y - temp + z, EntProg) = Inactive Or_
EntityRecord(y - temp + z, EntProg) = Active Then
    'if the entity before the current entity in the same part set
    'is completed, and if processing on current entity has not
    'started the start time of the current entity is equal to the
    'current time
    If EntityRecord(y - temp + z - 1, EntProg) = Complete Then
        If Origin = 1 Then
            EntityRecord(y - temp + z, EntStart) = _
                osiman.RunCurrentTime
        End If
        Slack = EntityRecord(y - temp + z, EntDD) - _
            EntityRecord(y - temp + z, EntStart) - _
            EntityRecord(y - temp + z, EntTotTime)
    Else
        'If entity before current entity in the same part set is
        'busy being processed, the earliest start time of the
        'current entity is after processing the previous entity
        'is finished
        If EntityRecord(y - temp + z - 1, EntProg) = Busy Then
            EntityRecord(y - temp + z, EntStart) = _
                EntityRecord(y - temp + z - 1, EntTotTime)
            Slack = EntityRecord(y - temp + z, EntDD) - _
                EntityRecord(y - temp + z, EntStart) - _
                EntityRecord(y - temp + z, EntTotTime)
        Else
            'earliest start time equals start time of previous
            'entity in part set plus its processing time
            st = EntityRecord(y - temp + z - 1, EntStart) + _
                EntityRecord(y - temp + z - 1, EntTotTime)
            EntityRecord(y - temp + z, EntStart) = st
        End If
    End If
End If

```

```

        Slack = EntityRecord(y - temp + z, EntDD) -_
        EntityRecord(y - temp + z, EntStart) -_
        EntityRecord(y - temp + z, EntTotTime) -_
    End If
End If
End If
End If

'the entity has no slack if processing started or completed on it
If EntityRecord(y - temp + z, EntProg) < 2 Then
    EntityRecord(y - temp + z, EntSlack) = Slack
Else
    EntityRecord(y - temp + z, EntSlack) = 0
End If

End Sub

Private Sub determineSlack(Origin As Integer)

    temp = 1
    parts = 1

    For x = 1 To TotalEntCount - 1
        If x > 1 Then
            If EntityRecord(x, EntType) = EntityRecord(x - 1, EntType)_
            Then
                'count number of entities in current part set
                temp = temp + 1
            Else
                'determine slack of entity in part set when all entities in
                'part set have been counted
                y = x - 1
                z = 1

                Do While temp > 0
                    'get due date and processing time of next entity in
                    'part set
                    Slack = EntityRecord(y - temp + z, EntDD) -_
                    EntityRecord(y - temp + z, EntStart) -_
                    EntityRecord(y - temp + z, EntTotTime) -_
                    z = z + 1
                    y = y - 1
                    temp = temp - 1
                Loop

                temp = 1
            End If

            If x = TotalEntCount - 1 Then
                y = x
                z = 1
                Do While temp > 0
                    'get due date and processing time of next entity in
                    'part set
                    Slack = EntityRecord(y - temp + z, EntDD) -_
                    EntityRecord(y - temp + z, EntStart) -_
                    EntityRecord(y - temp + z, EntTotTime) -_
                    z = z + 1
                Loop
            End If
        End If
    Next x
End Sub

```

```

        y = y - 1
        temp = temp - 1
    Loop
    temp = 1
End If

End If

Next x

End Sub

Private Sub UpdateStartTimes ()

    temp = 1
    parts = 1

    For x = 1 To TotalEntCount - 1
        If x > 1 Then
            If EntityRecord(x, EntType) = EntityRecord(x - 1, EntType) Then
                temp = temp + 1
            Else
                y = x - 1
                z = 1

                Do While temp > 0
                    DetermineStartTime (1)
                    'determineSlack
                    z = z + 1
                    y = y - 1
                    temp = temp - 1
                Loop

                temp = 1
            End If

            If x = TotalEntCount - 1 Then
                y = x
                z = 1
                Do While temp > 0

                    DetermineStartTime (1)
                    'determineSlack
                    z = z + 1
                    y = y - 1
                    temp = temp - 1
                Loop
                temp = 1
            End If

        End If

    Next x

End Sub

Private Sub updateStartProgress ()

```

```

osiman.EntityAttribute(osiman.ActiveEntity, attrProgress) = Busy
tempstart = osiman.RunCurrentTime
osiman.EntityAttribute(osiman.ActiveEntity, attrStartTime) = tempstart

'change entity attribute status
q = 1
Do While q <= TotalEntCount
  If osiman.EntityAttribute(osiman.ActiveEntity, attrEnt_Name) = _
    EntityRecord(q, EntName) Then
    EntityRecord(q, EntProg) = Busy
    EntityRecord(q, EntStart) = tempstart

    'convert hours into days using minutes
    HoursInDays = tempstart / 9.5
    DaysInMinutes = (Fix(HoursInDays) * 24 * 60) + ((HoursInDays - _
      Fix(HoursInDays)) * 9 * 60)
    UpdateSimOpsTable CInt(EntityRecord(q, EntName)), "Status_Id", _
      CDb1(Busy)
    UpdateSimOpsTable CInt(EntityRecord(q, EntName)), _
      "Start_DateTime", CDb1(tempstart) 'DaysInMinutes / nReps
  End If
  q = q + 1
Loop

```

End Sub

```

Function UpdateSimOpsTable(Ops_No As Integer, Column As String, NewValue As Double)

```

```

'Scan trough records to find particular one and update accordingly
TempRS.MoveFirst

Do While Not TempRS.EOF
  If TempRS.Fields("Ops_no") = Ops_No Then
    TempRS.Edit

    If Column = "Status_Id" Then
      TempRS.Fields(Column).value = NewValue
    Else
      TempRS.Fields(Column).value = NewValue + _
        TempRS.Fields(Column).value
    End If

    TempRS.Update
  End If
  TempRS.MoveNext
Loop
End Function

```

Private Sub updateCompleteProgress()

```

Dim pic As Long
Dim SetTemp, InspTemp, ProdTemp As Double

'assign completed status to processed entity
osiman.EntityAttribute(osiman.ActiveEntity, attrProgress) = Complete
osiman.EntityAttribute(osiman.ActiveEntity, attrCompletionTime) = _
  osiman.RunCurrentTime

```

```

Ops_No = osiman.EntityAttribute(osiman.ActiveEntity, attrEnt_Name)

'change entity picture to resemble completed status (green)
q = 1
Do While q <= TotalEntCount

    If osiman.EntityAttribute(osiman.ActiveEntity, attrEnt_Name) = _
        EntityRecord(q, EntName) Then
        osiman.EntityAttribute(osiman.ActiveEntity, attrStartTime) = _
            EntityRecord(q, EntStart)
        osiman.EntityAttribute(osiman.ActiveEntity, attrTotMakespan) = _
            osiman.RunCurrentTime + EntityRecord(q, entTotMakespan)

        EntityRecord(q, EntProg) = Complete
        EntityRecord(q, EntEnd) = osiman.RunCurrentTime
        endtime = osiman.RunCurrentTime
        pic = EntityRecord(q, EntType) * 3 - 2
        osiman.EntitySetPicture EntityRecord(q, EntLoc), pic + 2

        'determine date that ops was finished - convert ammount of
        'hours into days that it could be added to the start date and
        'time of the simulation
        HoursInDays = EntityRecord(q, EntEnd) / 9.5
        DaysInMinutes = (Fix(HoursInDays) * 24 * 60) + ((HoursInDays - _
            Fix(HoursInDays)) * 9.5 * 60)

        'Update completed time on each type of operation on ops
        SetTemp = CDb1(osiman.EntityAttribute(EntityRecord(q, EntLoc), _
            attrCSetup_Time) / nReps)
        InspTemp = CDb1(osiman.EntityAttribute(EntityRecord(q, EntLoc) _
            , attrCInsp_Time) / nReps)
        ProdTemp = CDb1(osiman.EntityAttribute(EntityRecord(q, EntLoc) _
            , attrCProd_Time) / nReps)
        'update the dynamic simulation table records
        UpdateSimOpsTable CInt(Ops_No), "CSetup_Time", CDb1(SetTemp)
        UpdateSimOpsTable CInt(Ops_No), "CInsp_Time", CDb1(InspTemp)
        UpdateSimOpsTable CInt(Ops_No), "CProd_Time", CDb1(ProdTemp)

    End If
    q = q + 1
Loop

'update the dynamic simulation table records
'Update status to completed
UpdateSimOpsTable CInt(Ops_No), "Status_Id", CDb1(Complete)

'update Finished date
EndDate = DateAdd("n", DaysInMinutes, oModel.StartDateTime)
UpdateSimOpsTable CInt(Ops_No), "End_DateTime", CDb1(endtime)
'DaysInMinutes / nReps

UpdateStartTimes

'check if entity has successor, then change successor status to active
If osiman.EntityAttribute(osiman.ActiveEntity, attrEnt_Successor) <> 0 _
    Then
        UpdateSuccessor
    End If

```

DetermineCR

End Sub

Private Sub UpdateSnapShotAtEndRep()

Dim TempVal As Double

```
'update information on ops that are still being processed
For i = 1 To TotalEntCount - 1
  'find all ops that are busy
  If EntityRecord(i, EntProg) = Busy Then
    'check what type of operation is being done on ops - if the
    'Process attr has the value 1 it has been completed
    If osiman.EntityAttribute(EntityRecord(i, EntLoc), _
      attrCSetup_Process) = 1 Then
      'setup has been completed - record in simulation database
      'check if Csetup_time has a value, meaning that work has been
      'done on setup process then the new amount of work must be
      'added to the previous ammount
      TempRS.MoveFirst
      RS.MoveFirst
      Do While Not TempRS.EOF
        If TempRS.Fields("Ops_no") = EntityRecord(i, EntName) _
          Then
          Then
            TempVal = CDb1(osiman.EntityAttribute(EntityRecord_
              (i, EntLoc), attrCSetup_Time) / nReps)
          End If
          TempRS.MoveNext
          RS.MoveNext
        Loop
        UpdateSimOpsTable CInt(EntityRecord(i, EntName)), _
          "CSetup_Time", TempVal

      If osiman.EntityAttribute(EntityRecord(i, EntLoc), _
        attrCInsp_Process) = 1 Then
        'Inspection has been completed - record in simulation
        'database check if CInsp_time has a value, meaning that
        'work has been done on Insp process then the new amount
        'of work must be added to the previous ammount
        TempRS.MoveFirst
        RS.MoveFirst
        Do While Not TempRS.EOF
          If TempRS.Fields("Ops_no") = EntityRecord(i, _
            EntName) Then
            Then
              TempVal = CDb1(osiman.EntityAttribute_
                (EntityRecord(i, EntLoc), attrCInsp_Time) / _
                nReps)
            End If
            TempRS.MoveNext
            RS.MoveNext
          Loop
          UpdateSimOpsTable CInt(EntityRecord(i, EntName)), _
            "CInsp_Time", TempVal
```

```

If  osiman.EntityAttribute(EntityRecord(i,  EntLoc),_
attrCProd_Process) = 1 Then
'Ops has been completed, but the simulation stopped
'before it could be saved as completed check if
'CProd_time has a value, meaning that work has been
'done on Prod process then the new amount of work must
'be added to the previous amount
  TempRS.MoveFirst
  RS.MoveFirst
  Do While Not TempRS.EOF
    If TempRS.Fields("Ops_no") = EntityRecord(i,_
EntName) Then
      TempVal = Cdbl(osiman.EntityAttribute_
(EntityRecord(i, EntLoc), attrCProd_Time)_
/ nReps)
    End If
    TempRS.MoveNext
    RS.MoveNext
  Loop
  UpdateSimOpsTable CInt(EntityRecord(i, EntName)),_
"CProd_Time", TempVal

Else
'Production process is not completed check if
'CProd_time has a value, meaning that work has been
'done on Prod process then the new amount of work
'must be added to the previous amount
  TempVal = Cdbl(osiman.RunCurrentTime -_
osiman.EntityAttribute(EntityRecord(i,  EntLoc),_
attrProd_Start))
  TempRS.MoveFirst
  RS.MoveFirst
  Do While Not TempRS.EOF
    If TempRS.Fields("Ops_no") = EntityRecord(i,_
EntName) Then
      TempVal = TempVal / nReps
    End If
    TempRS.MoveNext
    RS.MoveNext
  Loop
  UpdateSimOpsTable CInt(EntityRecord(i, EntName)),_
"CProd_Time", TempVal

End If
Else
'The Inspection process is not completed check if
'CInsp_time has a value, meaning that work has been
'done on Insp process then the new amount of work must
'be added to the previous amount
  TempVal = Cdbl(osiman.RunCurrentTime -_
osiman.EntityAttribute(EntityRecord(i, EntLoc),_
attrInsp_Start))
  TempRS.MoveFirst
  RS.MoveFirst
  Do While Not TempRS.EOF
    If TempRS.Fields("Ops_no") = EntityRecord(i,_
EntName) Then

```

```

        TempVal = TempVal / nReps
    End If
    TempRS.MoveNext
    RS.MoveNext
Loop
UpdateSimOpsTable CInt(EntityRecord(i, EntName)),_
    "CInsp_Time", TempVal

End If
Else
'The setup process is not completed, thus the completed
'time of setup must be completed check if Csetup_time has a
'value, meaning that work has been done on setup process
'then the new amount of work must be added to the previous
'\ammount
TempVal = CDb1(osiman.RunCurrentTime -osiman._
    EntityAttribute(EntityRecord(i, EntLoc),_
        attrSetup_Start))
TempRS.MoveFirst
RS.MoveFirst
Do While Not TempRS.EOF
    If TempRS.Fields("Ops_no") = EntityRecord(i, EntName)_
        Then
            TempVal = TempVal / nReps
        End If
        TempRS.MoveNext
        RS.MoveNext
Loop
UpdateSimOpsTable CInt(EntityRecord(i, EntName)),_
    "CSetup_Time", TempVal

End If
End If
Next i

```

End Sub

Private Sub UpdateSuccessor()

```

q = 1

Do While q <= TotalEntCount
    'find entity successor and change status and picture
    If osiman.EntityAttribute(osiman.ActiveEntity, attrEnt_Successor)_
        = EntityRecord(q, EntName) Then
        EntityRecord(q, EntProg) = Active    'update entity array
        osiman.EntityAttribute(EntityRecord(q, EntLoc), attrTotTime) =_
            EntityRecord(q, EntTotTime)    'update entity time attr
        osiman.EntityAttribute(EntityRecord(q, EntLoc), attrProgress)_
            = Active    'update entity progress attr
        osiman.EntityAttribute(EntityRecord(q, EntLoc), attrDD) =_
            EntityRecord(q, EntDD)    'update entity DD attr
        'update entity earliest Start time attr
        osiman.EntityAttribute(EntityRecord(q, EntLoc), attrStartTime)_
            = EntityRecord(q, EntStart)
        osiman.EntityAttribute(EntityRecord(q, EntLoc), attrSlack) =_
            EntityRecord(q, EntSlack)    'update entity slack attr
    End If
    q = q + 1
End Do

```



```

osiman.EntityAttribute(EntityRecord(q, EntLoc), attrCR) =_
EntityRecord(q, EntCR)      'update entity critical ratio attr

'change status attr in simulation database
UpdateSimOpsTable CInt(EntityRecord(q, EntName)), "Status_Id",_
CInt(Active)

'there are 3 types of the same picture
pic = EntityRecord(q, EntType) * 3 - 2
osiman.EntitySetPicture EntityRecord(q, EntLoc), pic + 1

'find queue that contains newly updated active entity
For qNum = 1 To osiman.QueuesMaximum
  For qRank = 1 To osiman.QueueNumberOfEntities(qNum)
    If osiman.QueueEntityLocationAtRank(qRank, qNum) =_
EntityRecord(q, EntLoc) Then
      'reshuffle queue according to selected dispatching rule
      Reshuffle (qNum)
      Exit Do
    End If
  Next qRank
Next qNum

End If
q = q + 1
Loop

```

End Sub

Private Sub DetermineCR()

```

i = 2
pt = 0
tempPartSize = 1
'find CR for all the entities
Do While i <= TotalEntCount - 1
  'if 2 entities that follow each other have the same entPart they
  'belong to the same part set and the PT of the 1st one must be
  'considered
  If EntityRecord(i - 1, EntPart) = EntityRecord(i, EntPart) Then
    tempPartSize = tempPartSize + 1
    'if work has not started on previous entity, the PT of the
    'entity must be included in the remaining PT of the part
    If EntityRecord(i - 1, EntProg) <= Active Then
      pt = pt + EntityRecord(i - 1, EntTotTime)
    Else
      'if work is still being done, the remaining PT of the
      'entity must be added to the remaining PT of the part
      If EntityRecord(i - 1, EntProg) = Busy Then
        If EntityRecord(i - 1, EntStart) < 0 Then
          pt = EntityRecord(i - 1, EntTotTime)
        Else
          pt = osiman.RunCurrentTime - EntityRecord(i - 1, _
EntStart)
        End If
      'else if it is completed, its PT doesn't play a role
    Else
      pt = 0
    End If
  End If
  i = i + 1

```

```

End If
'if the 2 entities don't match, a new part set has started.
'thus the last entity of the previous part determine the CR of the
'part. After CR is determined, PT must be set to zero - in case
'there is a part with only 1 entity (or else the it will use the PT
'of the previous part)
Else
  If EntityRecord(i - 1, EntProg) = Busy Then
    pt = osiman.RunCurrentTime - EntityRecord(i - 1, EntStart)
    If CInt(EntityRecord(i - 1, EntDD) - osiman.RunCurrentTime_
      ) <= 0 Then
      CR(EntityRecord(i - 1, EntName)) = pt / 1E-18
    Else
      CR(EntityRecord(i - 1, EntName)) = pt / Cint_
        (EntityRecord(i - 1, EntDD) - osiman.RunCurrentTime)
    End If
    pt = 0
  End If
  If EntityRecord(i - 1, EntProg) <= Active Then
    pt = pt + EntityRecord(i - 1, EntTotTime)
    'if the due date of the entity has passed, the CR must be
    'really big
    If CInt(EntityRecord(i - 1, EntDD) - osiman._
      RunCurrentTime) <= 0 Then
      CR(EntityRecord(i - 1, EntName)) = pt / 1E-18
    Else
      CR(EntityRecord(i - 1, EntName)) = pt / Cint_
        (EntityRecord(i - 1, EntDD) - osiman.RunCurrentTime)
    End If
    pt = 0
  End If
  'if it is complete, the part is also complete then the CR = zero
  If EntityRecord(i - 1, EntProg) = Complete Then
    pt = 0
    CR(EntityRecord(i - 1, EntName)) = 0
  End If
  'CR of the last ops of part set has been calculated, assign
  'same value to each ops of the part set
  For a = 1 To tempPartSize - 1
    CR(EntityRecord(i - 1, EntName) - tempPartSize + a) =
      CR(EntityRecord(i - 1, EntName))
  Next a
  tempPartSize = 1
End If

'if the current entity is the last entity of the last part
'the CR must be computed
If i = TotalEntCount - 1 Then
  If EntityRecord(i, EntProg) = Busy Then
    pt = pt + osiman.RunCurrentTime - EntityRecord(i, EntStart)
    If CInt(EntityRecord(i, EntDD) - osiman.RunCurrentTime) <=_
      0 Then
      CR(EntityRecord(i, EntName)) = pt / 1E-18
    Else
      CR(EntityRecord(i, EntName)) = pt / Cint_
        (EntityRecord(i, EntDD) - osiman.RunCurrentTime)
    End If
    pt = 0
  End If

```

```

End If
If EntityRecord(i, EntProg) <= Active Then
    pt = pt + EntityRecord(i, EntTotTime)
    If Cint(EntityRecord(i, EntDD) - osiman.RunCurrentTime) <= 0 Then
        CR(EntityRecord(i, EntName)) = pt / 1E-18
    Else
        CR(EntityRecord(i, EntName)) = pt / Cint_
            (EntityRecord(i, EntDD) - osiman.RunCurrentTime)
    End If
    pt = 0
End If
'if it is complete, the part is also complete then the CR = zero
If EntityRecord(i, EntProg) = Complete Then
    pt = 0
    CR(EntityRecord(i, EntName)) = 0
End If
'CR of the last ops of part set has been calculated, assign
'same value to each ops of the part set
For a = 1 To tempPartSize - 1
    CR(EntityRecord(i, EntName) - tempPartSize + a) =
        CR(EntityRecord(i, EntName))
Next a
End If
i = i + 1
Loop

For i = 1 To TotalEntCount - 1
    EntityRecord(i, EntCR) = CR(EntityRecord(i, EntName))
Next i

```

End Sub

Private Sub Reshuffle(qNum As Integer)

```

Dim qSize, qCount, qLoc, Loc As Integer
Dim qAttr(100, 5) As Integer

```

```

'reset temp variables
qSize = osiman.QueueNumberOfEntities(qNum)
qCount = 1
Loc = 1
For i = 1 To 100
    For k = 1 To 5
        qAttr(i, k) = 0
    Next k
Next i

'find the locations of all the entities in the queue
Do While qCount <= qSize
    qLoc = osiman.QueueEntityLocationAtRank(qCount, qNum)
    qAttr(qCount, Loc) = qLoc
    qCount = qCount + 1
Loop

'remove all entities from queue and reinsert them, FIFO and active
'first will automatically apply
qCount = 1

```

```

Do While qCount <= qSize
    osiman.QueueRemoveEntity qAttr(qCount, Loc), qNum
    osiman.EntityInsertIntoQueueByRank qAttr(qCount, Loc), qNum
    blab = osiman.EntityAttribute(qAttr(qCount, Loc), attrDD)
    qCount = qCount + 1
Loop

```

End Sub

Private Sub DetermineLateness()

```

'Determine if part was late and record value to total lateness
i = osiman.SymbolNumber("varTotLateness")
lateness = osiman.EntityAttribute(osiman.ActiveEntity, attrDD) -_
    osiman.EntityAttribute(osiman.ActiveEntity, attrCompletionTime)
If lateness < 0 Then
    osiman.VariableArrayValue(i) = osiman.VariableArrayValue(i) +_
        Abs(lateness)
End If

```

End Sub

Private Sub DetermineEarliness()

```

'Determine if part was early and record value to total earliness
i = osiman.SymbolNumber("varTotEarliness")
Earliness = osiman.EntityAttribute(osiman.ActiveEntity, attrDD) -_
    osiman.EntityAttribute(osiman.ActiveEntity, attrCompletionTime)
If Earliness > 0 Then
    osiman.VariableArrayValue(i) = osiman.VariableArrayValue(i) +_
        Earliness
End If

```

End Sub

Private Sub VBA_Block_19_Fire()

```

'lateness and earliness are calculated in VBA because the total of each
'must only be adjusted when it is actually a late entity or early entity,
'which ever applies a record module doesn't distinguish if it is late or
'not, it just adds the value
    DetermineLateness
    DetermineEarliness

    CellCount = CellCount + 1

    'write Part ID
    Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range("T" &_
        CellCount).value = osiman.EntityAttribute(osiman.ActiveEntity, -_
            attrPart_ID)
    'write Part DD
    Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range("U" &_
        CellCount).value = osiman.EntityAttribute(osiman.ActiveEntity, -_
            attrDD)
    'write completion time of part
    Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range("V" &_
        CellCount).value = osiman.EntityAttribute(osiman.ActiveEntity, -_
            attrCompletionTime)

```

```

If      osiman.EntityAttribute(osiman.ActiveEntity,      attrDD)      =_
osiman.EntityAttribute(osiman.ActiveEntity, attrCompletionTime) < 0_
Then
    Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range("W"_
        & CellCount).value = Abs(osiman.EntityAttribute_
            (osiman.ActiveEntity, attrDD) - osiman.EntityAttribute_
            (osiman.ActiveEntity, attrCompletionTime))
Else
    Excel.Application.Workbooks(1).Worksheets(Dispatch + 1).Range("X"_
        & CellCount).value = osiman.EntityAttribute(osiman.ActiveEntity,_
            attrDD) - osiman.EntityAttribute(osiman.ActiveEntity,_
            attrCompletionTime)
End If

End Sub

Private Sub UpdateAttrTimesforBusyEnt ()

    osiman.EntityAttribute(osiman.ActiveEntity,      attrSetup)      =_
        EntityRecord(osiman.EntityAttribute(osiman.ActiveEntity,_
            attrEnt_Name), EntSet)
    osiman.EntityAttribute(osiman.ActiveEntity,      attrInspection)  =_
        EntityRecord(osiman.EntityAttribute(osiman.ActiveEntity,_
            attrEnt_Name), EntInsp)
    osiman.EntityAttribute(osiman.ActiveEntity,      attrTime)      =_
        EntityRecord(osiman.EntityAttribute(osiman.ActiveEntity,_
            attrEnt_Name), EntTime)

End Sub

Private Sub VBA_Block_2_Fire()
    updateCompleteProgress
End Sub

Private Sub VBA_Block_20_Fire()
    updateCompleteProgress
End Sub

Private Sub VBA_Block_22_Fire()
    UpdateAttrTimesforBusyEnt
End Sub

Private Sub VBA_Block_23_Fire()
    UpdateAttrTimesforBusyEnt
End Sub

Private Sub VBA_Block_24_Fire()
    UpdateAttrTimesforBusyEnt
End Sub

Private Sub VBA_Block_25_Fire()
    UpdateAttrTimesforBusyEnt
End Sub

Private Sub VBA_Block_26_Fire()
    UpdateAttrTimesforBusyEnt
End Sub

```

```
Private Sub VBA_Block_27_Fire()  
    UpdateAttrTimesforBusyEnt  
End Sub  
  
Private Sub VBA_Block_28_Fire()  
    UpdateAttrTimesforBusyEnt  
End Sub  
  
Private Sub VBA_Block_29_Fire()  
    UpdateAttrTimesforBusyEnt  
End Sub  
  
Private Sub VBA_Block_3_Fire()  
    updateCompleteProgress  
End Sub  
  
Private Sub VBA_Block_4_Fire()  
    updateCompleteProgress  
End Sub  
  
Private Sub VBA_Block_5_Fire()  
    updateCompleteProgress  
End Sub  
  
Private Sub VBA_Block_6_Fire()  
    updateCompleteProgress  
End Sub  
  
Private Sub VBA_Block_7_Fire()  
    updateCompleteProgress  
End Sub  
  
Private Sub VBA_Block_8_Fire()  
    updateCompleteProgress  
End Sub  
  
Private Sub VBA_Block_21_Fire()  
    updateStartProgress  
End Sub  
  
Private Sub VBA_Block_9_Fire()  
    updateStartProgress  
End Sub  
  
Private Sub VBA_Block_12_Fire()  
    updateStartProgress  
End Sub  
  
Private Sub VBA_Block_13_Fire()  
    updateStartProgress  
End Sub  
  
Private Sub VBA_Block_14_Fire()  
    updateStartProgress  
End Sub  
  
Private Sub VBA_Block_15_Fire()
```

```
    updateStartProgress  
End Sub
```

```
Private Sub VBA_Block_16_Fire()  
    updateStartProgress  
End Sub
```

```
Private Sub VBA_Block_17_Fire()  
    updateStartProgress  
End Sub
```